Symmetric Cryptography for Long-Term Security

María Naya-Plasencia

MÉMOIRE D'HABILITATION À DIRIGER DES RECHERCHES

Université Pierre et Marie Curie, Paris 6

UFR 919

Spécialité: **Computer Science**

Presented by

María Naya-Plasencia

Inria de Paris

Symmetric Cryptography for Long-Term Security

defended on May 5th, 2017

Rapporteurs

Kaisa Nyberg	-	Emeritus Aalto University, Finland
David Pointcheval	-	CNRS/ENS/Inria, France
Bart Preneel	-	KU Leuven, Belgium

Examinateurs

Anne Canteaut	-	Inria, France
Joan Daemen	-	Radboud University, Netherlands and STMicroelectronics, Belgium
Pierre-Alain Fouque	-	University of Rennes 1, France
Henri Gilbert	-	ANSSI, France
Antoine Joux	-	UPMC, France
Adi Shamir	-	Weizmann Institute, Israel

Acknowledgments

The first words of these acknoledgements could only be for Anne Canteaut. Anne has taught me, among other important things, how to do things right, in all the senses of right. She has constantly been available to help me, always in a selfless way. On top of being a mentor and the ideal colleague, she has also become a friend. Words can not even start to express the amount of gratitude that I have for her.

Next, I'd like to thank all the members of the jury. I am humbled to have such esteemed researchers be a part of it.

I'd like to thank David Pointcheval for having reviewed this manuscript. All the more because it doesn't exactly fall in his main research area, which meant an extra effort. I am very grateful for that, and for all the valuable corrections and comments he sent me.

I have known Kaisa Nyberg since my PhD. I'd like to thank her for many interesting discussions invaluable advices and help, for her interest and passion on new research discussions, as well as for the great and kind welcome she gave me when I went to visit their group for a seminar. Many thanks also for all her pertinent comments on the manuscript that have definitely improved its quality.

Bart Preenel never ceases to amaze me: his day seems to have 100 hours. He is able to do everything, and to do it right. Without his support and altruistic help, the cryptography world would stumble. I am very grateful that he has found the time to be a reviewer of my HDR and for his priceless comments.

I am also extremely grateful to Joan Daemen: for his inspiring talks (I have been using his "stay critical" quote from Asiacrypt 2011 a *lot*), as well as for his stimulating and competition-winning designs, which gave me quite a few sleepless nights. I hope the future is full of permutation-based primitives!

I'd like to thank Pierre-Alain Fouque for many interesting discussions over the years, not always related to research. It is great that Rennes now has a team working also on symmetric cryptography!

I am lucky to have discussed with Henri Gilbert during many research retreats, joint projects and conferences or workshops. Thank you for many enjoyable discussions, for listening so well, and showing always such great interest in new research ideas or topics.

I first met Antoine Joux when he was teaching the Masters algorithms course I attended at Versailles. I found it amazing. I've had the chance to meet him again at many conferences and events and during my post-doc year at Versailles. I enjoyed all the interesting and rich discussions, including the ones about pros and (mostly) cons of lightweight cryptography, or even on how to repair your own washing machine!

I'd like to sincerely thank Adi Shamir for being a part of this jury. Thanks to him and to his recent research results, I have unplugged our smart light bulb. Most of all, I'd like to express my sincere gratitude that someone like him has chosen the amazing field of symmetric cryptography as one of his main research domains. I think that this irrefutably proves that the sometimes-heard qualificative of "bit flippers" does not apply to us.

I am grateful for having had the opportunity of working with all my co-authors. It has been a real pleasure and an extremely positive and profitable experience. A special mention goes to Marine, Christina, François-Xavier, Céline, Benoît, Dmitry, Thomas, Jérémy and Simon for all

the enjoyable discussions, work-related or not.

I am very grateful to Willi Meier for his warm welcome and the time spent with him during my 2-year post-doc, and for staying in touch all these years. His work is always as inspiring, and it is always a pleasure to see him again and catch up.

I have been very lucky working aside Bart Preneel and Florian Mendel and I want to thank them for this great new and exciting experience being the editors of ToSC, the IACR journal on Transactions on Symmetric Cryptology. A lot of work, but definitely worth it!

I have to thank Cryptoexperts for their warm and generous welcome, giving us desks to work next to them, on Fridays, for many years, and for all the interesting meals and conversations: Pascal, Matthieu x2, Cécile, Thomas, Tancrède, Antoine and Louis.

The European Research Council has awarded me a starting grant, QUASYModo, and are trusting me with my research project. I am extremely grateful for that.

In my humble opinion, the SECRET team is the ideal scientific enviroment for working in my field. It is also a warm and welcoming workplace, and coffee breaks can help you become a master of crossword puzzles and foosball. I'd like to thank Pascale, Nicolas, Jean-Pierre and Anne. I know them since my PhD, and their presence makes the cantine of both Rocquencourt and Paris better than a 3 stars restaurant. Since I joined the team, Anthony, André and Gaëtan have also become part of it. I'd like to thank Anthony for his fun comments and discussions, André for being always so positive, it is great to ask for his expertise! As for Gaëtan, I really hope that he will stay in our team: he's a top researcher and would be an invaluable addition to our team. I want to thank the whole team for the perfect atmosphere, for all the smiles, good thoughts, and nice moments together: Yann, Sébastian, Xavier, Rodolfo, Julia, Thomas x2, Kevin, Adrien, Irene, Nicky, Valentin, Matilde, Kaushik, Vivien, Antoine, André, Grégory, Christina, Valentin, Audrey, Joelle, Mamdouh, Marion, Denise, Rafael, Dimitris and Virginie. A big thanks go to Christelle for all her help, disponibility and savoir-faire. Another one goes to all the personnel from Inria Paris (like for instance Cécile, Hubert, Julien, Muriel, Laurence..) for their help and smiles.

Virginie Lallemand defended her PhD in October 2016, and Xavier Bonnetain started his MS internship in March 2016, followed by his PhD. I cannot believe my luck of having found these two amazing students. I want to thank them for everything that I have learned by their side. André Schrottenloher started his MS internship two months ago, and has already impressed me: I am conviced he will be a bright researcher, and I do hope that he will stay with me, as planned, for his PhD. Thank to the three of you for having embraced the subjects with passion and energy, for all the fruitfull discussions and moments shared. I'd also like to thank Chloé and Victoire for doing their internship with me and helping me grow as an advisor.

An important change happened at Inria in 2016: the research center moved from its remote location (Rocquencourt) to the current site in Paris. This changed my life, not only personally (I gained back about 2 hours of commute time per day!), but also professionally: it enabled many research collaborations, helped when inviting experts, finding students... I'd like to thank the Inria's DG in general for this, and Isabelle Ryl in particular.

I thank Andrea, Yann, Gaël, Matthias, Ben, Vadim, Ferielle, Alyssa, Gaëtan, Joana, Marcio,

Paco, Mada, Fred, Anne, Christina, Bea and Chris for all the great moments together, and for many more to come (and thank you Vadim for having the pleasure of coming to my "pot" ;). A special mention for Lore: it is great to have you here, not just because you're family, but because you're the most fun and great person! Thanks to Camille for allowing us to have Wednesday nights out.

Muchas gracias a toda mi familia y la familia de Fab por el apoyo y los buenos momentos (Miguel te echaremos de menos esta vez), y en especial a Élea por haber venido a formar parte de nuestras vidas este año. A mis abuelos les agradezco todos los recuerdos maravillosos vividos con ellos. Las palabras de Buesa suenan a veces, espontáneas, en mi cabeza (...te digo adiós para toda la vida, aunque toda la vida siga pensando en tí..). No os olvidamos, ni olvidaremos.

Quiero finalmente agradecer a mis padres las oportunidades de las que dispuse para elegir mi camino en la vida (que me ha traído hasta aquí), la magnífica escala de valores que me han transmitido, asi como lo cerca que están siempre de nosotros, aunque ojalá lo estuviesen aún más. Sois los mejores abuelos del mundo. A mi hermano Guille le tengo que agradecer el orgullo tan inmenso que me hace sentir el pensar que algún merito tendrá la hermana mayor cuando aparece en el mundo una persona tan excepcional. Ahora que he acabado éste manuscrito deberíamos resucitar a Barba y Eva María! Gracias Patri por formar también parte de nuestras vidas y de nuestra familia, por cuidaros mutuamente y haceros tan felices.

Este documento está dedicado a mis tres tesoros: Olivier, Nicolas y Fabien. Dos pequeños, aunque no por eso menos importantes, y uno grande. Gracias a los tres por hacer que mi vida me parezca tan absolutamente maravillosa. Fab: contigo a mi lado, todo es posible. Oli y Nico: sois, sin comparación posible, lo mejor que he hecho, no ya en estos últimos 8 años, sino jamás.

Résumé en français

Les résultats que je présente dans ce manuscrit sont la continuation logique de la recherche entamée pendant mon doctorat. J'ai continué à m'intéresser aux sujets de conception et cryptanalyse des primitives symétriques, mais ma recherche s'est aussi approfondie dans plusieurs directions:

J'ai proposé trois nouvelles primitives, chacune dans un scénario différent en demande de primitives symétriques: primitives à bas coût, primitives faciles à masquer et primitives pour FHE (*Fully Homomorphic Encryption*, la cryptographie homomorphique). Ces primitives sont Quark [AHMNP10, AHMN13], Zorro [GGNPS13] et Kreyvium [CCF+16] respectivement. Quark et Kreyvium ont été distingués comme étant dans les trois meilleurs articles des conférences CHES10 et FSE16, respectivement.

J'ai proposé quelques algorithmes qui permettaient de réduire considerablement la complexité d'un grand nombre d'attaques par rebond. J'ai ensuite généralisé ces algorithmes, et ils ont pu trouver de nombreuses autres applications.

Dans le cadre de mon projet personel de recherche, je me suis attelée à la généralisation et l'amélioration des familles connues de cryptanalyse. La technicité des attaques ne permet pas, dans la plupart des cas, une bonne comprehension des outils cryptanalytiques utilisés, ce qui les rend difficiles à vérifier, et malheureusement, implique qu'il existe des erreurs publiées. Nous avons généralisé et amélioré significativement plusieurs familles.

Ces deux dernières années j'ai commencé à m'intéresser aux effets qu'un ordinateur quantique aurait sur la cryptographie symétrique. J'ai été surprise par le peu de travail de recherche effectué sur les attaques symétriques quantiques, car c'est le seul moyen qu'on a d'avoir confiance dans les primitives que nous utilisons. J'ai reçu une bourse européenne (ERC starting grant), QUASYModo, qui commencera en septembre 2017. J'ai déjà obtenu quelques résultats préliminaires encourageants.

Par ailleurs, j'ai continué à travailler sur les attaques dediées, et trouvé des nouvelles attaques. Ces résultats ont été publiés dans [BLNS16, KLLN16b, JNP14, LN15a, CLN15, LN15b, CFG+15, JNP13, NPP12a, JNPP12a, NPTV11, ABNP+11a, MNPP11, JNPS11, ANPS11, NPRM11, KNPRS10, GLM+10].

Conception

De nouveaux besoins ont récement apparu pour des primitives symétriques. On peut citer par exemple le besoin de cryptographie à bas coût [BLP+08], de primitives facile à masquer [PRC12], ou encore de primitives pour la crypographie homomorphique [ARS+15]. Grâce à l'expérience acquise lors de mes travaux de cryptanalyse, j'ai pu contribuer à la conception de telles primitives de ce type, et proposer de nouvelles directions pour chaque cas.
Quark [AHMN13] est une fonction de hachage à bas coût que nous avons propsé à CHES 2010. Malgré l'attention poussée qu'elle a reçu depuis, elle reste très sûre (grande marge de sécurité),

performante, et a inspiré de nombreux cryptosystèmes ultérieurs (198 citations pour l'instant). L'article associé fut élu parmi les 3 meilleurs de la conférence, et benéficia d'une soumission invitée au JoC (Journal of Cryptology), publié en 2013.
Après avoir cherché une bonne primitive symétrique pour la cryptographie homomorphique, nous avons proposé dans [CCF+16] un nouveau chiffrement à flot à très faible profondeur multiplicative. Ses performances sont bonnes, et sa sécurité reste intacte depuis sa publication, contrairement à la plupart des chiffrements concurrents. L'article associé fut élu parmi les 3 meilleurs de FSE 2016, et bénéficia également d'une soumission invitée au JoC (en cours de revue).
Dans [GGNPS13] nous avons exploré de nouveaux chiffrements faciles à masquer, en nous inspirant du chiffrement par flot AES. Nous en avons produit un, risqué car provocateur (par le peu de marge qu'il offrait), qui fut cassé par la suite. Mais l'interêt éveillé par notre construction a provoqué un éveil et un engouement de la communité, et des études ont montré que la faiblesse de notre construction était due au choix des paramètres, et pouvait se réparer facilement. Cela a aboutit à un nouveau type de construction, le PSPN [BDD+15] (Partial Substitution-Permutation Network). Certaines de ces constructions sont toujours considérées comme sûres et performantes.

Algorithmique: fusion de listes par rapport à une relation

En travaillant sur la généralisation et l'amélioration de familles de cryptanalyses, j'ai identifié un problème récurrent en cryptographie symétrique, qui était souvent l'élément dominant de la complexité des attaques, et n'était pas résolu de manière optimale. Ce problème est la fusion de listes sujettes à une relation: étant données N listes d'éléments d'un ensemble E, et une relation $\mathcal{R} : E^N \to 0, 1$, nous voulons obtenir tous les N-uplets d'éléments (des N listes) vérifiant R. Ce problème apparaît notamment dans plusieurs attaques par rebond, qui étaient les plus utilisées contre les fonctions de hachages candidates de la compétition SHA-3. Dans [NP11] j'ai proposé plusieurs algorithmes génériques qui réduisent la complexité de beaucoup de ces attaques par rebond. J'ai ensuite donné une extension de ces algorithmes dans [ABNP+11b], ainsi que de nouvelles applications [NPTV11, JNPS11, JNPP12a] dans le contexte des attaques par rebond. Dans [NPRM11, LN15b, LN15a], nous avons appliqué ces algorithmes à d'autres scénarios de cryptanalyse, réduisant la complexité des attaques étudiées. Nous avons encore exhibé une autre application dans [CNPV13], sur les attaques par le milieu *meet-in-the-middle*.

Généralization de familles de cryptanalyse

La cryptanalyse a récemment été l'objet d'un grand nombre d'avancées. De nouvelles applications sont apparues, comme les attaques par rebond, les *cube attacks*... Dans la plupart des cas, ces nouvelles techniques sont introduites en ciblant un cryptosystème spécifique, et sont décrites comme des techniques ad-hoc, ce qui les rend difficiles à généraliser. La complexité technique inhérente au cryptosystème ciblé cache souvent les idées principales sous-jacentes de ces innovations, et les rend difficiles à maitriser, à adapter et à optimiser: ce n'est, en général, pas fait. Il y a donc un réel besoin de généralisation de ces attaques. J'ai pris l'initiative de travailler dans cette direction: généraliser de façon systématique les techniques de cryptanalyse.

Par exemple, la cryptanalyse qui utilise des différentielles impossibles [BNS14, BLNS16], différentielles conditionelles [KMNP10], attaques par le milieu [CNPV13], attaques par corrélation [CN12] et les attaques "multiple limited birthday" [JNP13] peuvent maintenant être appliquées de façon quasi-automatique, et avec des complexités optimisées. De plus, grâce à la version complète et simplifiée de ces attaques, de nouvelles idées pour les améliorer ont été trouvées, ce qui permet de construire des attaques encore plus puissantes. Dans ce manuscrit, on évoquera les idées principales de nos généralisations et des améliorations d'attaques par differentielle impossible, attaques par le milieu, et attaques (tronquées) differentielles.

Cryptanalyse symétrique post-quantique

RSA [RSA78] est l'algorithme cryptographique asymétrique (à clé publique) le plus populaire aujourd'hui. Sa sécurité étant fondée sur la difficulté du problème de la factorisation des grand entiers, il serait gravement compromis par l'arrivée de l'ordinateur quantique. En effet, dans les années 90, Shor [Sho97] a proposé un algorithme qui résout le problème de la factorisation discréte et du logarithme discret en temps polynomial avec un ordinateur quantique. Récemment, la cryptographie post-quantique est en train de vivre un "boom" impressionant. Des sujets très en vogue dans la communauté cryptographique sont par exemple la cryptographie basée sur les treillis (*lattices*), la cryptographie multivariée, ou celle basée sur les codes. Leur sécurité est censée resister dans un monde post-quantique (c'est-à-dire où l'ordinateur quantique est une réalité), car ils ne reposent par sur la théorie des nombres. Mais leur performance et applicabilité ne sont pas encore au niveau de RSA. L'institut américain des standards et technologie (NIST), qui choisit la plupart des standards mondiaux, est très concerné par ce sujet et cherche activement chercher des alternatives, comme le montrent les appels à candidats récents.

La situation de la cryptographie symétrique est bien différente. Jusqu'à présent, dans un contexte post-quantique, les cryptographes se sont principalement intéréssés à la sécurité des primitives symétriques "idéales" (donc théoriques). Le résultat principal est l'algorithm de Grover [Gro96], qui permet de chercher une base de données de taille N avec un coût en temps de $O(N^{1/2})$ en utilisant un ordinateur quantique: il peut être appliqué à toute recherche exhaustive, en réduisant ainsi le temps par une racine carrée. Ce qui est considérable, mais reste nettement moins problématique que ce que RSA subirait: il suffirait en effet de doubler la taille des clés secrètes utilisées en cryptographie symétriques pour contrebalancer les effets de cet algorithme quantique. Par défaut d'autre résultat, la communauté cryptographique s'est relativement désintéressée du sujet, restant sur le consensus que doubler la longueur de clé (ou de hachage) serait assez pour continuer à avoir des algorithmes sûrs [BBD09].

Quelques résultats récents semblent néanmoins indiquer le contraire, et il y a deux ans j'ai commencé à étudier en détail les conséquences de l'existence d'ordinateurs quantiques pour la cryptographie symétrique. J'ai obtenu pour le moment trois résultats importants. Le premier, publié à Crypto 2016 [KLLN16a], montre que dans certains scenarios, certaines primitives symétriques sûres (en "classique", c'est-à-dire non-quantique) peuvent devenir complètement cassées face à un adversaire quantique. Le second, publié le IACR ToSC journal [KLLN16b],

propose des versions quantiques des attaques différentielles et linéaires, ainsi que des exemples contre-intuitifs d'applications. Enfin, le troisième, actuellement en soumission [BNP17], étudie l'effet d'une contremesure proposée pour les attaques précedentes. Ces trois résultats montrent d'ores et déjà que beaucoup de travail doit encore être réalisé : des constructions solides et sûres dans le monde classique peuvent devenir complètement cassées (comme l'est RSA) dans le monde post-quantique.

Attaques dédiées

Le besoin émergent de primitives symétriques telles que celles décrites en section 1.2.4.2 ont généré l'apparition d'un grand nombre de constructions innovantes.

Par exemple, il existe une forte demande, émanant autant de la communauté cryptographique que de l'industrie, de primitives à bas coût (voir [BLP+08]), qui ont souvent une marge de sécurité réduite. Cette demande a provoqué l'apparition d'un énorme nombre de nouveaux candidats prometteurs, chacun avec ses propres qualités liées à l'implantation.

Quelques exemples sont PRESENT [BKL+07b], CLEFIA [SSA+07], KATAN/KTANTAN [CDK09a], LBlock [WZ11], TWINE [SMMK12], LED [GPPR11], PRINCE [BCG+12], KLEIN [GNL11], Trivium [CP08] et Grain [HJM07].

Le besoin d'avoir une recommandation claire pour un chiffrement à bas coût implique faire un énorme tri parmi tous ces candidats potentiels.

Dans ce contexte, le besoin d'un effort cryptanalytique significatif est évident. Ceci a été prouvé par l'énorme nombre d'analyses de sécurité apparu sur les primitives précedentes (pour citer quelques exemples: [LAAZ11, BKLT11, MRTV12, NWW13, CS09, BR10, TSLL11]).

Idéalement, les concepteurs auraient dû déjà bien analyser la ou les primitives qu'ils proposent vis-à-vis des attaques connues [1] On doit donc trouver des nouvelles attaques, spécifiques aux primitives ciblées, pour s'adapter à ces nouvelles constructions. Citons l'exemple de PRINTcipher: malgré sa ressemblance avec PRESENT, qui est un chiffrement sûr, cette variante est maintenant cassée grace à des nouvelles attaques dédiées. Quelques-uns de mes papiers sélectionés decrivent des attaques dédiées :

Par rapport aux fonctions de hachage :

1. SHAvite-3-256 [MNPP11](meilleur connu) et 512 [GLM+10] (fonction de compression complète)
2. Luffa [KNPRS10] (meilleur connu)
3. ECHO [JNPS11] (meilleur connu, 7/8 tours de la fonction de compression)
4. Grøstl [JNPP12b](meilleur connu, 9/10 tours de la permutation)
5. Keccak [NPRM11] (première cryptanalyse pratique avec résultats sur 3/24 tours).

[1] Ce n'est pas toujours le cas malheureusement. Souvent, les attaques ne semblent pas applicables de façon évidente à cause du manque de techniques vraiment généralisées.

Par rapport aux chiffrements :

1. Klein [ANPS11, LN15b] (chiffrement cassé)
2. Sprout [LN15a] (chiffrement cassé)
3. Armadillo2 [ABNP+11b, NPP12b] (chiffrement cassé)
4. PRINCE [CFG+15] (meilleure attaque connue)
5. PICARO [CLN15] (attaque à clé liée sur tout le chiffrement)

General Introduction

The results presented in this manuscript are a logical continuation of the research embarked during my PhD. I have continued to work on the important problem of design and analysis of symmetric primitives, but my research has taken a deeper twist in several directions:

I have considered the three most studied scenarios where new symmetric primitives designs are needed: lightweight hash functions, easy-to-mask primitives and HFE-friendly primitives. I have proposed three new designs of symmetric primitives, one for each scenario: Quark [AHMNP10, AHMN13], Zorro [GGNPS13] and Kreyvium [CCF+16] respectively. Quark and Kreyvium received one of the three best papers distinctions of conferences the CHES10 and FSE16 respectively.

In the context of rebound attacks, I have proposed some algorithms [NP11] that allowed to considerably improve the previously best known complexities. These algorithms consider and solve a quite generic problem: the list merging with respect to a known relation. These algorithms have found many other applications as the ones I presented in [NPTV11, ABNP+11a, LN15a, JNPP12a, JNPS11, CNPV13], where I have been able to propose solutions to problems that we did not know how to solve before.

Particularly important is the task I have chosen of generalizing and improving known families of cryptanalysis. Indeed, the technicality of the attacks does not allow, most of the time, a perfect understanding of the available cryptanalysis tools, which implies cryptanalysis hard to verify and, unfortunately, many published errors. Providing generalized expressions for the complexities of these sophisticated attacks allows to apply them in a semi-automated way, avoiding mistakes, and allowing a better understanding. In all the cases this scenario has enabled us to propose significant improvements of previous families of attacks. Some of these examples include the generalizations of impossible differential attacks [BLNS16, BNS14] including the improvement of the state-test technique, of meet-in-the-middle attacks and bicliques [CNPV13] with the improvement of the sieve-in-the-middle technique, of correlation attacks on the combinator generator [CN12] with an algorithm for decreasing the time complexity and the required amount of data, conditional differential cryptanalysis [KMNP10, KMN11] and differential and truncated differential attacks generalization in order to provide a quantum version of them [KLLN16a].

Naturally, I have also continued working on dedicated cryptanalysis, discovering attacks on primitives or improved attacks on reduced-round versions of a large number of constructions. These results have been published in [BLNS16, KLLN16b, JNP14, LN15a, CLN15, LN15b, CFG+15, JNP13, NPP12a, JNPP12a, NPTV11, ABNP+11a, MNPP11, JNPS11, ANPS11, NPRM11, KNPRS10, GLM+10].

These last two years I have started getting interested in the implications that a quantum computer would have on symmetric cryptography. I found surprising that not much research

had been done in the direction of quantum symmetric attacks, as this is the way we have of obtaining confidence in primitives. I have been awarded an ERC starting grant, QUASYModo, that will start in September 2017. So far I have already obtained some preliminary results: In [KLLN16b] we proposed quantized versions of differential and linear attacks as well as some counter intuitive examples. The surprising result from [KLLN16a] showed that, in some scenarios, some symmetric constructions could become completely broken by a quantum adversary, with an exponential speedup compared to classical attacks. Recently, we extended this work in [BNP17], analyzing in detail the effects of a proposed countermeassure for preventing the previous attacks, based on replacing xors by modular additions.

Contents

Chapter 1

Introduction

Contents

In this chapter we will introduce the basic notions that describe the context of my research. The main topic is symmetric cryptology, one of the two big branches in cryptology. We will describe the most important constructions in this family, its properties, advantages, disadvantages and modus operandi. For understanding the historical scientific context of the research described in this manuscript, we have to present the state-of-the-art on symmetric cryptography, describing the series of competitions that have been launched in the last two decades and the implications of this on modern symmetric primitives. We will end the introduction by describing some of the current hot topics in this field and how they are relevant to this work.

1.1 Cryptology

Cryptology, whose main objective is to protect information against malicious users, can be decomposed into two main branches: Asymmetric cryptography, where the parties that communicate do not need to share a common secret in advance (e.g. RSA), and symmetric cryptography, where a secret needs to be shared prior to communication (e.g. AES), but which has much better performance and smaller implementations.

The paramount importance of cryptology is widely accepted: more and more communications in all domains are encrypted and secured to ensure their confidentiality and authenticity. These communications are nearly always encrypted with symmetric cryptography, which is much more performant and more suitable for lightweight environments with limited computational capacity (such as IoT devices), while asymmetric cryptography is typically used to perform the secret key exchange to initiate the communication. Asymmetric and symmetric cryptography complement each other perfectly to solve most of the current needs. My research is centered on symmetric cryptanalysis.

Cryptography and cryptanalysis We call cryptography the study of techniques for establishing secure communications, and cryptanalysis consists in studying the previous constructions

in order to detect potential flaws. Simplifying, we can say that cryptography's aim is to design primitives, and cryptanalysis tries to "break" them. It is hard to clearly separate both disciplines, as one cannot conceive a secure primitive if one has not tested its resistance before.

1.2 Symmetric cryptology

Applications of symmetric cryptography are vital in the Information Age. There exist three main types of primitives in symmetric cryptography: block ciphers, stream ciphers and hash functions. While the first two belong to symmetric cryptography by definition, *i.e.* they need a secret key for encryption, the third one does not require a key, but belongs to this category due to the similarity of the used transformations. The tools for analysing the three families are often similar and common.

1.2.1 Three main families

We provide here a description of each one of these families.

1.2.1.1 Block ciphers

Block ciphers encrypt a message by decomposing it into blocks of a fixed size n. Each block is transformed through the same parametrized permutation, where the parameter is the secret key k. The encrypted block is the output of the permutation. To decrypt, the inverse of the permutation must be applied to the ciphertext with the same key, in order to recover the plaintext.

Types. As we will see, block ciphers are typically composed of the iteration of several similar rounds. Several categories for such constructions exist, including substitution permutation networks (SPN), addition-rotation-xor (ARX), or Feistel networks.

Operation modes. In order to securely encrypt messages, block ciphers must be used with secure modes of operation. One of the constraints is to not allow an attacker to indentify when the same two blocks have been encrypted under the same key, without having to change the key for each block (which is a non-negligible operation). Some popular modes are Cipher Block Chaining, CBC [EMST76], or Counter Mode, CTR [LRW00]. It is also possible to build authenticated encryption primitives by using authentication modes, as the Offset Codebook Mode, OCB [KR11] proposed by Krovetz and Rogaway.

The AES. The AES (earlier known as *Rijndael*) was designed by Daemen and Rijmen [DR02]. The encryption transformation is composed of 10 to 14 rounds depending on the size of the key. It operates on message blocks of 128 bits, that can be seen as a matrix of 4×4 bytes. One round is composed of four transformations. In SubBytes (`SB`), a single 8-bit S-box is applied 16 times in parallel to each byte of the state matrix. In ShiftRows (`SR`), the 4 bytes in the ith row of the state matrix are rotated over i positions to the left. In MixColumns (`MC`), a linear transformation defined by an MDS matrix is applied independently to each column of the state

matrix. Finally, in AddRoundKey (AK), a 128-bit subkey provided by the key scheduling is added to the internal state by an exclusive or.

1.2.1.2 Stream ciphers

Stream ciphers combine the plaintext on the fly, typically bit by bit, with a secret sequence. For instance, in an additive synchronous cipher, each plaintext bit is combined through a XOR with a binary secret sequence of the same length as the message (the keystream) in order to generate the ciphertext. The one-time-pad corresponds to the case where the keystream is a secret and random sequence shared by both parties. This algorithm offers perfect secrecy [Sha49] but is highly impractical as we need to share a key as long as the message. Then, in most practical cases, the keystream is a pseudo-random sequence, that has been produced from a short secret seed, the master key, by a pseudo-random generator. The keystream must not be distinguishable from a truly random sequence unless the secret key is known. To decipher, sharing the small seed allows to generate the same pseudo-random sequence, and by combining it with the ciphertext, the plaintext can be recovered.

Typically, the pseudo-random generator is not only initialized from a secret key, but also from a public value, the IV, that allows us to reinitialize the generator without needing to change the secret key.

1.2.1.3 Hash functions

A hash function is a function H that, given a message M of an arbitrary length, returns a value of a fixed length $H(M) = h$. They have many applications in computer security, as in message authentication codes, digital signatures and user authentication. We require them to be easy to compute, and to verify some particular properties, the three most important are:

- Finding two messages M and $M' \neq M$ such that $H(M) = H(M')$ must be "hard". If this is true, H is collision resistant.
- Given a message M and its hash $H(M)$, finding another message M' such that $H(M) = H(M')$ must be "hard". In this case we say that H is second-preimage resistant.
- From a hash h, it must be "hard" to find a message M so that $H(M) = h$. If this is verified, H is preimage resistant.

In the next section we explain what "hard" means, or how it can be interpreted.

Types. There are several possible classifications for hash functions. Regarding the operating mode, they can be for instance Merkle-Damgård or sponges. We will describe the latter construction in the next section.

1.2.2 Security offered by symmetric primitives

Let us first describe some cryptanalysis scenarios and, then define what is expected from a secure primitive in symmetric cryptography: A primitive is considered "unbroken" if no attack "better"

than generic attacks exists (generic attacks are those that we can always apply, even to ideal primitives). The primitive is considered "broken" otherwise (it can be theoretically or practically broken, depending on whether the attack is implementable or not). The most common consensus on the definition of "better" is "needing less computation". Ideal and unbroken symmetric primitives have their security defined by the most performant generic attacks, which are usually directly related to the length of a parameter (the key for ciphers, the digest for hash functions). Dedicated attacks used to analyze the security of concrete instances of primitives are often well-known families of algorithms, sometimes more complex and less understood variants and improved versions of these families, and occasionally new and dedicated procedures.

1.2.2.1 Cryptanalysis scenarios

Let us briefly describe here some of the most popular cryptanalysis scenarios that are related to the research presented in this manuscript.

Classical scenarios. The most classical and realistic scenarios when analyzing symmetric ciphers are known-plaintext attacks and chosen-plaintext attacks. In these scenarios, the attacker is supposed to know some plaintext-ciphertext pairs obtained with a single secret key (where she might or might not have chosen the plaintext, depending on the model), and tries to recover information on the key.

Related-key attacks. In contrast to single-key attacks, related-key attacks consider a scenario where one or several plaintexts have been encrypted under two different and secret keys, that verify a known relation (often they sum to a known value). Though these scenarios are less realistic than single-key ones, in some contexts they have an enormous importance, as if a block cipher is used as a compression function, for instance, the role of the secret key might be taken by a chosen message block. Knowing the limitations of use of our ciphers is of main importance, as they can be used in various contexts.[1]

Quantum adversaries. Attacks on primitives might be designed in a classical scenario, where the attacker has access to classical computers, or also in a quantum one, where she can take advantage of a quantum computer. More information on quantum attacks is given in Chapter 5

1.2.2.2 Security requirements

Under equal conditions, we tend to prefer the primitives that remain secure in all settings. Even if the attacks are impractical, they only get better. Improvements can be found over the years, or some applications might misuse the primitives, not respecting the specifications. We prefer extensively examined primitives with no known attacks, in any of the settings.

[1] Known-key attacks, an even less realistic scenario for attackers, aim at detecting non-random properties of the functions, and might, in some scenarios, become weaknesses.

1.2.2.3 Ciphers

If we consider any cipher that uses a secret key of length $|k|$, we can always perform an exhaustive search of the key with a cost of $2^{|k|}$ calls to the cipher, that will allow us to retrieve the correct key. Typical values of $|k|$ are 80, 128 or 256 bits.

Secure ciphers must also resist to other types of attacks, such as distinguishers: that means that the output of a block cipher or a keystream must be indistinguishable from a random sequence if the secret key is not known. These attacks are a priori less devastating than key-recovery ones, but they also form a threat: for instance in the stream cipher setting, they could allow an attacker who knows that one out of two possible messages will be sent, to correctly guess which one has actually been sent.

1.2.2.4 Hash functions

Defining "hard" in order to determine the resistance to the attacks is hard to do formally and no consensus exists [Rog06]. We can consider here a strict meaning as we did for ciphers.[2] For instance, if we consider an iterative hash function, the most common case, we can define it by a compression function taking as input a chaining value and a message block, and also by a mode of operation that describes how to iterate the compression function until all the blocks of the message have been introduced. The hash length being $|h|$, by the birthday paradox, we can obtain a collision with a generic cost of $2^{|h|/2}$ calls to the compression function [Yuv79]. The strict meaning will imply that a function is secure while no attack requiring fewer compression function calls exists. For preimage or second preimage resistance, the generic attack is an exhaustive search, that costs $2^{|h|}$ calls to the compression function, and as long as no attack with fewer calls is known, the function is considered resistant to (second) preimage attacks.[3] Typical values of $|h|$ are 256 or 512 bits. Let us point out here that for the particular case of sponge functions, some other attacks apply leading to a redefinition of security that we will present in the next section.

1.2.3 Importance of cryptanalysis

The security of asymmetric primitives typically relies on the hardness of a well-established mathematical problem (e.g. integer factorization), which is then accepted as hard by the community. In contrast, the security of symmetric primitives is much less clearly established, and the existing *pseudo-security-proofs* always rely on ideal modelizations that are far from realistic (for example, modeling pseudo-random distributions by truly random ones). We are then often left with an *empirical* measure of the security, provided by a thorough (and, even more importantly, never-ending) study of symmetric primitives by cryptanalysts. Indeed, AES can be considered secure only because people have been and remain sceptical, and the security of AES is still under constant scrutiny.

That is why confidence on symmetric primitives is always based on the amount of cryptanalysis they have received, and on the security margin that they have left. It is crucial that

[2] During the SHA-3 competition, sometimes a weak meaning was considered, where the complexity measure to compare with the generic attack was the product of time and memory.

[3] For long messages and small internal states, better generic attacks in second preimage exist.

the cryptanalysis toolbox be continuously improved over the years.

It is important to note that attacks on reduced-round versions, in particular when attacks on full versions are not known, are of the most significance, as they define the remaining security margin. This margin is usually measured by the number of rounds covered by the "best attack". Often, the security margin provides a good measure of how far a primitive is from being broken.

1.2.4 State of the art

Symmetric cryptography has made substantial progress during the last two decades. The main reason maybe the large number of competitions for finding new standards and recommendations. Design expertise has been gained, but more importantly, the enormous effort the community has put into evaluating all these candidates has greatly contributed to the knowledge of new cryptanalytic techniques.

Some other topics have gained notoriety, though no competition has been launched yet, and there is an increasing interest in the community as it can clearly be seen from the publications or the NIST[4] organized workshops,

1.2.4.1 Competitions for finding new primitives

We will briefly enumerate here the most important cryptographic competitions that took place lately. The enormous amount of work that the community has spent on these competitions, has definitely contributed to a faster development of our research field.

AES. This competition, launched by NIST between 1997 and 2000, aimed at finding a new encryption standard to replace DES [Tuc97]. In total, 15 algorithms were submitted. The cipher Rijndael [DR00] won the competition, becoming the AES.

NESSIE. European project whose aim was to find recommended cryptographic primitives during the years 2000-2003. Three block ciphers were selected in the end. No stream cipher resisted cryptanalysis efforts, leading to the eSTREAM project.

eSTREAM. The European Network of Excellence ECRYPT launched eSTREAM to recommend stream ciphers for use, where 34 primitives were submitted in 2005. The portfolio was published in 2008, and currently contains a total of 7 primitives.

SHA-3. Due to the attacks [WY05, WYY05] discovered on MD5 [Riv91] and SHA-1 [NISa] [5] , the confidence on SHA-2 [NISb] was also undermined, and NIST launched another competition to find a new hash standard between 2008 and 2012. NIST retained 56 submissions, and chose Keccak [BDPA13] as the new hash function standard.

[4] Unites States National Institute of Standards and Technology

[5] Very recently, the first practical attack on SHA-1 has been found:https://shattered.it/#

CAESAR. In collaboration with NIST, Bernstein launched in 2014 the CAESAR competition for authenticated encryption. There were 55 proposals submitted to this ongoing competition.

1.2.4.2 Current hot topics and important problems

There are three largely studied scenarios where new symmetric primitives designs are needed: lightweight primitives, easy-to-mask primitives and HFE-friendly primitives.

Lightweight Cryptography. Let us point out that, while symmetric constructions allow for much more efficient and compact implementations than asymmetric primitives, the community has repeatedly expressed a need for significantly more lightweight and efficient symmetric algorithms (for instance, the NIST Lightweight workshop[6] or [BLP+08]). For example, the block cipher standard AES-128 (2400GE) is too "big" for some current real life applications, such as RFID tags or sensor networks. In recent years an enormous number of promising lightweight primitives has been proposed. The strong demand from industry for clearly recommended lightweight ciphers requires us to narrow down the large number of these potential candidates by cryptanalysis. While these functions are more compact or performant than standard AES, they suffer from a reduced security due to reduced key sizes (and reduced needs due to applications), typically from 128 bits. Since the trade-off between performance and security is a major issue for lightweight primitives, it is also very important to estimate the security margin of these ciphers, to determine for instance if some rounds need to be added, or if some can be omitted to achieve a given security level.

Easy-to-mask. A natural concern coming along with the development of lightweight cryptography is the concern of side-channel attacks (SCAs): since lightweight ciphers are dedicated to small embedded devices their implementations are indeed an attractive target. Then, some new designs such as the block ciphers PICARO [PRC12] or Zorro [GGNPS13] tend to address both problems together by proposing ciphers fitting some requirements of lightweight cryptography and being easy to protect against SCA. This last point corresponds mainly to limiting the number of non-linear operations, since they are hard to protect and induce important extra costs.

Low multiplicative depth. In 2009, Gentry [14] realised a notable advance in asymmetric cryptography by proposing the first fully homomorphic encryption (FHE) scheme, a solution that allows to delegate computations to a server without revealing any information on the data, but all these schemes proposed so far suffer from a huge ciphertext expansion. A good solution in order to be efficient, is to encrypt the data with a symmetric algorithm before sending it to the server, as we describe in the next chapter. Once again the constraints imposed on the symmetric algorithm in this setting are quite strict and differ from the ones that have ruled the AES design. Here, the important metrics are the multiplicative size and the multiplicative depth. Some examples are Kreyvium [CCF+16], LowMC [ARS+15] or FLIP [MJSC16].

[6] http://www.nist.gov/itl/csd/ct/lwc_workshop2015.cfm

Chapter 2

Design of symmetric primitives

Contents

New needs for cryptographic primitives have recently appeared. We can mention in particular lightweight cryptography [BLP+08], easy-to-mask primitives [PRC12] and symmetric cryptography for homomorphic encryption [ARS+15]. Thanks to all the insight acquired while working on cryptanalysis, I have been able to work on the design of these new types of primitives and to propose some new directions for each of them.

Quark [AHMN13] is a lightweight hash function that we have proposed at CHES 2010. It has received much attention, remains soundly secure (big security margin), performant and has inspired many later designs (198 citations in total). It received one of the 3 best paper awards of CHES 2010 and was invited for submission to the Journal of Cryptology (JoC).

Considering the problem of designing a good symmetric primitive for fully homomorphic encryption, we proposed in [CCF+16] a new stream cipher with a very low implementation depth. The performance of this cipher are good, and it remains secure since its publication, contrary to most of the ciphers proposed for this setting. It was one of the three best papers of FSE 2016 and invited for submission to the JoC.

In [GGNPS13] we explored new ciphers that would be easy to mask, basing the core idea on the AES block cipher. We gave a risky and challenging instantiation that was afterwards broken. The interest that this construction has generated in the community and the generated studies have shown that the weakness was due to the chosen parameters and it was easy to repair. It has led to a new type of construction, the PSPN [BDD+15]. Some instantiations of this construction are still considered secure and performant.

2.1 Quark

2.1.1 Sponge construction

Hash function Quark uses the sponge construction. As described in [BDPA08], this construction is parametrized by a *rate* (or block length) r, a *capacity* c, and an *output length* n. The *width* of a sponge construction is the size of its internal state $b = r + c$. A representation of this construction can be found in Fig. 2.1. A permutation P on b bits is used.

Given a fixed initial state, the sponge construction processes a message m in three steps:

1. **Initialization**: the message is padded by appending a '1' bit and as many zeroes as needed to reach a length multiple of r.

2. **Absorbing phase**: each r-bit message block is xored into the top r bits of the state interleaved with applications of the permutation P.

3. **Squeezing phase**: the top r bits of the state become part of the output, interleaved with applications of the permutation P, until n bits are returned.

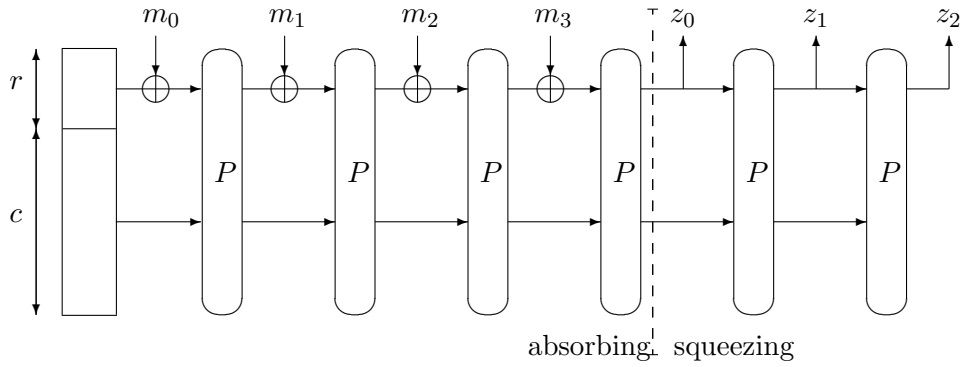


Figure 2.1: The sponge construction used in Quark, with an example of a 4-block message.

2.1.2 Permutation

Quark's permutation P is based on the stream cipher Grain and on the block cipher KATAN. It is generically represented in Fig. 2.2.

The permutation P uses three non-linear Boolean functions f, g, and h, a linear Boolean function p, that are applied to the internal state, composed at time t of:

- an NFSR X of $b/2$ bits $X^t = (X_0^t, \ldots, X_{b/2-1}^t)$;
- an NFSR Y of $b/2$ bits $Y^t = (Y_0^t, \ldots, Y_{b/2-1}^t)$;
- an LFSR L of $\lceil \log 4b \rceil$ bits $L^t = (L_0^t, \ldots, L_{\lceil \log 4b \rceil - 1}^t)$.

Permutation P processes a b-bit input in three stages, as follows:

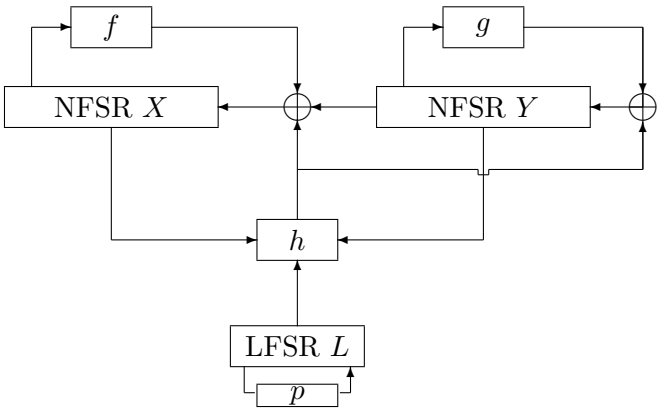


Figure 2.2: Diagram of the permutation of QUARK.

Initialization. Upon input $s = (s_0, \ldots, s_{b-1})$, P initializes its internal state as follows:

- X is initialized with the first $b/2$ input bits: $(X_0^0, \ldots, X_{b/2-1}^0) := (s_0, \ldots, s_{b/2-1})$;
- Y is initialized with the last $b/2$ input bits: $(Y_0^0, \ldots, Y_{b/2-1}^0) := (s_{b/2}, \ldots, s_{b-1})$;
- L is initialized to the all-one string: $(L_0^0, \ldots, L_{\lceil \log 4b \rceil - 1}^0) := (1, \ldots, 1)$.

State update. From an internal state (X^t, Y^t, L^t), the next state $(X^{t+1}, Y^{t+1}, L^{t+1})$ is computed by *clocking* the registers as follows:

1. The function h is computed on bits from X^t, Y^t, and L^t, leading to:

$$h^t := h(X^t, Y^t, L^t) \ ;$$

2. X is clocked and the feedback bit is computed using Y_0^t, the function f, and h^t:

$$(X_0^{t+1}, \ldots, X_{b/2-1}^{t+1}) := (X_1^t, \ldots, X_{b/2-1}^t, Y_0^t + f(X^t) + h^t) \ ;$$

3. Y is clocked and the feedback bit is computed using the function g and h^t:

$$(Y_0^{t+1}, \ldots, Y_{b/2-1}^{t+1}) := (Y_1^t, \ldots, Y_{b/2-1}^t, g(Y^t) + h^t) \ ;$$

4. L is clocked using the function p as feedback function:

$$(L_0^{t+1}, \ldots, L_{\lceil \log 4b \rceil - 1}^{t+1}) := (L_1^t, \ldots, L_{\lceil \log 4b \rceil - 1}^t, p(L^t)) \ .$$

Computation of the output of P. After the initialization, QUARK updates the state $4b$ times, and the output is the final value of the NFSR register's X and Y, using the same bit ordering as for the initialization.

Rationale. As long as the bits X_0 and Y_0 affect linearly the feedback functions, P will be a permutation.

When designing the involved Boolean functions, we decided to borrow the following features from Grain-v1:

- A mechanism in which each register's update depends on both registers.
- Boolean functions of high degree and high density.

From KATAN, we chose to reuse:

- Two NFSRs instead of an NFSR and an LFSR; as for hashing we do not need to ensure a long period.
- An auxiliary LFSR to act as a counter and to avoid self-similarity of the round function.

Furthermore, we aimed to choose the parallelization degree as a reasonable trade-off between performance flexibility and security. The number of rounds was chosen high enough to provide a comfortable security margin against future attacks.

We first chose the functions in QUARK according to their individual properties (nonlinearity, resilience, algebraic degree and density). The final choice was made by observing the empirical resistance to known attacks. The distinct taps for each register break the symmetry of the design. As h function we use a function of lower degree than f and g, but with more linear terms to increase the cross-diffusion between the two registers.

The taps of f and g, which correspond respectively to indices within the X and Y registers, were chosen with respect to criteria both analytical (invertibility, irregularity of intervals between two consecutive taps) and empirical (measured diffusion and resistance to cube testers and differential attacks). For h, and contrary to Grain, taps are distributed uniformly in X and Y.

Preliminary cryptanalysis. We took into account the most performant attacks on Grain and KATAN in order to correctly choose the functions and parameters. For instance, we applied cube testers, differentials and conditional differentials in order to adjust the total number of rounds. The biggest number of attacked rounds was a 23% of the total. To the best of my knowledge, these results appearing in the extended version [AHMN13] remain the best ones known on QUARK up to date. The register L permits to resist to slide resynchronisation attacks, idea used in KATAN.

There are three different variants of QUARK: U-QUARK, D-QUARK, and S-QUARK. We will detail here the less light variant S-QUARK and we refer to the original paper for the other versions.

2.1.2.1 S-QUARK

is the less light flavor of QUARK. It was designed to provide *112-bit security*, and to admit a parallelization degree of 16. It is a sponge with parameters $r = 32, c = 224, b = 2 \times 128, n = 256$.

Function f. Given register X, f returns

$$\begin{aligned}&X_0 + X_{16} + X_{26} + X_{39} + X_{52} + X_{61} + X_{69} + X_{84} + X_{94} + X_{97} + X_{103}\\&+X_{103}X_{111} + X_{61}X_{69} + X_{16}X_{28} + X_{84}X_{97}X_{103} + X_{39}X_{52}X_{61}\\&+X_{16}X_{52}X_{84}X_{111} + X_{61}X_{69}X_{97}X_{103} + X_{28}X_{39}X_{103}X_{111}\\&+X_{69}X_{84}X_{97}X_{103}X_{111} + X_{16}X_{28}X_{39}X_{52}X_{61} + X_{39}X_{52}X_{61}X_{69}X_{84}X_{97}\ .\end{aligned}$$

Function g. Given register Y, g returns

$$\begin{aligned}&Y_0 + Y_{13} + Y_{30} + Y_{37} + Y_{56} + Y_{65} + Y_{69} + Y_{79} + Y_{92} + Y_{96} + Y_{101}\\&+Y_{101}Y_{109} + Y_{65}Y_{69} + Y_{13}Y_{28} + Y_{79}Y_{96}Y_{101} + Y_{37}Y_{56}Y_{65}\\&+Y_{13}Y_{56}Y_{79}Y_{109} + Y_{65}Y_{69}Y_{96}Y_{101} + Y_{28}Y_{37}Y_{101}Y_{109}\\&+Y_{69}Y_{79}Y_{96}Y_{101}Y_{109} + Y_{13}Y_{28}Y_{37}Y_{56}Y_{65} + Y_{37}Y_{56}Y_{65}Y_{69}Y_{79}Y_{96}\ .\end{aligned}$$

Function h. Given 128-bit registers X and Y, and a 10-bit register L, h returns

$$\begin{aligned}&L_0 + X_1 + Y_3 + X_7 + Y_{18} + Y_{34} + X_{47} + X_{58} + Y_{71} + Y_{80} + X_{90} + Y_{91} +\\&X_{105} + Y_{111} + Y_8X_{100} + X_{72}X_{100} + X_{100}Y_{111} + Y_8X_{47}X_{72} + Y_8X_{72}X_{100} +\\&Y_8X_{72}Y_{111} + L_0X_{47}X_{72}Y_{111} + L_0X_{47}\ .\end{aligned}$$

Function p. It is used by the data-independent LFSR, and is the same for all three instances: given a 10-bit register L, p returns $L_0 + L_3$.

Security offered The security offered by s-QUARK, as for the other variants, is completely determined by the bounds corresponding to the generic attacks against the sponge construction. For the given parameters, s-QUARK claims a collision resistance of 112 bits, as the best generic attack, for the sponge construction, which differs from finding generic collisions, has a cost of $\min\{2^{n/2}, 2^{c/2}\} = 2^{112}$. The best generic second-preimage attack costs $\min\{2^n, 2^{c/2}\} = 2^{112}$, so the second-preimage resistance is the same as the collision resistance. With respect to preimages, the best generic attack on the sponge construction with these parameters needs $\min\{2^{\min(b,n)}, \max\{2^{\min(b,n)-r}, 2^{c/2}\}\} = 2^{224}$, and therefore, the preimage resistance is 224 bits.

2.1.3 On the redefinition of security

Note that, though the digest is of size 256, implying that generic attacks for finding collision, second preimage and preimage generic attacks have complexities 2^{128}, 2^{256} and 2^{256} respectively, the generic attacks intrinsic to the sponge construction are more efficient, and consequently redefine the notions of security that we provided in the first chapter. Several other lightweight hash functions proposed after QUARK [GPP11, BKL+11] have followed the same design principle, not adapting the parameters for the ideal generic attacks, but accepting this redefinition of the security.

2.2 Kreyvium

In typical applications of homomorphic encryption, the first step consists for Alice to encrypt some plaintext m under Bob's public key pk and to send the ciphertext $c = \mathsf{HE}_{\mathsf{pk}}(m)$ to some third-party evaluator Charlie. In order to efficiently send c from Alice to Charlie, we can use a symmetric encryption scheme E, as considered in [NLV11]. Alice picks a random key k and sends a much smaller ciphertext $c' = (\mathsf{HE}_{\mathsf{pk}}(k), \mathsf{E}_k(m))$ that Charlie decompresses homomorphically into the original c using a decryption circuit $\mathcal{C}_{\mathsf{E}^{-1}}$.

In [CCF+16] we have chosen for our construction E an additive IV-based stream cipher. We investigated the performance offered in this context by Trivium, which belongs to the eSTREAM portfolio, and we also proposed a variant with 128-bit security: Kreyvium. We have been able to show that Trivium, whose security has been firmly established for over a decade, and the new variant Kreyvium has a very good performance. In this manuscript we will describe our original proposal Kreyvium.

2.2.1 Description of Kreyvium

Our first aim was to offer a variant of Trivium with 128-bit key and IV, without increasing the multiplicative depth of the corresponding circuit. Besides a higher security level, another advantage of this variant is that the number of possible IVs, and thus the maximal length of data which can be encrypted under the same key, increases from $2^{80}N_{\mathsf{trivium}}$ to $2^{128}N_{\mathsf{kreyvium}}$.[1] Increasing the key and IV-size in Trivium is a challenging task, mentioned as an open problem in [Sma14, p. 30] for instance. In particular, Maximov and Biryukov [MB07] pointed out that increasing the key-size in Trivium without any additional modification cannot be secure due to some attack with complexity less than 2^{128}. A first attempt in this direction has been made in [MB07] but the resulting cipher accommodates 80-bit IV only, and its multiplicative complexity is higher than in Trivium since the number of AND gates is multiplied by 2.

Description. Our proposal, Kreyvium, accommodates a key and an IV of 128 bits each. The only difference with the original Trivium is that we have added to the 288-bit internal state a 256-bit part corresponding to the secret key and the IV. This part of the state aims at making both the filtering and state update functions key- and IV-dependent. More precisely, these two functions f and Φ depend on the key bits and IV bits, through the successive outputs of two shift-registers K^* and IV^* initialized by the key and by the IV respectively. The internal state is then composed of five registers of sizes 93, 84, 111, 128 and 128 bits, corresponding to an internal state size of 544 bits in total, among which 416 become unknown to the attacker after initialization.

We will use the same notation as in the description of Trivium, and for the additional registers we use the usual shift-register notation: the leftmost bit is denoted by K^*_{127} (or IV^*_{127}), and the rightmost bit (*i.e.*, the output) is denoted by K^*_0 (or IV^*_0). At each clock, each one of these two registers is rotated independently from the rest of the cipher. The generator is described below, and depicted on Fig. 2.3.

[1] N represents the maximal number of keystream bits generated for a certain multiplicative depth

$(s_1, s_2, \ldots, s_{93}) \leftarrow (K_0, \ldots, K_{92})$
$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (IV_0, \ldots, IV_{83})$
$(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (IV_{84}, \ldots, IV_{127}, 1, \ldots, 1, 0)$
$(K^*_{127}, K^*_{126}, \ldots, K^*_0) \leftarrow (K_0, \ldots, K_{127})$
$(IV^*_{127}, IV^*_{126}, \ldots, IV^*_0) \leftarrow (IV_0, \ldots, IV_{127})$
for $i = 1$ to $1152 + N$ **do**
  $t_1 \leftarrow s_{66} + s_{93}$
  $t_2 \leftarrow s_{162} + s_{177}$
  $t_3 \leftarrow s_{243} + s_{288} + \boldsymbol{K^*_0}$
    if $i > 1152$ **do**
      output $z_{i-1152} \leftarrow t_1 + t_2 + t_3$
    end if
  $t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171} + \boldsymbol{IV^*_0}$
  $t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{264}$
  $t_3 \leftarrow t_3 + s_{286} \cdot s_{287} + s_{69}$
  $t_4 \leftarrow K^*_0$
  $t_5 \leftarrow IV^*_0$
  $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
  $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
  $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$
  $(K^*_{127}, K^*_{126}, \ldots, K^*_0) \leftarrow (t_4, K^*_{127}, \ldots, K^*_1)$
  $(IV^*_{127}, IV^*_{126}, \ldots, IV^*_0) \leftarrow (t_5, IV^*_{127}, \ldots, IV^*_1)$
end for

Related ciphers. KATAN [CDK09b] is a lightweight block cipher with a lot in common with Trivium. It is composed of two registers, whose feedback functions are very sparse, and have a single nonlinear term. The key, instead of being used for initializing the state, is introduced by xoring two key information-bits per round to each feedback bit. The recently proposed stream cipher Sprout [AM15], inspired by Grain but with much smaller registers, also inserts the key in a similar way: instead of using the key for initializing the state, one bit of key information is xored at each clock to the feedback function. The attacks that applied to Sprout, as for instance [LN15a], do not apply to Kreyvium in part because of its big state. We can see the parallelism between these two ciphers and our newly proposed variant. In particular, the previous security analysis on KATAN shows that this type of design does not introduce any clear weakness. Indeed, the best attacks on round-reduced versions of KATAN so far [FM14] are meet-in-the-middle attacks, that exploit the knowledge of the values of the first and the last internal states (due to the block-cipher setting). As this is not the case here, such attacks,

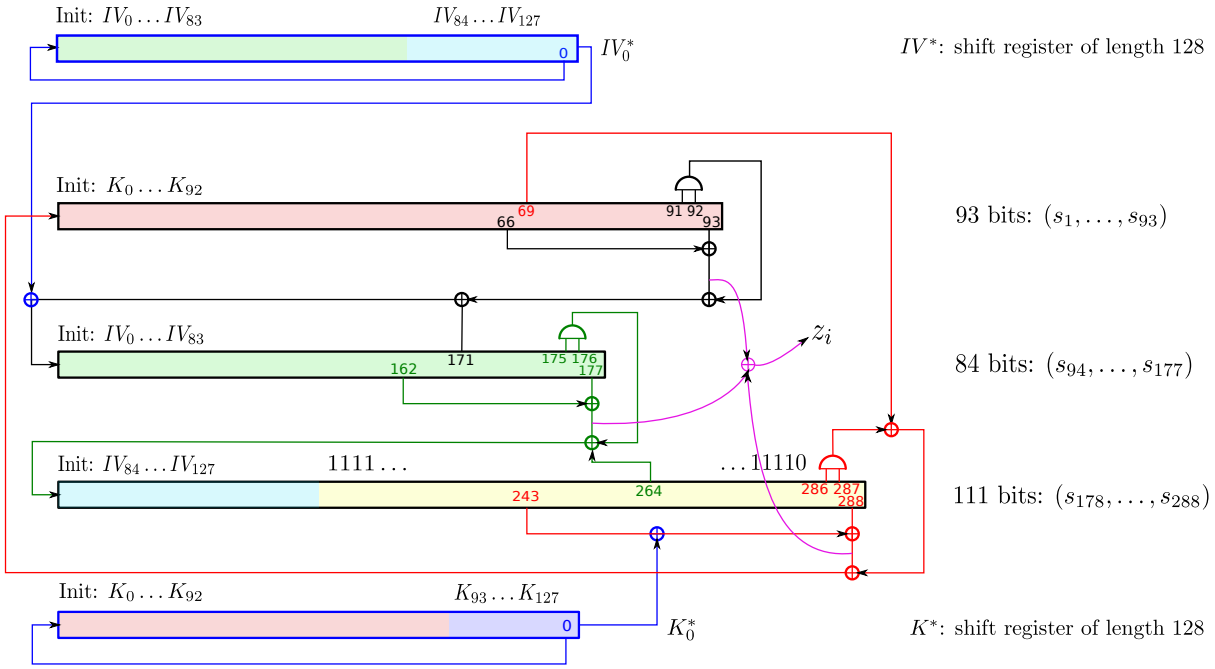


Figure 2.3: Kreyvium. The three registers in the middle correspond to the original Trivium. The modifications defining Kreyvium correspond to the two registers in blue.

as well as the recent interpolation attacks against LowMC [DLMW15], do not apply. The best attacks against KATAN, when excluding MitM techniques, are conditional differential attacks [KMNP10, KMN11].

Design rationale. In Kreyvium, we have decided to xor the key bit K_0^* to the feedback function of the register that interacts with the content of $(s_1, \dots, s_{63})$ the later, since $(s_1, \dots, s_{63})$ is initialized with some key bits. The same goes for the IV^* register. Moreover, as the key bits that start entering the state are the ones that were not in the initial state, all the key bits affect the state the soonest possible.

We also decided to initialize the state with some key bits and with all the IV bits, and not with a constant value, as this way the mixing will be performed quicker. Then we can expect that the internal-state bits after initialization are expressed as more complex and less sparse functions in the key and IV bits.

Our change of constant is motivated by the conditional differential attacks from [KMN11]: the conditions needed for a successful attack are that 106 bits from the IV or the key are equal to '0' and a single one needs to be '1'. This suggests that values set to zero "encourage" non-random behaviors, leading to our new constant. In other words, in Trivium, an all-zero internal state is always updated to an all-zero state, while an all-one state will change through time. The 0 at the end of the constant is added for preventing slide attacks.

2.2.2 Comparison with other proposals

Being a recently flourishing area, few other ciphers have been proposed, like LowMC [ARS+15] or FLIP [MJSC16]. Though their current versions remain unbroken, previous versions of both ciphers have been attacked, and the ciphers consequently tweaked. Our proposal is the only one for which no attack has been found so far.

An interesting follow-up study that we have embarked is to propose an extended version of Kreyvium with a 256-bit key, in order to be useful for other scenarios, for instance post-quantum cryptography. Is it possible to just increase the key size?

2.3 Zorro: an experiment on reducing AES multiplications

A first issue we addressed in [GGNPS13] was how to choose an S-box with fewer multiplications than the one in the AES by trading cryptanalytic properties for more efficient masking. A complementary approach in order to design a block cipher that is easy to mask is to additionally reduce the total number of S-box evaluations. For this purpose, a natural solution is to consider rounds where not all the state goes through the S-boxes. To some extent, this proposal can be viewed as similar to an NLFSR-based cipher (*e.g.* Grain [HJM07], Katan [CDK09b], Trivium [CP08]), where the application of a non-linear component to the state is not homogeneous. For example, say we consider two n-bit block ciphers with s-bit S-boxes: the first (parallel) one applies n/s S-boxes in parallel in each of its R rounds, while the second (serial) one applies only a single S-box per round, at the cost of a larger number R' of rounds. If we can reach a situation such that $R' < R \cdot \frac{n}{s}$, then the second cipher will indeed require fewer S-box evaluations in total, hence being easier to protect against side-channel attacks. Different trade-offs are possible. In general, the relevance of such a proposal highly depends on the diffusion layer. For example, we have been able to conclude that wire-crossing permutations (like the one of PRESENT [BKL+07a]) cannot lead to any improvement of this type. By contrast, an AES-like structure is better suited to our goal. The rationale behind this intuition relates to the fact that the AES Rijndael has strong security margins against statistical attacks, and the most serious concerns motivating its number of rounds are structural (e.g. [KW02]). Hence, iterating simplified rounds seems a natural way to prevent such structural attacks while maintaining security against linear/differential cryptanalysis. Taking all this into account, we proposed Zorro, a risky primitive aimed at being a proof of concept, that is described next.

2.3.1 Preliminary investigations: how many S-boxes per round?

As for finding S-boxes that are easier to mask, an exhaustive analysis of all the round structures that could give rise to fewer S-box executions in total is out of reach. Yet, and as this number of S-box executions mainly depends on the `SB` operations, we considered several variants of it, while keeping `SR`, `MC` and `AK` unchanged. For this purpose, we have first analyzed how some elementary diffusion properties depend on the number and positions of the S-boxes within the state. Namely, we considered (1) the number of rounds so that all the input bytes have passed at least once through an S-box (`NrSbox`); (2) the number of rounds so that all the output bytes have at least one non-linear term (`NrNlin`); and (3) the maximal number of rounds so that an input difference has a non-linear effect in all the output bytes (`NrDiff`). In all three cases,

these number of rounds should ideally be low. While such an analysis is of course heuristic, it indicates that considering four S-boxes per round, located in a single row of the state matrix seemed an appealing solution. Our goal was to show that an AES-like block cipher where each round only applies four "easy-to-mask" S-boxes can be secure. In particular, we selected the number of rounds as $R' = 21$, so that we have (roughly) twice fewer S-boxes executed than in the original AES Rijndael (i.e. 21×4 vs. 10×16).

2.3.2 The block cipher `Zorro`: specifications

We use a block size and key size of $n = 128$ bits, iterate 24 rounds and call the combination of 4 rounds a step. Each round is a composition of four transforms: `SB`*, `AC`, `SR`, and `MC`, where the last two ones are exactly the same operations as in the AES Rijndael, `SB`* is a variant of `SB` where only 4 S-boxes are applied to the 4 bytes of the first row in the state matrix, and `AC` is a round-constant addition: The round-constant addition is limited to the first state row. Constants can be generated "on-the-fly" according to $\{i, i, i, i \ll 3\}$, where i is the round index and $\ll$ the left shift operator.

We additionally perform a key addition `AK` before the first step and after each step. We selected an S-box with twice fewer multiplications than the one in the AES.

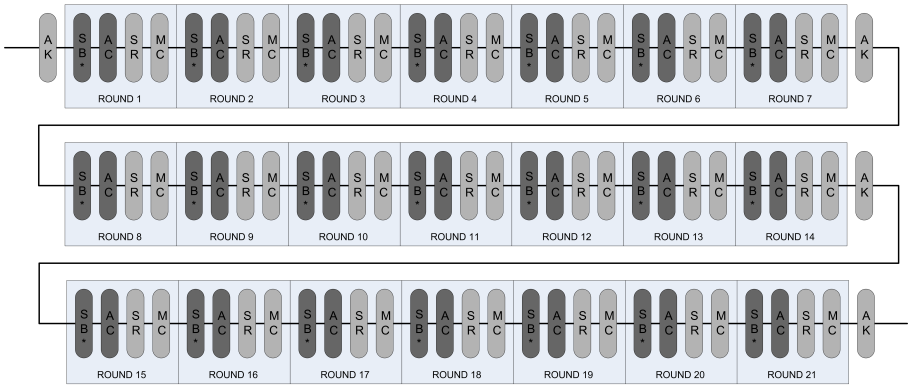


Figure 2.4: The block cipher `Zorro`: light gray operations are AES-like, dark gray ones are new.

2.3.3 Cryptanalysis of `Zorro` and conclusions

Due to the problems of applying conventional tools for bounding the probabilities of the best differential and linear paths, we performed an analysis taking into account freedom degrees: each time that we want to control a transition, it costs us the corresponding freedom degrees. Taking into account the limit of the available degrees and the minimal possible probability of the path in order to still be able to build an attack, we counted 14 as the maximal number of rounds that could be cryptanalyzed with differential or linear cryptanalysis.

Wang et al. [WWGY14] published an attack that showed us wrong. This is due to the fact that the order of MC is 4. This allows them to build iterative characteristics through 4 rounds, where after the first round, no more degrees of freedom need to be consumed as the linear

relations stay satisfied. These attacks were improved in [BDD+15]. The authors reduced the complexity down to practical, and performed an extensive study of other variants to find out if some could be secure. They could show that `Zorro`'s weakness was produced by an unlucky choice of parameters and not by an intrinsic weakness of the construction. Other variants proposed in their paper remain still unbroken.

Chapter 3

Algorithmic results on list merging

Contents

While considering the generalization and improvement of families of cryptanalytic attacks, I realized that a recurrent problem that appeared in symmetric cryptanalysis and that was often not solved in an optimal way was merging lists with respect to certain relations: given N lists of elements, we only want to keep the N-tuples that verify a certain relation $\mathcal{R}$. It was the case for instance for several rebound attacks, which was the most used attack on the candidates of the SHA-3 competition. In [NP11] I proposed several generic algorithms that improve the complexities of many dedicated rebound attacks. I gave an extension of these algorithms in [ABNP+11b], as well as several new applications in [NPTV11, JNPS11, JNPP12a] in the context of rebound attacks. In [NPRM11, LN15b, LN15a] the algorithms were applied in other cryptanalysis scenarios, improving the complexities of the attacks. In [CNPV13], where we proposed an improved type of meet-in-the-middle attacks, new applications where shown. We describe in this chapter the algorithms that apply when $\mathcal{R}$ is group-wise[1], as they are the ones that can be applied in a large number of scenarios to improve the attacks complexities.

3.1 General problem

In many cryptanalyses we need to solve a step that involves enumerating, from a very large set of possible candidates represented as a cross product of lists, all those that verify a given relation $\mathcal{R}$. We call this operation *"merging" the lists*.

In [NP11] I proposed two scenarios for the problem, and their corresponding solutions: When $\mathcal{R}$ is group-wise and for the so-called stop-in-the-middle problems. We will explain here the first one and the proposed solutions, as this problem finds more applications in other cryptanalysis settings.

[1] Group-wise refers here to a grouping of bits, *i.e.* $\mathcal{R}$ can be decomposed in smaller relations $\mathcal{R}_i$ that have to be verified in parallel.

3.1.1 Merging n lists with respect to a relation $\mathcal{R}$

The general problem is represented in Fig. 3.1. Let $\mathcal{R}$ be a Boolean function taking N k-bit words as input, *i.e.* $\mathcal{R} : (\{0,1\}^k)^N \to \{0,1\}$. Let $L_1, \ldots, L_N$ be N given lists of k-bit words drawn uniformly and independently at random from $\{0,1\}^k$. We assume that the probability over all N-tuples X in $L_1 \times \ldots \times L_N$ that $\mathcal{R}(X) = 1$ is π. For any given function $\mathcal{R}$ and any given N-tuple of lists $(L_1, \ldots, L_N)$ the *merging problem* consists in finding the list $\mathcal{L}_{sol}$ of all $X \in L_1 \times \ldots \times L_N$ satisfying $\mathcal{R}(X) = 1$. We call this operation *merging* the lists $L_1, \ldots, L_N$ to obtain $\mathcal{L}_{sol}$.

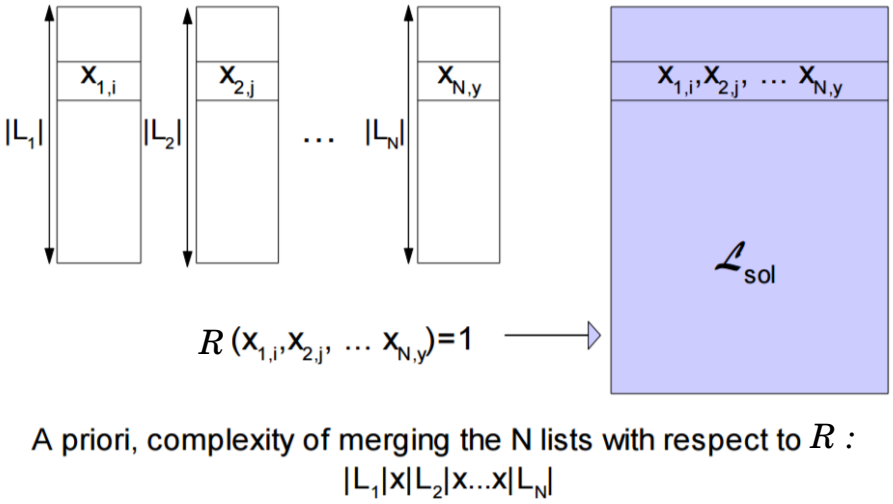


Figure 3.1: General merging problem

It is assumed that the image of a given input under $\mathcal{R}$ can be easily computed. In the following, the size of a list L is denoted by $|L|$. A brute force method for solving this problem therefore consists in enumerating all the $|L_1| \times \ldots \times |L_N|$ inputs, in computing $\mathcal{R}$ on all of them and in keeping the ones verifying $\mathcal{R} = 1$. Note that, in absence of any additional information on $\mathcal{R}$, it is theoretically impossible to do better. However, in practice, the function $\mathcal{R}$ often has a set of properties which can be exploited to optimize this approach. We aim at reducing the number of candidates that have to be examined, in some cases by a preliminary sieving similar to the one used in [NP09]. We will now detail the case when $\mathcal{R}$ is group-wise.

3.1.2 When $\mathcal{R}$ is group-wise

In some cases we can considerably reduce the complexity of the *merging* problem by redefining it into a more concrete one. We consider here a very common case that will appear in many cryptanalysis scenarios, as we will later show with the examples. This case corresponds to a function $\mathcal{R}$ that can be decomposed into smaller functions. Note that in most concrete examples that we studied, the number N of lists was either 2, 4 or 6, but we preferred to state the problem in full generality, for any possible N.

Problem 1: Let $L_1, \ldots, L_N$ be N lists of size $2^{l_1}, \ldots, 2^{l_N}$ respectively, where the elements are drawn uniformly and independently at random from $\{0,1\}^k$.
Let $\mathcal{R}$ be a Boolean function, $\mathcal{R} : \left(\{0,1\}^k\right)^N \to \{0,1\}$ and let t be an integer such that there exists $N' < N$ and some triples of functions $\mathcal{R}_j : \{0,1\}^{2s} \to \{0,1\}$, $f_j : (\{0,1\}^k)^{N'} \to \{0,1\}^s$ and $f'_j : (\{0,1\}^k)^{(N-N')} \to \{0,1\}^s$ for $j = 1, \ldots, t$ such that, $\forall\, (\boldsymbol{x_1}, \ldots, \boldsymbol{x_N}) \in L_1 \times \ldots \times L_N$:

$$\mathcal{R}(\boldsymbol{x_1}, \ldots, \boldsymbol{x_N}) = 1 \quad \Leftrightarrow \quad \begin{array}{l} \forall j = 1, \ldots, t, \\ \left\{ \begin{array}{ll} & \mathcal{R}_j(v_j, v'_j) = 1 \\ \text{with} & v_j = f_j(\boldsymbol{x_1}, \ldots, \boldsymbol{x_{N'}}) \\ \text{and} & v'_j = f'_j(\boldsymbol{x_{N'+1}}, \ldots, \boldsymbol{x_N}) \end{array} \right. \end{array}$$

Let π be the probability that $\mathcal{R} = 1$ for a random input.
Problem 1 consists in *merging* these N lists to obtain the set $\mathcal{L}_{sol}$, of size $\pi 2^{\sum_{i=1}^{N} \ell_i}$, of all N-tuples of $(L_1 \times \ldots \times L_N)$ verifying $\mathcal{R} = 1$.

Reduction from N to 2: For any $N \geq 2$ *Problem 1* can be reduced to an equivalent problem with $N = 2$, *i.e.* merging two lists $\mathcal{L}_A$ and $\mathcal{L}_B$, which consist of elements of $(\{0,1\}^s)^t$ corresponding to $\boldsymbol{x_A} = \boldsymbol{v} = (v_1, \ldots, v_t)$ and $\boldsymbol{x_B} = \boldsymbol{v'} = (v'_1, \ldots, v'_t)$, with respect to the function $(\boldsymbol{x_A}, \boldsymbol{x_B}) \to \Pi_{j=1}^{t} \mathcal{R}_j(v_j, v'_j)$. The reduction is performed as follows:

1. Build a table T_A^* of size $2^{\sum_{i=1}^{N'} \ell_i}$ storing each element $\boldsymbol{e_A} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_{N'}})$ of $L_1 \times \ldots \times L_{N'}$, indexed[2] by the value of $(f_1(\boldsymbol{e_A}), \ldots, f_t(\boldsymbol{e_A}))$, *i.e.* $(v_1, \ldots, v_t)$. Store the corresponding $(v_1, \ldots, v_t)$ in a list $\mathcal{L}_A$. Note that several $\boldsymbol{e_A}$ may lead to the same value of $(v_1, \ldots, v_t)$.

2. Build a similar table T_B^* of size $2^{\sum_{i=N'+1}^{N} \ell_i}$ storing each element $\boldsymbol{e_B} = (\boldsymbol{x_{N'+1}}, \ldots, \boldsymbol{x_N})$ of $L_{N'+1} \times \ldots \times L_N$, indexed by $(f_1(\boldsymbol{e_B}), \ldots, f_t(\boldsymbol{e_B}))$, *i.e.* $(v'_1, \ldots, v'_t)$. Store $(v'_1, \ldots, v'_t)$ in a list $\mathcal{L}_B$.

3. *Merge* $\mathcal{L}_A$ and $\mathcal{L}_B$ with respect to $\Pi_{j=1}^{t} \mathcal{R}_j$ and obtain $\mathcal{L}_{sol}$.

4. Build $\mathcal{L}_{sol}^*$ by iterating over each pair $((v_1, \ldots, v_t), (v'_1, \ldots, v'_t))$ of $\mathcal{L}_{sol}$, and adding the set of all $(x_1, \ldots, x_{N'}, x_{N+1}, \ldots, x_N) \in T_A^* [(v_1, \ldots, v_t)] \times T_B^* [(v'_1, \ldots, v'_t)]$. $\mathcal{L}_{sol}^*$ is the solution to the original problem, represented in Fig. 3.2.

Let $2^{T_{\text{merge}}}, 2^{M_{\text{merge}}}$ be the time and memory complexities of step 3. The total time complexity of solving *Problem 1* is $\mathcal{O}(st2^{\sum_{i=1}^{N'} \ell_i} + st2^{\sum_{i=N'+1}^{N} \ell_i} + 2^{T_{\text{merge}}} + \pi 2^{\sum_{i=1}^{N} \ell_i})$ where the last term comes from the fact that only the N-tuples satisfying $\mathcal{R} = 1$ are examined at step 4 because of the sieve applied at step 3. The proportion of such tuples is then π. The memory complexity is $\mathcal{O}((ts + N'k)2^{\sum_{i=1}^{N'} \ell_i} + (ts + (N - N')k)2^{\sum_{i=N'+1}^{N} \ell_i} + 2^{M_{\text{merge}}} + \pi 2^{\sum_{i=1}^{N} \ell_i})$ (where the last term appears only when the solutions must be stored).

Using the brute force approach, $2^{T_{\text{merge}}}$ would be $2^{\ell_A + \ell_B}$ where 2^{ℓ_A} (respectively 2^{ℓ_B}) denotes the size of $\mathcal{L}_A$ ($\mathcal{L}_B$), and $2^{M_{\text{merge}}}$ would be negligible. We present in the following sections some algorithms for solving *Problem 1* considering $N = 2$ with $\mathcal{L}_A$ and $\mathcal{L}_B$, that provide better complexities than the brute force approach. Those algorithms can be applied for obtaining a smaller $2^{T_{\text{merge}}}$ when $N > 2$.

[2]Here and in the following sections we can use standard hash tables for storage and lookup in constant time, since the keys are integers.

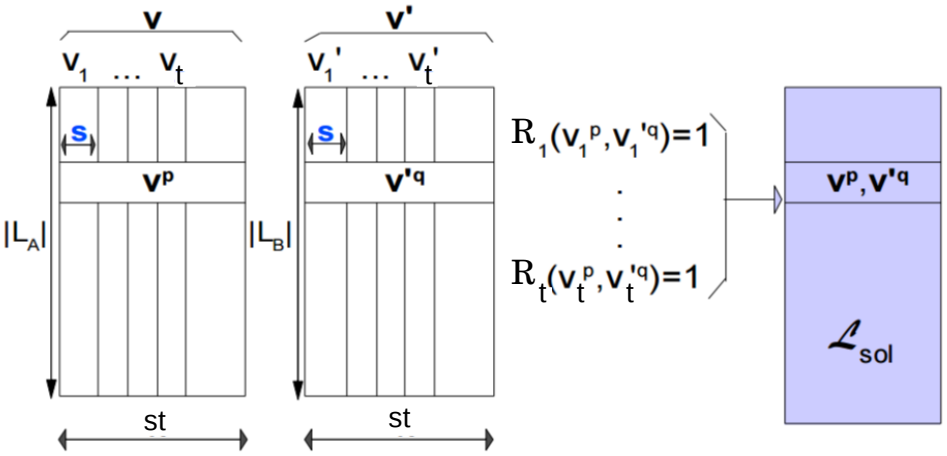


Figure 3.2: The merging problem with $\mathcal{R}$ group-wise and $N = 2$. Here, p and q represent a particular pair of positions od v and v' so that the correspondign pair of elements verifies $\mathcal{R}$.

3.1.3 A first algorithm: instant/gradual matching

For the algorithms to work we do not need the t groups to be of the same size. We can generalize the problem as follows: we consider two lists, $\mathcal{L}_A$ of size 2^{ℓ_A} and $\mathcal{L}_B$ of size 2^{ℓ_B}, whose roles are interchangeable. The elements of both lists can be decomposed into t groups: the i-th group of $a \in \mathcal{L}_A$ has size m_i, while the i-th group of $b \in \mathcal{L}_B$ has size p_i. The Boolean relation $\mathcal{R}$ can similarly be considered group-wise: $\mathcal{R}(a, b) = 1$ if and only $\mathcal{R}_i(a_i, b_i) = 1$ for all $1 \leq i \leq t$. The sieving probability π associated to $\mathcal{R}$ then corresponds to the product of the sieving probabilities π_i associated to each $\mathcal{R}_i$. We can consider for instance that each $\mathcal{R}_i$ corresponds to an S-box S_i with n_i-bit inputs, a table storing all (a_i, b_i) such that $\mathcal{R}_i(a_i, b_i) = 1$ can be built with time complexity 2^{n_i}, by computing all $(x_i, S_i(x_i)), x_i \in \mathbf{F}_2^{n_i}$. The corresponding memory complexity is proportional to $\pi_i 2^{m_i+p_i}$. These tables are only built once for all and, in some situations, these tables can be built "on-the-fly" with a few operations.

We now provide a complete description of three matching algorithms. For the sake of simplicity, we assume in the description of the algorithms that the lists are sorted, but in practice we can use standard hash tables for storage and lookup in constant time. It is worth noticing that the size of the list $\mathcal{L}_{sol}$ returned by the matching algorithms is a priori not included in the memory complexity since most of the times each of its elements can be used and tested as soon as it has been found.

3.1.3.1 Instant matching

Instant matching successively considers all elements in $\mathcal{L}_B$: for each $b \in \mathcal{L}_B$, a list $\mathcal{L}_{aux}$ of all a such that $\mathcal{R}(a, b) = 1$ is built, and each element of $\mathcal{L}_{aux}$ is searched within $\mathcal{L}_A$.

$$\text{Time} = \pi 2^{\ell_B + \sum_{i=1}^{t} m_i} + \pi 2^{\ell_A + \ell_B} \text{ and Memory} = 2^{\ell_A} + 2^{\ell_B} \ .$$

Algorithm 1 Instant matching algorithm of $\mathcal{L}_A$ and $\mathcal{L}_B$ with respect to $\mathcal{R}$.

for j from 1 to t **do**
  Build the table T_j such that $T_j[v_j]$ corresponds to all u_j with $\mathcal{R}_j(u_j, v_j) = 1$.
for each $(b_1, \ldots, b_t) \in \mathcal{L}_B$ **do**
  $\mathcal{L}_{aux} \leftarrow \emptyset$.
  for j from 1 to t **do**
    if $T_j[b_j]$ is empty, **then** go to 3.
  Add all tuples $(x_1, \ldots, x_t)$ with $x_j \in T_j[b_j]$, $\forall j$, to $\mathcal{L}_{aux}$.
  for each $(x_1, \ldots, x_t)$ in $\mathcal{L}_{aux}$ **do**
    if $(x_1, \ldots, x_t) \in \mathcal{L}_A$ **then**
      Add $(x_1, \ldots, x_t, b_1, \ldots, b_t)$ to $\mathcal{L}_{sol}$.
Return $\mathcal{L}_{sol}$.

3.1.3.2 Gradual matching

Gradual matching is a recursive procedure as detailed by Algo 2. All elements are decomposed into two parts, the first t' groups and the last $(t - t')$, with $t' < t$. For each possible value β of the first t' groups, the sublist $L_B(\beta)$ is built. It consists of all elements in $\mathcal{L}_B$ whose first t' groups take the value β. Now, for each α such that $\mathcal{R}_i(\alpha_i, \beta_i) = 1, 1 \leq i \leq t'$, $L_B(\beta)$ is merged with the sublist $L_A(\alpha)$ which consists of all elements in $\mathcal{L}_A$ whose first t' groups take the value α. Then, we need to merge two smaller lists, of respective sizes $2^{\ell_A - \sum_{i=1}^{t'} m_i}$ and $2^{\ell_B - \sum_{i=1}^{t'} p_i}$.

$$\text{Time} = \left(\prod_{i=1}^{t'} \pi_i \right) 2^{\sum_{i=1}^{t'} m_i + p_i} C_{\mathsf{merge}} \text{ and Memory} = 2^{\ell_A} + 2^{\ell_B} \ .$$

where C_{merge} is the cost of merging the two remaining sublists.

Algorithm 2 Gradual matching algorithm of $\mathcal{L}_A$ and $\mathcal{L}_B$ with respect to $\mathcal{R}$.

for j from 1 to t **do**
  Build the table T_j such that $T_j[v_j]$ corresponds to all u_j with $\mathcal{R}_j(u_j, v_j) = 1$.
for each $\beta = (\beta_1, \ldots, \beta_{t'})$ in $(\mathbf{F}_2^{\sum_{j=1}^{t'} p_j})$ **do**
  $L_B(\beta) \leftarrow \{b \in \mathcal{L}_B \text{ with } (b_1, \ldots, b_{t'}) = \beta\}$
  $\mathcal{L}_{aux} \leftarrow \emptyset$.
  for each $(\alpha_1, \ldots, \alpha_{t'})$ with $\alpha_j \in T_j[\beta_j]$, $\forall j \leq t'$ **do**
    add $(\alpha_1, \ldots, \alpha_{t'})$ to $\mathcal{L}_{aux}$.
  for each $\alpha = (\alpha_1, \ldots, \alpha_{t'})$ in $\mathcal{L}_{aux}$ **do**
    $L_A(\alpha) \leftarrow \{a \in \mathcal{L}_A \text{ with } (a_1, \ldots, a_{t'}) = \alpha\}$
    Merge $L_A(\alpha)$ with $L_B(\beta)$ with respect to $\mathcal{R}' = \prod_{j=t'+1}^{t} \mathcal{R}_j$.
    Add the solutions to $\mathcal{L}_{sol}$.
Return $\mathcal{L}_{sol}$.

3.1.4 Parallel matching and dissection problems

In [DDKS12] Dinur, Dunkelman, Keller and Shamir introduced a new type of algorithm called dissection, that can be applied to a large class of diverse problems under the condition of

having a bicomposite structure. We provided in [CNPV13] the first general description of the memoryless version of parallel matching. The details are provided by Algo 3. This algorithm applies [DDKS12] to the parallel matching algorithm from [NP11]: instead of building a big auxiliary list as in the original parallel matching, we here build small ones which do not need to increase the memory needs. In parallel matching, the elements in both lists are decomposed into three parts: the first t_1 groups, the next t_2 groups, and the remaining $(t - t_1 - t_2)$ groups. Both lists $\mathcal{L}_A$ and $\mathcal{L}_B$ are sorted in lexicographic order. Then, $\mathcal{L}_A$ can be seen as a collection of sublists $\mathcal{L}_A(\alpha)$, where $\mathcal{L}_A(\alpha)$ is composed of all elements in $\mathcal{L}_A$ whose first t groups equal α. Similarly, $\mathcal{L}_B$ is seen as a collection of $\mathcal{L}_B(\beta)$. The matching algorithm then proceeds as follows. For each possible value α for the first t groups, an auxiliary list $\mathcal{L}_{aux}$ is built, corresponding to the union of all $\mathcal{L}_B(\beta)$ where (α, β) satisfies the first t relations $\mathcal{R}_j$. The list $\mathcal{L}_{aux}$ is sorted by its next t_2 groups. Then, for each element in $\mathcal{L}_A(\alpha)$, we check if a match for its next t_2 groups exists in $\mathcal{L}_{aux}$. For each finding, the remaining $(t - t_1 - t_2)$ groups are tested and only the elements which satisfy the remaining $(t - t_1 - t_2)$ relations are returned.

Algorithm 3 Memoryless parallel matching algorithm of $\mathcal{L}_A$ and $\mathcal{L}_B$ with respect to $\mathcal{R}$.

for j from 1 to t' **do**
  Build the table T_j such that $T_j[v_j]$ corresponds to all u_j with $\mathcal{R}_j(u_j, v_j) = 1$.
for each $\alpha = (a_1, \ldots, a_{t_1})$ appearing in $\mathcal{L}_A$ **do**
  $\mathcal{L}_A(\alpha) \leftarrow \{a \in \mathcal{L}_A : (a_1, \ldots, a_{t_1}) = \alpha\}$.
  // Compute $\mathcal{L}_{aux}$
  $\mathcal{L}_1 \leftarrow \{\beta : \mathcal{R}_j(\alpha_j, \beta_j) = 1, 1 \le j \le t_1\}$
  $\mathcal{L}_{aux} \leftarrow \emptyset$
  for each $\beta \in \mathcal{L}_1$ **do**
    $\mathcal{L}_B(\beta) \leftarrow \{b \in \mathcal{L}_B : (b_1, \ldots, b_{t_1}) = \beta\}$.
    add all elements of $\mathcal{L}_B(\beta)$ to $\mathcal{L}_{aux}$.
  Sort $\mathcal{L}_{aux}$ by $\beta' = (b_{1+t_1}, \ldots, b_{t_1+t_2})$.
  // Merge $\mathcal{L}_A(\alpha)$ and $\mathcal{L}_{aux}$ with respect to the next t_2 groups.
  for each a in $L_A(\alpha)$ **do**
    $\mathcal{L}_2 \leftarrow \{\beta' : \mathcal{R}_j(\alpha_j, \beta'_j) = 1, t_1 < j \le t_1 + t_2\}$
    for each $\beta' \in \mathcal{L}_2$ **do**
      if $\beta' \in \mathcal{L}_{aux}$ **then**
        for each $b \in \mathcal{L}_{aux}$ with $(b_{t_1+1}, \ldots, b_{t_1+t_2}) = \beta'$ **do**
          if $\mathcal{R}_j(a_j, b_j)$ for all $t_2 < j \le t$ **then**
            Add (a, b) to $\mathcal{L}_{sol}$.

The time and memory complexities can be evaluated as follows. We first evaluate the average sizes of all lists involved in the algorithm. For each α, the average size of $\mathcal{L}_A(\alpha)$ is $2^{\ell_A - \sum_{i=1}^{t_1} m_i}$. Also, we have

$$|\mathcal{L}_1| = \left(\prod_{i=1}^{t_1} \pi_i\right) 2^{\sum_{i=1}^{t_1} p_i} \ , \ |\mathcal{L}_2| = \left(\prod_{i=t_1+1}^{t_1+t_2} \pi_i\right) 2^{\sum_{i=t_1+1}^{t_1+t_2} p_i}$$

and

$$|\mathcal{L}_{aux}| = \left(\prod_{i=1}^{t_1} \pi_i\right) 2^{\ell_B} \ .$$

Finally, the average number N of elements b which match with a on the first t_1+t_2 groups and that should be tested at Line 18 in the algorithm is $\left(\prod_{i=1}^{t_1+t_2}\pi_i\right)2^{\ell_B}$. Then, the average time complexity of parallel matching can be decomposed as

$$\begin{aligned}\text{Time} &= 2^{\sum_{i=1}^{t_1} m_i}\left[|\mathcal{L}_{aux}| + |\mathcal{L}_A(\alpha)|\left(|\mathcal{L}_2| + N\right)\right] \\ &= \left(\prod_{i=1}^{t_1}\pi_i\right)2^{\ell_B+\sum_{i=1}^{t_1} m_i} + \left(\prod_{i=t_1+1}^{t_1+t_2}\pi_i\right)2^{\ell_A+\sum_{i=t_1+1}^{t_1+t_2} p_i} + \left(\prod_{i=1}^{t_1+t_2}\pi_i\right)2^{\ell_A+\ell_B}\ .\end{aligned}$$

It is worth noticing that the two lists $\mathcal{L}_1$ and $\mathcal{L}_2$ do not need to be stored since their elements are entirely defined by the tables T_j describing the valid transitions for $\mathcal{R}_j$. The average memory required by the algorithm then corresponds to

$$\text{Memory} = |\mathcal{L}_A| + |\mathcal{L}_B| + |\mathcal{L}_{aux}| = 2^{\ell_A} + 2^{\ell_B} + \left(\prod_{i=1}^{t_1}\pi_i\right)2^{\ell_B}\ .$$

Parallel matching for non-random elements. In [ABNP+11a] I adapted for the needs of a concrete cryptanalysis the previous algorithm for the case of lists composed of non-random ternary vectors. They can be adapted to many more scenarios following that example.

3.2 Applications and conclusion

These algorithms were introduced in the context of rebound attacks. Proposed in 2009 [MRST65], rebound attacks where the most used tool to analyze the candidates of the SHA-3 competition. After a detailed study of these attacks I realized that the complexity bottleneck for all of them was an algorithmic step that was not yet solved optimally: merging N big lists in orden to obtain all the N-tuples verifing a given relation $\mathcal{R}$. Thanks to my new algorithms, I was able to provide in [NP11] new improved rebound attacks on JH, Grøstl, ECHO, Luffa and LANE), as the ones in [NPTV11, JNPS11, JNPP12a]. Later, in a collaboration with Toz and Varici [NPTV11], we were able to propose a way of finding solutions for differential paths covering the whole number of rounds of the compression function of JH, and with the new merging algorithms we could build an attack on the whole function. Similarly, I was able to find a new rebound attack [JNPS11] on the function ECHO. In [JNPP12b] we presented the best known results on the finalist of the SHA-3 competition Grøstl: I found a way of applying my algorithms in order to extend by one round the best known paths on these type of construction (which had been an open problem for a while).

In our results on Keccak [NPRM11], Klein [LN15b] and Sprout [LN15a] the algorithms were applied in other cryptanalysis scenarios, improving the complexities of the attacks. In all these cases, we could improve the complexity of the attacks by computing partial solutions of a bigger problem, that where stored in lists and combined afterwards with the merging algorithms. This proved that they can be very useful in guess-and-determine scenarios. In [ABNP+11a], the parallel matching algorithm was applied in a guess-and-determine attack with a different setting.

Together with Canteaut and Vayssière, we proposed a generic improvement of meet-in-the-middle attacks [CNPV13] that will be described in the next chapter. The algorithms presented

here have proved to be essential for applying this improvement efficiently. I do believe that new applications will appear, as they are quite general algorithms.

I believe the general problem described here of list merging with respect to a given relation will find more new applications in cryptanalysis. Most of the time, in guess-and-determine scenarios, this kind of problem appears. Recognizing it is a fundamental task that should be more developed. The results from [DDKS12] were a big step in that direction. To sum up, considering partial solutions seems an important line of cryptanalysis improvements. This is not done systematically, and more understanding for recognizing these situations would be of great help.

Chapter 4

Generalization of families of cryptanalysis

Contents

Cryptanalysis has recently experienced a large number of new advances. In particular, new tools have appeared, like rebound attacks, cube attacks, etc. In most cases, new cryptanalysis techniques are introduced in the context of a particular algorithm and are described as ad-hoc techniques, which makes them hard to generalize. The technical complexity acquired by being applied to a particular case hides very often the main ideas and makes them difficult to master, and therefore to optimize and adapt. That is why a technical task of generalization of attacks needs to be done—or, I should say, continued. Indeed, I have been leading a recent initiative of systematic generalization of symmetric cryptanalysis techniques. For instance, cryptanalysis using impossible differentials [BNS14, BLNS16], conditional differentials [KMNP10], meet-in-the-middle [CNPV13], correlation attacks [CN12] and multiple limited birthday [JNP13] can now be done in a near-automatic way[1], and with optimized complexities; and, due to the final simplified and complete version of the attacks, new ideas for improving them have been found,

[1] even automatic in some cases like in [DF16] thanks to the previous analysis

which allows even more efficient attacks. In this manuscript we will recall the main ideas of our improvements and generalization of impossible differential attacks, meet-in-the-middle attacks, and (truncated) differential attacks.

4.1 Symmetric cryptanalysis context

Cryptanalysis has recently seen a large number of advances. In particular, new tools have appeared including rebound attacks [MRST09] and cube attacks [DS09]. But in most cases, these new tools have been introduced in the context of a particular algorithm, and have been described as *ad-hoc* techniques, which makes them hard to generalize. Merely finding the foundations of the attack itself is a very difficult and technical task, whence the difficulty of optimizing such attacks and applying them to new algorithms arises. A systematic generalization of symmetric cryptanalysis techniques is needed. This is also the main direction of the research proposal I submitted to get my current position. I have already found several results, for example on impossible differentials [BNS14], rebound attacks [NP11], meet-in-the-middle attacks [CNPV13], and correlation attacks [CN12]. These results show not only that the cryptanalysis becomes applicable in a nearly automated way, but also that the complexities are optimized. Very often, due to the final simplified and complete version of the attack, new ideas for improving them have been found, allowing even more performant attacks. This task in itself is fundamental for symmetric cryptography: Cryptanalysis is essential to build up confidence in symmetric cryptography. Another point that independently demonstrates the importance of this task is the recent profusion of competitions to seek and select standards or recommendations. These competitions, while important, have the negative side-effect of "rushing" cryptanalysts and results, thus generating some "cryptanalysis chaos". This means that most of the new techniques and ideas that have appeared (and some not-so-new ones, such as impossible differential cryptanalysis) are not fully understood by the community; and the important work of comprehension, generalization, and optimization needs to be carried out. Our papers on scrutinizing impossible differentials, such as [BNS14], are a striking example. We detected a large number of published wrong results trying to use the technique; we extracted the main ideas; and we generalized them, and provided improvements, with the help of the newly-acquired simplified and clear version. This allowed us to provide the best reduced-round attacks on several high profile ciphers, including Clefia, Crypton, and Camellia.

4.2 Impossible differential attacks: generalization and improvements

4.2.1 Context

Impossible differential cryptanalysis is a very powerful attack against block ciphers introduced independently by Knudsen [Knu98] and Biham et al. [BBS99]. The idea of these attacks is to exploit impossible differentials, which are differentials occurring with probability zero. The general approach is then to extend the impossible differential by some rounds, possibly in both directions, guess the key bits that intervene in these rounds and check whether a trial pair is partially encrypted (or decrypted) to the impossible differential. In this case, we know that

the guessed key bits are certainly wrong and we can remove the corresponding key from the candidate key space. Impossible differential attacks have been successfully applied to a large variety of block ciphers, based both on the SPN and the Feistel construction. In some cases, they yield the best cryptanalysis against the targeted cipher; this is the case for the standardized Feistel cipher Camellia [LLG+12, BNS14], for example. Furthermore, impossible differential attacks were for a long time the most successful attacks against AES-128 [ZWF07, LDKK08, MDRM10].

Recently, we proposed a generalized complexity analysis of impossible differential attacks against Feistel ciphers [BNS14]. Starting from this generalized vision, several flaws in previous attacks were detected and many new attacks were proposed. In [BLNS16] we extended the analysis given in [BNS14], that has inspired since its publication new results and analyses (e.g. [Der16, BDP15, Min16, YHSL15, Blo15, LJF16]). The techniques introduced in this paper correct, complete, and improve the techniques and analyses given in [BNS14]. We showed how to combine all of these concepts in practice to mount optimized impossible differential attacks, considering also SPN ciphers.

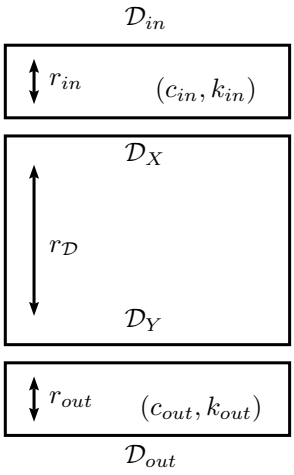
Table 4.1: Summary of flaws in previous impossible differential attacks on CLEFIA-128, Camellia, LBlock and Simon.

Algorithm	# rounds	Reference	Type of error	Gravity of error	Where discovered
CLEFIA-128 (without whit. layers)	14	[ZH08]	data complexity higher than codebook	attack does not work	[Tea09]
CLEFIA-128	13	[Tez10]	cannot be verified without implementation	-	[Blo13]
Camellia (without FL/FL^{-1} layers)	12	[WZF07]	big flaw in computation as in [WZZ08]	attack does not work	our paper
Camellia-128	12	[WZZ08]	big flaw in computation	attack does not work	[MSDB09]
Camellia-128/192/256 (without FL/FL^{-1} layers)	11/13/14	[LKKD08]	small complexity flaws	corrected attacks work	[WZF07]
LBlock	22	[MNP12]	small complexity flaw	corrected attack works	[Min13]
Simon (all versions)	14/15/15/16/16/ 19/19/22/22/22	[AL13]	data complexity higher than codebook	attacks do not work	Table 1 of [AL13]
Simon (all versions)	13/15/17/20/25/	[ALLW13, ALWL15]	big flaw in computation	attacks do not work	our paper

4.2.2 Proposing a framework

An impossible differential attack against an n-bit block cipher, parametrized by a key K of length $|k|$, starts with the discovery of an impossible differential composed of an input difference in a set $\mathcal{D}_X$ that propagates after $r_{\mathcal{D}}$ rounds to an output difference in a set $\mathcal{D}_Y$ with probability zero. After this, one extends this differential r_{in} rounds backwards to obtain a set of differences that we will denote $\mathcal{D}_{in}$ and r_{out} rounds forwards to obtain a set of differences called $\mathcal{D}_{out}$. The $\log_2$ of the size of a set $\mathcal{D}$ will be denoted by Δ.

The two appended differentials are used to eliminate the candidate keys that encrypt and decrypt data to the impossible differential. Indeed, if for a candidate key both differentials $\mathcal{D}_{in} \rightarrow \mathcal{D}_X$ and $\mathcal{D}_{out} \rightarrow \mathcal{D}_Y$ are satisfied, then this key is certainly wrong as it leads to an impossible differential and must therefore be rejected.



Two important quantities in an impossible differential attack are the total number of key bits that intervene in the appended rounds and the number of bit-conditions that must be satisfied in order to get to $\mathcal{D}_X$ from $\mathcal{D}_{in}$ and to $\mathcal{D}_Y$ from $\mathcal{D}_{out}$. We will therefore let $\#k_{in}$ (resp. $\#k_{out}$) denote the number of key bits that have to be guessed during the first (resp. last) rounds, and $|k_{in} \cup k_{out}|$ the entropy of the involved key bits when considering relations due to the key schedule. Similarly, c_{in} (resp. c_{out}) will denote the number of bit-conditions to be verified during the first (resp. last) rounds.

The probability that for a given key, a pair of inputs already satisfying the differences in $\mathcal{D}_{in}$ and $\mathcal{D}_{out}$ verifies all the $(c_{in} + c_{out})$ bit-conditions is $2^{-(c_{in}+c_{out})}$. In other words, this is the probability that for a pair of inputs satisfying the difference in $\mathcal{D}_{in}$ and whose outputs satisfy the difference in $\mathcal{D}_{out}$, a key from the possible key set is discarded. Therefore, by repeating the procedure with N different input (or output) pairs, the probability that a trial key is kept in the set of candidate keys is commonly denoted by

$$p = (1 - 2^{-(c_{in}+c_{out})})^N.$$

There is not a unique strategy for choosing the number of input (or output) pairs N. This choice principally depends on the overall time complexity, which is influenced by N, and the induced data complexity. Different trade-offs are therefore possible. A common strategy, generally used by default is to choose N such that only the right key is left after the sieving procedure. This amounts to choose p as

$$p = (1 - 2^{-(c_{in}+c_{out})})^N < \frac{1}{2^{|k_{in} \cup k_{out}|}} \ .$$

However, as shown in [BNS14], a different approach can be applied helping to reduce the number of pairs needed for the attack and to offer better trade-offs between the data and time complexity. More precisely, it is permitted to consider smaller values of N. By proceeding like this, one will be probably left with more than one key in the set of candidate keys and will need to proceed to an exhaustive search among the remaining candidates, but the total time complexity of the attack will probably be much lower. In practice, one will start by considering values of N such that p is slightly smaller than $\frac{1}{2}$ so to reduce the exhaustive search by at least one bit. So N should be chosen such that

$$p = (1 - 2^{-(c_{in}+c_{out})})^N \approx e^{-N\times 2^{-(c_{in}+c_{out})}} < \frac{1}{2}, \tag{4.1}$$

and (4.1) determines the minimal value of N. We remind here that the quantity N determines the memory complexity of the attack.

The data complexity of an attack can be determined by the following formula given in [BNS14].

$$C_N = \max\left\{\min_{\Delta\in\{\Delta_{in},\Delta_{out}\}}\left\{\sqrt{N2^{n+1-\Delta}}\right\}, N2^{n+1-\Delta_{in}-\Delta_{out}}\right\}, \tag{4.2}$$

where Δ_{in} is the number of active bits in $\mathcal{D}_{in}$ ($\log_2$ of the size of the input set) and Δ_{out} is the number of active bits in $\mathcal{D}_{out}$.

The formula provided is a lower-bound approximation of the time complexity. This is because each of the terms in this formula represents the minimum complexity of the operations that should be performed in order to accomplish each step.

By following the early abort technique, the attack consists in storing the N pairs and testing out step by step the key candidates, by reducing at each time the size of the remaining possible pairs. The time complexity is then determined by three quantities. The first term is the cost C_N, that is the amount of needed data (see (4.2)) for obtaining the N pairs, where N is such that $p < 1/2$. The second term corresponds to the number of candidate keys $2^{|k_{in}\cup k_{out}|}$, multiplied by the average cost of testing the remaining pairs. For all the applications that we have studied, this cost can be very closely approximated by $\left(N + 2^{|k_{in}\cup k_{out}|}\frac{N}{2^{c_{in}+c_{out}}}\right)C'_E$, where C'_E is the ratio of the cost of partial encryption to the full encryption. Finally, the third term is the cost of the exhaustive search for the key candidates still in the set of candidate keys after the sieving. By taking into account the cost of one encryption C_E, the approximation of the time complexity is given by

$$C_T = \left(C_N + \left(N + 2^{|k_{in}\cup k_{out}|}\frac{N}{2^{c_{in}+c_{out}}}\right)C'_E + 2^K p\right)C_E \ . \tag{4.3}$$

Obviously, as the attack complexity should be smaller than that of exhaustive search, the quantity C_T should be smaller than $2^K C_E$. We will provide a corrected time complexity formula that takes all the new improvements into account as well as the role of the key schedule.

We aim at deriving different possible trade-offs for the time, data and memory complexity of an attack. For this reason, we introduce a parameter ε offering this possibility. More precisely, we take $N = 2^{c_{in}+c_{out}+\varepsilon}$. The data and time complexity formulas are subsequently modified. Different values of ε provide different complexity trade-offs.

4.2.3 Improvements

4.2.3.1 Multiple differentials and multiple impossible differentials

The idea of multiple impossible differentials, first introduced by Tsunoo et al. [TTS+08] and later formalized in [BNS14], is to simultaneously consider several impossible differentials $(\mathcal{D}_X, \mathcal{D}_Y)$. This technique reduces the data complexity of the attack compared to a cryptanalysis that only exploits one impossible differential. This is because the use of multiple impossible differentials reduces the number of bit-conditions that need to be verified (as one has more choice), and the

number of bit-conditions directly affects the number of pairs N and thus the amount of data, as it can be seen in Eq. (4.2).

In [BLNS16], we introduce the idea of using multiple differentials and multiple impossible differentials together to further reduce the amount of data. If n_{in} is the number of input differences in $\mathcal{D}_{in}$, n_{out} the number of output differences in $\mathcal{D}_{out}$, m_{in} is the number of input differences in $\mathcal{D}_X$ and m_{out} the number of output differences in $\mathcal{D}_Y$ then the reduced data complexity by combining both techniques is

$$C'_N = \frac{C_N}{n_{in}n_{out}m_{in}m_{out}} \ . \tag{4.4}$$

This formula is directly derived from the formula for the data complexity given in [BNS14] for multiple impossible differentials.

4.2.3.2 State test technique

The aim of the state-test technique, that we introduced in [BNS14], is to eliminate some candidate keys without having to consider all of the possibilities for the involved key bits. Let us consider the value x of a word of size s of the internal part of the state needed to verify if a condition is satisfied in the second round. Typically, with a linear transformation L from the diffusion layer and an invertible S-box S, we could write $x = x' + L(S(p_i + K_i)) + K_j$, where x' is an already known value that we have computed from the knowledge of the plaintexts/ciphertexts and the already guessed key bits. The s-bit variable p_i, corresponds to the fixed part of the state, *i.e.* it has the same value for all the considered pairs. The variables K_i and K_j correspond to the not yet guessed nor determined involved parts of the key, of size s each. We easily see that if instead of guessing both variables K_i and K_j we directly guess the value $x + x'$, then we can perform the rest of the attack in a similar way, with a complexity reduced by s bits, as the number of guesses is reduced by this amount. Each guess of $x + x'$ will imply a disjoint set of possibilities for K_i and K_j, and considering all the values of $x' + x$ will provide all possible combinations of K_i and K_j. The attack is performed as before, where now we will determine the candidate values for $x + x'$. Note again that this is only possible because the value of p_i is fixed. The state-test technique can be combined with multiple (impossible) differentials.

Consider a simple attack, *i.e.* implying a single impossible differential, performed with N_s pairs. Let p_s be the proportion of candidate keys that we retain, and let C_{N_s} be the data complexity of the corresponding attack. The number of remaining key candidates is $2^{|k_{in} \cup k_{out}|} \cdot p_s 2^{K-|k_{in} \cup k_{out}|} = 2^K \cdot p_s$.

Now, suppose that we repeat this attack T times in parallel for different sets of data, possibly involving different key bits. While the parameters of the repeated attacks are the same as for the first one, the number of candidate keys left will be $(2^{|k_{in} \cup k_{out}|} \cdot p_s)^T \cdot 2^{-k_{int}}$, where k_{int} is the total number of duplicate bits from $\mathcal{K}$ when we consider all the key bits affected by all the multiple differentials together. The data complexity in this case is $T \cdot C_{N_s}$, for a proportion of keys ${p_s}^T$, and the time complexity is about $T \cdot C_{T_s}$. It is easy to see that when we perform a multiple instead of a parallel repetition we are following a similar procedure, but we can reuse the data. Therefore the data complexity of this multiple attack will be smaller, while the time and memory complexities will a priori stay the same.

It is now straightforward to combine the above representationsof the state-test and multiple impossible differentials techniques, together with taking into account the key schedule when using multiple differentials.

4.2.3.3 Including the key-schedule costs

We took into account the fact that the nature of the key schedule has an impact on the complexity of an impossible differential attack. Indeed, if the cipher's key schedule is strongly non-linear, the first few subkeys have necessarily a very complicated relation with the subkeys of the last rounds. We will take this into account in the next final formula.

4.2.3.4 Corrected generalized formula

Combining both the state-test and the multiple (impossible) differentials is now straightforward, while correctly taking into account the effect of the key schedule. Combining everything, the new time complexity formula that we propose is

$$C_T = \left(C_N + \left(N + 2^{k_A+k_B} \frac{N}{2^{c_{in}+c_{out}}} \right) C'_E + 2^K \cdot p \cdot 2^{k_A^{inv}} \cdot C'_{KS} + 2^K \cdot p \right) C_E \ , \tag{4.5}$$

where C'_{KS} is the ratio of the cost of the key schedule compared to the full encryption and k_A^{inv} denotes the number of k_A bits that are involved in at least one of the multiple differentials.

4.2.4 Applications

We have found many applications improving upon previously best known impossible differential attacks. The main results are represented in Table 4.2 for Feistel ciphers, and in Table 4.3 for SPNs.

4.2.5 Limitations of the model and related work

In order to verify and validate the applicability of the proposed techniques, we implemented two of the techniques on toy ciphers. These experiments confirm that our theoretical estimates are indeed good estimates of the complexities. However, we insist that for an exact determination of the complexity, one must perform the detailed attack step by step. The generic formula that we provided is a lower-bound approximation. This approximation is most of the time met in practice, but as shown in [Der16], some counter-examples may exist.

4.3 Meet-in-the-middle

In [CNPV13] we provided a general framework for meet-in-the-middle attacks that included bicliques and a new generic improvement of MITM algorithms, named *sieve-in-the-middle*, which allows to attack a higher number of rounds. Instead of looking for collisions in the middle, the main idea is to compute some input and output bits of a particular middle S-box S. The merging step of the algorithm then consists in efficiently discarding all key candidates which do not correspond to a valid transition through S. Intuitively, this technique allows to attack more rounds than classical MITM since it also covers the rounds corresponding to the middle S-box

Table 4.2: Summary of the best impossible differential attacks on CLEFIA-128, Camellia, LBlock and SIMON. Note here that we provide only the best of our results with respect to the time complexity. Other trade-offs can be found.

Algorithm	Rounds	Data	Time	Memory	Ref.
CLEFIA-128	13	$2^{121.2}$	$2^{117.8}$	$2^{86.8}$	[MDS11]
using state-test technique	13	$2^{116.90}$	$2^{116.33}$	$2^{83.33}$	our paper [BNS14]
using multiple impossible differentials	13	$2^{122.26}$	$\mathbf{2^{111.02}}$	$\mathbf{2^{82.60}}$	our paper [BNS14] *
combining with state-test technique	13	$\mathbf{2^{116.16}}$	$2^{114.58}$	$2^{83.16}$	our paper [BNS14] *
Camellia-128	11	2^{122}	2^{122}	2^{98}	[LLG+12]
	11	$\mathbf{2^{118.43}}$	$\mathbf{2^{118.4}}$	$\mathbf{2^{92.4}}$	our paper [BNS14] *
Camellia-192	12	$2^{187.2}$	2^{123}	$2^{155.41}$	[LLG+12]
	12	$\mathbf{2^{161.06}}$	$\mathbf{2^{119.7}}$	$\mathbf{2^{150.7}}$	our paper [BNS14] *
Camellia-256	13	$2^{251.1}$	2^{123}	2^{203}	[LLG+12]
	13	$\mathbf{2^{225.06}}$	$\mathbf{2^{119.71}}$	$\mathbf{2^{198.71}}$	our paper [BNS14] *
Camellia-256†	14	$2^{250.5}$	2^{120}	2^{120}	[LLG+12]
	14	$\mathbf{2^{220}}$	$\mathbf{2^{118}}$	2^{173}	our paper [BNS14]
LBlock	22	$2^{79.28}$	2^{58}	$2^{72.67}$	[KDH12]
	22	$\mathbf{2^{71.53}}$	2^{60}	$\mathbf{2^{59}}$	our paper [BNS14],[BMNPS14]
	23	$2^{75.36}$	2^{59}	2^{74}	our paper [BNS14],[BMNPS14]*
SIMON32/64	**19**	$2^{62.56}$	2^{32}	2^{44}	our paper [BNS14]*
SIMON48/72	**20**	$2^{70.69}$	2^{48}	2^{58}	our paper [BNS14]*
SIMON48/96	**21**	$2^{94.73}$	2^{48}	2^{70}	our paper [BNS14]*
SIMON64/96	**21**	$2^{94.56}$	2^{64}	2^{60}	our paper [BNS14]
SIMON64/128	**22**	$2^{126.56}$	2^{64}	2^{75}	our paper [BNS14]
SIMON96/96	**24**	$2^{94.62}$	2^{94}	2^{61}	our paper [BNS14]
SIMON96/144	**25**	$2^{190.56}$	2^{128}	2^{77}	our paper [BNS14]
SIMON128/128	**27**	$2^{126.6}$	2^{94}	2^{61}	our paper [BNS14]
SIMON128/192	**28**	$2^{190.56}$	2^{128}	2^{77}	our paper [BNS14]
SIMON128/256	**30**	$2^{254.68}$	2^{128}	2^{111}	our paper [BNS14]

S. This new improvement is related to some previous results, including [AS08] where transitions through an ARX construction are considered; a similar idea was applied in [KNPRS10] in a differential attack, and in [BHNS10] for side-channel attacks. This new generic improvement can be combined with bicliques, since short bicliques also allow to add a few rounds without increasing the time complexity. But, the price to pay is a higher data complexity. Also in [CNPV13], we proposed a new technique to reduce this increased data requirement by constructing some improved bicliques. This technique usually works if the key size of the cipher is larger than its block size. We refer to the original paper in appendix (page 200 of this manuscript) for this technique, and we will present here the generic model including MITM attacks, bicliques and the sieve-in-the middle improvement.

4.3.1 Framework

Meet-in-the-middle (MITM) attacks are a widely used tool introduced by Diffie and Hellman in 1977. Through the years, they have been applied for analyzing the security of a substantial number of cryptographic primitives, including block ciphers, stream ciphers and hash functions,

Table 4.3: Summary of best single-key attacks against AES-128, CRYPTON-128, ARIA-128, CLEFIA-128, Camellia-256‡ and LBlock. * Estimated memory requirements since not given in the original papers. ⋄ Incorrect result not taking into account the key-schedule. † Complexity estimated in [Mal14]. ‡ Without whitening keys and FL layers. § Additional trade-offs of the attacks in [DF13] provided by P. Derbez (private communication).

Algorithm	Rounds	Data (CP)	Time	Memory (Blocks)	Technique	Ref.
AES-128 [FIP01]	7	$2^{106.2}$	$2^{110.2}$	$2^{90.2}$	ID	[MDRM10]
	7	2^{105}	$2^{105} + 2^{99}$	2^{90}	MITM	[DFJ13]
	7	2^{97}	2^{99}	2^{98}	MITM	[DFJ13]
	7	2^{121}	$2^{121} + 2^{83}$	2^{74}	MITM §	[DF13]
	7	2^{113}	$2^{113} + 2^{75}$	2^{82}	MITM §	[DF13]
	7	$2^{113.1}$	$2^{113.1} + 2^{105.1}$	$2^{74.1}$	ID	our paper [BLNS16]
	7	2^{105}	$2^{106.88}$	2^{74}	ID	our paper [BLNS16]
CRYPTON-128 [Lim99]	7	2^{97}	$2^{97.2}$	2^{100}	Trunc. Diff.	[KHL+04]
	7	2^{121}	$2^{121} + 2^{116.2}$	2^{119} †	ID	[MSD10]
	7	$2^{114.92}$	$2^{114.92} + 2^{113.7}$	$2^{88.5}$	ID	our paper [BLNS16]
	8	2^{126}	$2^{126.2}$	2^{100}	Trunc. Diff.	[KHL+04]
ARIA-128 [KKP+04]	6	2^{113}	$2^{121.6}$	2^{113*}	ID	[LSZL08]
	6	2^{121}	$2^{121} + 2^{112}$	2^{121*}	ID	[WZF07]
	6	$2^{120.5}$	$2^{120.5} + 2^{104.5}$	2^{121*}	ID	[LSZL08]
	6	2^{120}	$2^{120} + 2^{96}$	2^{120*}	ID	[LS08]
	6	2^{111}	$2^{111} + 2^{82}$	2^{71}	ID	our paper [BLNS16]
	7	$2^{105.8}$	$2^{105.8} + 2^{100.99}$	$2^{79.73}$	LC	[LGL+11]
CLEFIA-128 [SSA+07]	13	$2^{111.02}$	$2^{122.26}$	$2^{82.6}$	ID ⋄	[BNS14]
	13	$2^{114.58}$	$2^{116.16}$	$2^{83.16}$	ID ⋄	[BNS14]
	13	$2^{114.4}$	$2^{114.4}$	2^{80}	ID	our paper [BLNS16]
	13	2^{99}	2^{99}	2^{80}	Trunc. Diff.	[LJWD15]
	14	2^{100}	2^{108}	$2^{101.3}$	Trunc. Diff.	[LJWD15]
Camellia-256‡ [AIK+00]	14	2^{120}	$2^{250.5}$	2^{120}	ID	[LLG+12]
	14	2^{118}	2^{220}	2^{173}	ID ⋄	[BNS14]
	14	$2^{117.7}$	$2^{215.7}$	$2^{166.7}$	ID	our paper [BLNS16]
LBlock [WZ11]	23	2^{59}	$2^{75.36}$	2^{74}	ID	[BNS14]
	23	$2^{63.87}$	$2^{74.30}$	2^{60}	ZC	[BM15]
	23	$2^{55.5}$	2^{72}	2^{65}	ID	our paper [BLNS16]

e.g. [CE85, Sas13, BR10, DSP07, IS12, Iso11]. They exploit the fact that some internal state in the middle of the cipher can be computed both forwards from the plaintext and backwards from the ciphertext, and that none of these computations requires the knowledge of the whole master key. The attacker then only keeps the (partial) key candidates which lead to a collision in that internal state and discards all the other keys. This generic attack has drawn a lot of attention and raised many improvements, including the partial matching, where the computed internal states are not entirely known, the technique of guessing some bits of the internal state [DSP07], the all-subkeys approach [IS12], splice-and-cut [AS08, AS09, GLRW10] and

bicliques [KRS12]. The most popular application of bicliques is an accelerated exhaustive search on the full AES [BKR11]. But, besides this degenerated application where the whole key needs to be guessed, short bicliques usually allow to increase the number of rounds attacked by MITM techniques without increasing the time complexity, but with a higher data complexity. Moreover, following [BDF11b], low-data attacks have attracted a lot of attention, motivated in part by the fact that, in many concrete protocols, only a few plaintext-ciphertext pairs can be obtained. MITM attacks belong to this class of attacks in most cases (with a few exceptions like bicliques): usually, 1 or 2 known plaintext-ciphertext pairs are enough for recovering the key.

The basic idea of our improved attack, sieve-in-the-middle, is as follows. The attacker knows one pair of plaintext and ciphertext (P, C) (or several such pairs), and she is able to compute from the plaintext and from a part K_1 of the key candidate an m-bit vector u, which corresponds to a part of an intermediate state x. On the other hand, she is able to compute from the ciphertext and another part K_2 of the key candidate a p-bit vector v, which corresponds to a part of a second intermediate state y. Both intermediate states x and y are related by $y = S(x)$, where S is a known function from $\mathbf{F}_2^n$ into $\mathbf{F}_2^{n'}$, possibly parametrized by a part K_3 of the key. In practice, S can be a classical S-box, a superbox or some more complex function, as long as the attacker is able to precompute and store all possible transitions between the input bits obtained by the forward computation and the output bits obtained backwards (or sometimes, these transitions can even be computed on the fly). In particular, the involved intermediate states x and y usually correspond to partial internal states of the cipher, implying that their sizes n and n' are smaller than the blocksize.

4.3.2 General inclusive model

Sieve-in-the-middle, as a generic technique, can be combined with other improvements of MITM attacks, in particular with bicliques [BKR11, KRS12]. We provide here a description of an attack including sieve-in-the-middle and bicliques. The general purpose of bicliques is to increase the number of rounds attacked by MITM techniques. This can be done at no computational cost, but requires a higher data complexity. In order to avoid this drawback, we proposed an improvement of bicliques which applies when the key length exceeds the block size of the cipher.

4.3.2.1 Sieve-in-the-middle and classical bicliques

The combination of both techniques is depicted on Figure 4.1: the bottom part is covered by bicliques, while the remaining part is covered by a sieve-in-the-middle algorithm.

In the following, $H_{K_8} : X \mapsto C$ denotes the function corresponding to the bottom part of the cipher, and K_8 represents the key bits involved in this part. Then, K_8 is partitioned into three disjoint subsets, K_5, K_6 and K_7. The value taken by K_i with $5 \leq i \leq 7$ will be represented by an integer in $\{0, \ldots, 2^{k_i} - 1\}$. A biclique can be built if the active[2] bits related to the variation of K_6 in the computation of $H_{K_8}(X)$ and the active bits in the computation of $H_{K_8}^{-1}(C)$ when K_5 varies are two disjoint sets. In this case, an exhaustive search over K_7 is performed and a biclique is built for each value h of K_7 as follows. We start from a given ciphertext C^0 and a chosen key $K_8^0 = (0, 0, h)$ formed by the candidate for K_7 and the zero value for K_5 and K_6.

[2]The term active bits refers to the bits affected by a certain difference.

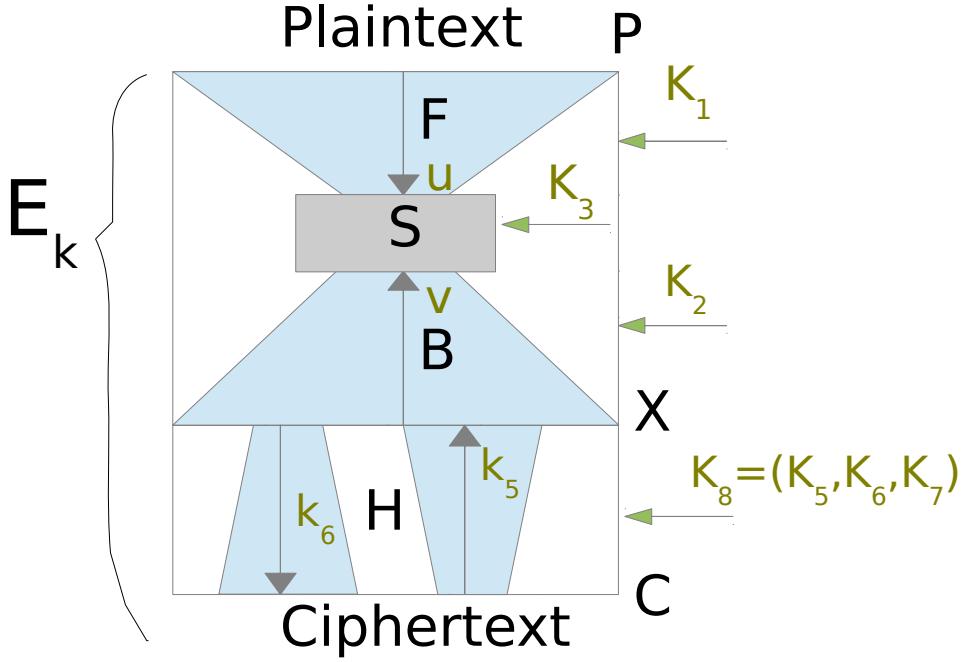


Figure 4.1: Generic representation of Sieve-in-the-Middle and bicliques

We compute $X_h^0 = H_{0,0,h}^{-1}(C^0)$. Next, we compute backwards from C^0 the intermediate state $X_h^i = H_{i,0,h}^{-1}(C^0)$ for each possible value i for K_5. Similarly, we compute forwards from X_h^0 the ciphertext $C_h^j = H_{0,j,h}(X_h^0)$ for each possible value j of K_6. Since the two differential paths are independent, we deduce that $H_{i,j,h}(X_h^i) = C_h^j$ for all values (i, j) of (K_5, K_6).

Then, the sieve-in-the-middle algorithm can be applied for each value h of K_7 and each value of $(K_1 \cap K_2)$. The list $\mathcal{L}_b$ of all output vectors v is computed backwards from X_h^i for each value i of K_5 and each value of $K_2 \setminus (K_1 \cap K_2)$. The list $\mathcal{L}_f$ of all input vectors u is computed forwards from all plaintexts P_h^j corresponding to C_h^j for each value j of K_6 and each value of $K_1 \setminus (K_1 \cap K_2)$. We then merge those two lists of respective sizes $2^{|K_2 \cup K_5|}$ and $2^{|K_1 \cup K_6|}$.

The problem of efficiently merging these list can be easily recognized as one of the problems presented in Chapter 3: we recover one list of partial inputs of the S-box (of values for u), and another of partial outputs of the S-box (of values for v), and we want to only keep the pairs that are compatible with an S-box transition (relation $\mathcal{R}$). We have to choose the algorithm that provides the best complexity in order to optimize the attack.

As in classical MITM with bicliques, the decomposition of K_8 should be such that the bits of K_5 do not belong to K_1, the bits of K_6 do not belong to K_2 and the bits of K_7 should lie in $(K_1 \cap K_2)$. The best strategy here seems to choose (K_5, K_6) such that the bits of K_5 belong to $K_2 \setminus (K_1 \cap K_2)$, and the bits of K_6 belong to $K_1 \setminus (K_1 \cap K_2)$. In this case, we have to add to the time complexity of the attack the cost of the construction of the bicliques, i.e., $2^{k_7}(2^{k_5} + 2^{k_6})c_H$ (very rarely the bottleneck), where c_H is the cost of the partial encryption

or decryption corresponding to the rounds covered by the bicliques. The main change is that the data complexity has increased since the attack now requires the knowledge of all plaintext-ciphertext pairs (P_h^j, C_h^j) corresponding to all possible values (j, h) for (K_6, K_7). The data complexity then would correspond to $2^{k_6+k_7}$ chosen ciphertexts, but it is usually smaller since the ciphertexts C_h^j only differ on a few positions, as for each value of K_7, we can chose the same first pair of plaintext and ciphertext, and then the number of different C_h^j needed will depend exclusively on the modifications produced by K_6.

4.3.2.2 Improved biclique for some scenarios

We proposed a new technique for improving bicliques in certain scenarios and reducing the data complexity to a single plaintext-ciphertext pair. The main idea is to make a reordering of the precomputations, in order to make all the transitions to come from the same state, which normally works in cases where the key size is bigger than the state. We refer to the original paper for details [CNPV13]. This was very helpful for reducing the data complexity and building an attack on 8-round PRINCE for instance, which was not possible before because the data complexity is included by the designers in their security claims.

4.3.3 Applications

We were able to apply these new improvements and techniques to four primitives which improved previously known attacks at the time[3]. We applied the sieve-in-the-middle algorithm combined with the improved biclique construction to 8 rounds (out of 12) of PRINCE, with 2 known plaintext-ciphertext pairs, while the previous best known attack was on six rounds. We also proposed a sieve-in-the-middle attack on 8 rounds (out of 32) of PRESENT, which provides a very illustrative and representative example of our technique. This attack applies up to 8 rounds, while the highest number of rounds reached by classical MITM is only 6. We provided a similar analysis on DES: our attack achieves 8 rounds, while the best previous MITM attack (starting from the first one) was on 6 rounds. We implemented the cores of these two attacks, confirming our theoretical analysis. We could also show that we can slightly improve on some platforms the speed-up factor in the accelerated exhaustive search on the full AES performed by bicliques.

4.4 Differential and truncated differential attacks

We have provided in [KLLN16b] the generalized formulas for building differential, truncated differential and linear attacks in order to be able to efficiently quantize them. To the best of our best knowledge, this is the first time such a synthetic representation was provided in particular of last-round attacks, and we believe it simplifies the application of such attacks. We provide here the generalized view of differential and truncated differential attacks, refering for linear cryptanalysis to the original paper.

[3]Since then the best known attack on PRINCE has been improved by our results from [CFG+15]

4.4.1 Differential cryptanalysis

Differential cryptanalysis was introduced in [BS91] by Biham and Shamir. It studies the propagation of a difference (δ_{in}) in the input of a function and its influence on the generated output difference (δ_{out}). In this section, we present a generalized version of the two main types of differential attacks on block ciphers: the *differential distinguisher* and the *last-round attack*.

We denote by n the block-size, k the key-size and h_S the probability ($-\log$) of the differential characteristic. Differential attacks exploit the fact that there exists an input difference δ_{in} and an output difference δ_{out} to a cipher E such that

$$h_S := -\log \Pr_x[E(x \oplus \delta_{\text{in}}) = E(x) \oplus \delta_{\text{out}}] < n \ , \tag{4.6}$$

i.e., such that we can detect some non-random behaviour of the differences of plaintexts x and $x \oplus \delta_{\text{in}}$. Here, "$\oplus$" represents the bitwise xor of bit strings of equal length. The value of h_S is generally considered on average over all keys, and as usual in the literature, we will assume that Eq. (4.6) approximately holds for the secret key[4]. Such a relation between δ_{in} and δ_{out} is typically found by studying the internal structure of the primitive in detail.

4.4.1.1 Differential distinguisher

This non-random behaviour can already be used to attack a cryptosystem by distinguishing it from a random function. This distinguisher is based on the fact that, for a random function and a fixed δ_{in}, obtaining the δ_{out} difference in the output would require 2^n trials, where n is the block size. On the other hand, for the cipher E, if we collect 2^{h_S} input pairs verifying the input difference δ_{in}, we can expect to obtain one pair of outputs with output difference δ_{out}. The complexity of such a distinguisher exploiting Eq. (4.6) is 2^{h_S+1} in both data and time, and is negligible in terms of memory:

$$T_{\text{C}}^{\text{s. dist.}} = D_{\text{C}}^{\text{s. dist.}} = 2^{h_S+1}. \tag{4.7}$$

Here, *s. dist.* refers to "simple distinguisher" by opposition to its truncated version later in the text.

Assuming that such a distinguisher exists for the first R rounds of a cipher, we can transform the attack into a key recovery on more rounds by adding some rounds at the end or beginning of the cipher. This is called a *last-round attack*, and allows to attack more rounds than the distinguisher, typically one or two, or even more depending on the cipher.

4.4.1.2 Last-round attack

We denote by Δ_{in} the (log) size of the set of input differences respectively, Δ_{fin} the (log) size of the set of differences $\mathcal{D}_{\text{fin}}$ after the last rounds, h_{out} the probability ($-\log$) of generating δ_{out} from $\mathcal{D}_{\text{fin}}$, k_{out} the number of key bits required to invert the last rounds, $C_{k_{\text{out}}}$ the cost of recovering the last round subkey from a good pair.

For simplicity and without loss of generality, we consider that the rounds added to the distinguisher are placed at the end. We attack a total of $r = R + r_{\text{out}}$ rounds, where R is the number of rounds covered by the distinguisher. The main goal of the attack is to reduce the key

[4] see for instance [DR07, BBL13] for a discussion on this topic.

space that needs to be searched exhaustively from 2^k to some $2^{k'}$ with $k' < k$. For this, we use the fact that we have an advantage for finding an input x such that $E^{(R)}(x) \oplus E^{(R)}(x \oplus \delta_{\text{in}}) = \delta_{\text{out}}$.

For a pair that generates the difference δ_{out} after R rounds, we denote by $\mathcal{D}_{\text{fin}}$ the set of possible differences generated in the output after the final r_{out} rounds and the size of this set by $2^{\Delta_{\text{fin}}} = |\mathcal{D}_{\text{fin}}|$. Let $2^{-h_{\text{out}}}$ denote the probability of generating the difference δ_{out} from a difference in $\mathcal{D}_{\text{fin}}$ when computing r_{out} rounds in the backward direction, and by k_{out} the number of key bits involved in these rounds. The goal of the attack is to construct a list L of candidates for the partial key that contains almost surely the correct value, and that has size strictly less than $2^{k_{\text{out}}}$. For this, one starts with lists L_M and L_K where L_M is a random subset of 2^{h_S} possible messages and L_K contains all possible k_{out}-bit strings. From Eq. (4.6), the list L_M contains an element x such that $E^{(R)}(x) \oplus E^{(R)}(x \oplus \delta_{\text{in}}) = \delta_{\text{out}}$ with high probability. Let us apply two successive tests to the lists.

The first test keeps only the $x \in L_M$ such that $E(x) \oplus E(x \oplus \delta_{\text{in}}) \in \mathcal{D}_{\text{fin}}$. The probability of satisfying this equation is $2^{\Delta_{\text{fin}}-n}$. This gives a new list L'_M of size $|L'_M| = 2^{h_S+\Delta_{\text{fin}}-n}$. The cost of this first test is 2^{h_S+1}.

The second test considers the set $L'_M \times L_K$ and keeps only the pairs (x, κ) such that $E_\kappa^{(R)}(x) + E_\kappa^{(R)}(x + \delta_{in}) = \delta_{\text{out}}$. This is done by computing backward the possible partial keys for a given difference in δ_{out}. Denote $C_{k_{\text{out}}}$ the average cost of generating those keys for a given input pair. Notice that $C_{k_{\text{out}}}$ can be 1 when the number of rounds added is reasonably small[5], and is upper bounded by $2^{k_{\text{out}}}$, that is, $1 \leq C_{k_{\text{out}}} \leq 2^{k_{\text{out}}}$. For a random pair (x, κ), the probability of passing this test is $2^{-h_{\text{out}}}$. The size of the resulting set is therefore expected to be $2^{-h_{\text{out}}} \times |L'_M| \times |L_K| = 2^{h_S+\Delta_{\text{fin}}-n+k_{\text{out}}-h_{\text{out}}}$. The cost of this step is $C_{k_{\text{out}}} 2^{h_S+\Delta_{\text{fin}}-n}$.

The previous step produces a list of candidates for the partial key corresponding to the key bits involved in the last r_{out} rounds and leading to a difference δ_{out} after R rounds. The last step of the attack consists in performing an exhaustive search within all partial keys in this set completed with all possible $k - k_{\text{out}}$ bits. The cost of this step is $2^{h_S+\Delta_{\text{fin}}-n+k-h_{\text{out}}}$.

In practice, the lists do not need to be built and everything can be performed "on the fly". Consequently, memory needs can be made negligible. The total time complexity is:

$$T_{\text{C}}^{\text{s. att.}} = 2^{h_S+1} + 2^{h_S+\Delta_{\text{fin}}-n} \left(C_{k_{\text{out}}} + 2^{k-h_{\text{out}}} \right), \tag{4.8}$$

while the data complexity of this classical attack is $D_{\text{C}}^{\text{s. att.}} = 2^{h_S+1}$. By definition, the attack is more efficient than an exhaustive search if $T_{\text{C}}^{\text{s. att.}} < 2^k$.

4.4.2 Truncated Differential Cryptanalysis

Truncated differential cryptanalysis was introduced by Knudsen [Knu95] in '94. Instead of fixed input and output differences, it considers sets of differences.

We assume in the following that we are given two sets $\mathcal{D}_{\text{in}}$ and $\mathcal{D}_{\text{out}}$ of input and output differences such that the probability of generating a difference in $\mathcal{D}_{\text{out}}$ from one in $\mathcal{D}_{\text{in}}$ is 2^{-h_T}. We further consider that $\mathcal{D}_{\text{in}}$ and $\mathcal{D}_{\text{out}}$ are vector spaces.

As in the simple differential case, we first present the differential distinguisher based on the non-random property of the differences behaviour, and then discuss the last-round attack obtained from the truncated differential distinguishers.

[5] For example, using precomputation tables with the values that allow the differential transitions through the S-Boxes.

4.4.2.1 Truncated differential distinguisher

We denote by Δ_{in} and Δ_{out} the (log) size of the set of input and output differences respectively, Δ_{fin} the (log) size of the set of differences $\mathcal{D}_{\text{fin}}$ after the last rounds, h_{out} the probability ($-\log$) of generating a difference in Δ_{out} from $\mathcal{D}_{\text{fin}}$, k_{out} the number of key bits required to invert the last rounds, $C_{k_{\text{out}}}$ the cost of recovering the last round subkey from a good pair.

Let $2^{\Delta_{\text{in}}}$ and $2^{\Delta_{\text{out}}}$ denote the sizes of the input and output subspaces of differences, respectively. For simplicity and without loss of generality, we assume to have access to an encryption oracle, and therefore only consider the truncated differential as directed from input to output[6]. We denote by 2^{-h_T} the probability of generating a difference in $\mathcal{D}_{\text{out}}$ from one in $\mathcal{D}_{\text{in}}$. The condition for the distinguisher to work is that $2^{-h_T} > 2^{\Delta_{\text{out}}-n}$. In this analysis, we assume that $2^{-h_T} \gg 2^{\Delta_{\text{out}}-n}$.

The advantage of truncated differentials is that they allow the use of structures, *i.e.*, sets of plaintext values that can be combined into input pairs with a difference in $\mathcal{D}_{\text{in}}$ in many different ways: one can generate $2^{2\Delta_{\text{in}}-1}$ pairs using a single structure of size $2^{\Delta_{\text{in}}}$. This reduces the data complexity compared to simple differential attacks.

Two cases need to be considered. If $\Delta_{\text{in}} \geq (h_T+1)/2$, we build a single structure $\mathcal{S}$ of size $2^{(h_T+1)/2}$ such that for all pairs $(x, y) \in \mathcal{S} \times \mathcal{S}$, $x \oplus y \in \mathcal{D}_{\text{in}}$. This structure generates 2^{h_T} pairs. If $\Delta_{\text{in}} \leq (h_T+1)/2$, we have to consider multiple structures $\mathcal{S}_i$. Each structure contains $2^{\Delta_{\text{in}}}$ elements, and generates $2^{2\Delta_{\text{in}}-1}$ pairs of elements. We consider $2^{h_T-2\Delta_{\text{in}}+1}$ such structures in order to have 2^{h_T} candidate pairs.

In both cases, we have 2^{h_T} candidate pairs. With high probability, one of these pairs (x, y) shall satisfy $E(x) \oplus E(y) \in \mathcal{D}_{\text{out}}$, something that should not occur for a random function if $2^{-h_T} \gg 2^{\Delta_{\text{out}}-n}$. Therefore detecting a single valid pair gives an efficient distinguisher.

The attack then works by checking if, for a pair generated by the data, the output difference belongs to $\mathcal{D}_{\text{out}}$. Since $\mathcal{D}_{\text{out}}$ is assumed to be a vector space, this can be reduced to trying to find a collision on $n - \Delta_{\text{out}}$ bits of the output (*i.e.* on the restrictions of the output to the complementary of $\mathcal{D}_{\text{out}}$). Once the data is generated, looking for a collision is not expensive (*e.g.* using a hash table), which means that time and data complexities coincide:

$$D_{\text{C}}^{\text{tr. dist.}} = T_{\text{C}}^{\text{tr. dist.}} = \max\{2^{(h_T+1)/2}, 2^{h_T-\Delta_{\text{in}}+1}\} \ . \tag{4.9}$$

4.4.2.2 Last-round attack

Last-round attacks work similarly as in the case of simple differential cryptanalysis. For simplicity, we assume that r_{out} rounds are added at the end of the truncated differential. The intermediate set of differences is denoted $\mathcal{D}_{\text{out}}$, and its size is $2^{\Delta_{\text{out}}}$. The set $\mathcal{D}_{\text{fin}}$, of size $2^{\Delta_{\text{fin}}}$ denotes the possible differences for the outputs after the final round. The probability of reaching a difference in $\mathcal{D}_{\text{out}}$ from a difference in $\mathcal{D}_{\text{in}}$ is 2^{-h_T}, and the probability of reaching a difference in $\mathcal{D}_{\text{out}}$ from a difference in $\mathcal{D}_{\text{fin}}$ is $2^{-h_{\text{out}}}$. Applying the same algorithm as in the simple differential case, the data complexity remains the same as for the distinguisher:

$$D_{\text{C}}^{\text{tr. att.}} = \max\{2^{(h_T+1)/2}, 2^{h_T-\Delta_{\text{in}}+1}\} \ . \tag{4.10}$$

[6] In the case where the other direction provides better complexities, we could instead perform queries to a decryption oracle and change the roles of input and output in the attack. We assume that the most interesting direction has been chosen.

The time complexity in this case is:

$$T_{\mathrm{C}}^{\mathrm{tr.\,att.}} = \max\{2^{(h_T+1)/2}, 2^{h_T-\Delta_{\mathrm{in}}+1}\} + 2^{h_T+\Delta_{\mathrm{fin}}-n}\left(C_{k_{\mathrm{out}}} + 2^{k-h_{\mathrm{out}}}\right) \ , \tag{4.11}$$

where $C_{k_{\mathrm{out}}}$ is the average cost of finding all the partial key candidates corresponding to a pair of data with a difference in $\mathcal{D}_{\mathrm{out}}$. As mentioned earlier, $C_{k_{\mathrm{out}}}$ ranges from 1 to $2^{k_{\mathrm{out}}}$.

4.5 Conclusion

We have shown several examples of how providing a generalized view allows to improve our understanding, to correct errors, to find improvements and to ease the applications of the attacks. This implies an important step forward for designers and cryptanalysts. This important task of generalizing families of cryptanalysis is also of big help for providing automated tools to apply the attacks.

Future work in this direction that I plan to pursue is: generalization and study of zero-correlation attacks in order to study to which extend can we apply improvements of impossible differential attacks like the state test technique; generalization of all-subkeys and multidimensional meet-in-the-middle attacks and relations with previous improvements; and also extending our generalization of differential and truncated differential attacks to include several technical improvements, as neutral bits and conditional differentials, to this generalization.

As we will describe in the next chapter, the generalization of families of cryptanalysis is also necessary in order to provide quantized version of known attacks, *i.e.* attacks accelerated by using quantum computers.

Chapter 5

Post-quantum cryptanalysis of symmetric primitives

Contents

Two years ago I started studying the consequences of the existence of quantum computers on symmetric cryptography, which is an important but quite understudied topic. I have obtained so far three main results, published at Crypto16 [KLLN16a], in the IACR ToSC journal [KLLN16b] and under submission [BNP17], that show that much work needs yet to be done: Some solid and reliable constructions in the classic world would become completely insecure in a post-quantum setting.

5.1 Post-quantum symmetric cryptography

As years go by, the existence of quantum computers becomes more tangible.[1] Governments and large private companies such as Google, Microsoft, and IBM are willing to invest considerable amounts of resources to make it occur. Though a universal quantum computer still seems far from reality, the scientific community is already anticipating the enormous consequences of the induced breakthrough in computational power (*e.g.* [Ber11]). Indeed, this ground-breaking achievement would shake the foundations of several disciplines, including cryptology. Furthermore, in this case, the dangers are also related to long-term pre-quantum

[1] See the recent article [New17].

secrets: today's encrypted information would become available to non-authorized eyes; hence the interest in switching in advance to post-quantum secure systems, in order to protect our existing confidential information from future quantum attackers.[2]

Hybrid systems: unknown impact. Before the emergence of full quantum computers, we expect that hybrid systems (*i.e.* not a full quantum computer but rather some dedicated quantum modules within a classical computer) will become increasingly available. These systems will have a significantly increased computational power with respect to classical computers, and pose a more imminent threat. Post-quantum cryptography, that resists a full quantum computer, is also the solution for this, instead of partial and temporary measures.

Cryptography in the post-quantum world. One of the most popular asymmetric algorithms used nowadays is RSA [RSA78]. The arrival of the quantum computer, where factorization stops being a hard problem to solve, would mean that, for instance, the RSA algorithm could no longer be used securely.[3] Indeed, in the 90s Shor [Sho97] proposed a polynomial-time algorithm for efficiently solving factorization and discrete logarithms with a quantum computer. That is why post-quantum cryptography has experienced an impressive boom. Very hot topics in the cryptographic community include lattice-based cryptography, multivariate cryptography or code-based cryptography. Their security would continue in the quantum world because they do not rely on number theory, though their performances and features do not yet compete with RSA. The American institute of standards (NIST), which decides most of the world-wide used standards, is deeply concerned by this topic and actively seeks alternatives, as shown by their recent call for primitives.

The situation of symmetric primitives is very different. Until now, cryptographers have mainly only considered the security of the ideal primitives in the post-quantum world. Indeed, Grover's algorithm [Gro96], which allows us to search a database of size N in $O(N^{1/2})$ time with a quantum computer, can be applied to any generic exhaustive search, reducing the complexity by a square root. Therefore, the cryptographic community widely believes that doubling the key lengths (or hash lengths) would be enough to continue having secure symmetric algorithms [BBD09], and consider the topic settled. It is worth noticing that, while Shor's algorithm attacks RSA by exploiting the specificities of the primitive, the acceleration of generic attacks has nearly only been considered so far for symmetric primitives.

Powerful adversaries may own in the future quantum computers, but—at least for a long time—individual users won't. Therefore, though we want to have cryptosystems resistant to an adversary with access to a quantum computer, we still need lightweight, small, and performant cryptosystems that fit in our current devices because of implementation constraints. Let us point out here that, though NIST recognizes the importance of lightweight and post-quantum cryptography, as shown by the two previously mentioned workshops, the set of primitives satisfying the intersection of these two needs (which are consequently of enormous importance) is empty.

[2] For instance, if data has to be secret for 15 years, and migration to post-quantum cryptography costs 5 years, we should start migrating today if we expect the first large quantum computer in 2037.

[3] This is also the case for DH and ECDH.

We definitely have a lot of work to do with respect to symmetric cryptography. As symmetric cryptography completely depends on the ever-changing landscape of symmetric cryptanalysis, it is not possible to determine whether doubling the key length might make a concrete cipher secure in a post-quantum world without first understanding how a quantum adversary could attack the symmetric primitive.

Lately, new results in this direction have appeared: quantum generic meet-in-the-middle attacks on iterative block ciphers [Kap14], quantum linear and differential attacks [KLLN16b], or even recent quantum-secure constructions [GHS16]. Some other recent attacks are based on the quantum algorithm of Simon [Sim97], like [KM10, KM12, RS15] that respectively analyze the post-quantum security of 3-round Feistel schemes, the Even-Mansour construction and related-key attacks.

5.1.1 Attacker model: Quantum superposition queries.

Many of the recently appeared attacks apply in a scenario of superposition quantum queries.[4] That means that the adversary is not only allowed to perform local computations on a quantum computer[5], but is also allowed to perform superposition queries to a remote quantum cryptographic oracle, and is able to obtain the superposition of the outputs. These attacks have been described in several works as *superposition attacks* [DFNS14], *quantum chosen message attacks* [BZ13b] or *quantum security* [Zha12]. This scenario was also considered in [BZ13b] and [DFNS14], where secure constructions were provided with respect to superposition attacks.

This is a strong model for the attacker, but there are very good arguments for defending the fact that symmetric primitives should be secure in this setting, *i.e.*, that symmetric post-quantum cryptanalysis should be considered in this setting:[6]

1. This model is simple. Using another model would imply artificial and hard to respect measures with respect to cryptographic oracles in a world with quantum resources, with complex manipulations of yet uncertain outcome [7].

2. Security in this model implies security in any other scenario (including Hybrid ones). It includes any other model of quantum attacks, even the ones from advanced scenarios (*e.g.* obfuscated algorithms).

3. Though powerful, this model is not trivial: it doesn't make all primitives trivially breakable. Several primitives or constructions resistant in this model have actually been proposed, such as [BZ13b].

5.1.2 Summary of first results

I have published three papers on this topic. The first one [KLLN16b] quantizes differential, truncated differential and linear cryptanalysis, and allows us to deduce some new insights. The

[4] This model is used in this chapter.

[5] For instance in [BDF+11a, BHK+11, Zha15, Unr15] the adversary can query a quantum random oracle with arbitrary superpositions of the inputs

[6] Note that the two previous affirmations are equivalent: if we want symmetric primitives to be secure in this scenario, we have to analyze their security with respect to this setting.

[7] Implementations of theoretically secure quantum cryptography remain yet not fully understood, as shown by the attacks [ZFQ+08, LWW+10, XQL10]

result from [KLLN16a], includes two very exciting and surprising outcomes: it provides for the first time an exponential speed up of a classical cryptanalysis technique (slide attacks), while also showing that secure and widely used classical constructions, such as CBC-MAC, can be completely insecure in the post-quantum world. These encouraging results have convinced me of the enormous importance of solving the questions previously raised, and of all the unexpected possibilities that we might encounter, and which should be known and taken into account when designing symmetric cryptography for the post-quantum world. They have also corroborated my initial impression: this research needs to be done by a symmetric cryptanalyst, due to the technical knowledge needed on symmetric cryptography. In more recent work under submission [BNP17], we extend these last results in the case where modular additions are used instead of XORs. I will provide next a description of the two last results.

5.2 Using Simon's algorithm in Symmetric Cryptanalysis

In [KLLN16a] we showed that slide attacks, a well-known family of cryptanalysis, will benefit from an exponential speedup in a post-quantum setting, due to a variant of Simon's algorithm.

Simon's algorithm efficiently solves Simon's problem:
Given a Boolean function $f : \{0,1\}^n \to \{0,1\}^n$ *and the promise that there exists* $s \in \{0,1\}^n$ *such that for any* $(x, y) \in \{0,1\}^n$, $[f(x) = f(y)] \Leftrightarrow [x \oplus y \in \{0^n, s\}]$, *the goal is to find* s.
In [KLLN16a] we rewrote slide attacks in order to be able to apply Simon's algorithm to find the secret key k in linear time in the block size ($O(n)$ instead of the $O(2^{n/2})$ in the classical world). However, we realized that both implications in the problem were not verified, as the left implication is not true: with a relatively small probability, some cases where $f(x) = f(y)$ might appear where $x \oplus y \notin \{0^n, s\}$. We have been able to show that, if we make this probability small enough by chaining slide pairs for instance, then we can still apply Simon's algorithm with the same complexity even though the premises of the problem are not completely satisfied. It is important to point out that here we have been able to exploit Simon's algorithm not as a black box, but by adapting it to our needs. Slide attacks are much more accelerated than generic attacks (quadratic acceleration with Grover), which makes slide attacks a very powerful tool in the post-quantum world.

Also in [KLLN16a], we found very efficient post-quantum forgery attacks (with linear complexity) on many authenticated encryption schemes, including CBC-MAC or OCB. These attacks were successful because we were able to reduce them to solving Simon's problem (through some elaborate transformations). This shows that secure and widely-used constructions in the classical world, like CBC-MAC, can be completely insecure in the post-quantum one: doubling the key length is far from enough to preserve an equivalent ideal security.

Simon's with approximate promise. As previously pointed out, it is interesting to note that we do not apply Simon as a black box, but we are able to adapt it to our situation. In our cryptanalysis scenario, it is not always the case that the promise of Simon's problem is perfectly satisfied. More precisely, by construction, there will always exist an s such that $f(x) = f(x \oplus s)$ for any input x, but there might be many more collisions than those of this form. If the number of such unwanted collisions is too large, one might not be able to obtain a full rank linear

system of equations from Simon's subroutine after $O(n)$ queries. We have been able to show in [KLLN16a] that in the cases that interest us, we could bound the number of such unwanted collisions by a small enough number, in order to retrieve the solution without additional cost.

5.2.1 Simon's algorithm on constructions: Example CBC-MAC

5.2.1.1 CBC-MAC.

CBC-MAC is one of the first MAC constructions, inspired by the CBC encryption mode. Since the basic CBC-MAC is only secure when the queries are prefix-free, there are many variants of CBC-MAC to provide security for arbitrary messages. In the following we describe the Encrypted-CBC-MAC variant [BKR00], using two keys k and k', but the attack can be easily adapted to other variants [BR00, IK03, Dwo05]. On a message $M = m_1 \| \dots \| m_\ell$, CBC-MAC is defined as depicted on Figure 5.1): $x_0 = 0$, $x_i = E_K(x_{i-1} + m_i)$ and CBC-MAC$(M) = E_{k'}(x_l)$.

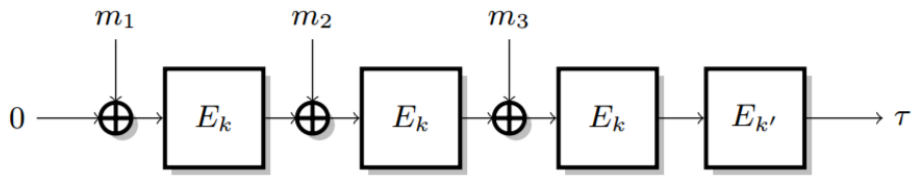


Figure 5.1: Encrypt-last-block CBC-MAC

CBC-MAC is standardized and widely used. It has been proved to be secure up to the birthday bound [BKR00], assuming that the block cipher is indistinguishable from a random keyed permutation.

5.2.1.2 Attack.

We can build a powerful forgery attack on CBC-MAC with very low complexity using superposition queries. We fix two arbitrary message blocks α_0, α_1, with $\alpha_0 \neq \alpha_1$, and we define the following function:

$$\begin{aligned} f : \{0,1\} \times \{0,1\}^n &\to \{0,1\}^n \\ (b, x) &\mapsto \text{CBC-MAC}(\alpha_b \| x) = E_{k'}\left(E_k\big(x \oplus E_k(\alpha_b)\big)\right). \end{aligned}$$

The function f can be computed with a single call to the cryptographic oracle, and we can build a quantum circuit for f given a black box quantum circuit for CBC-MAC$_k$. Moreover, f satisfies the promise of Simon's problem with $s = 1 \| E_k(\alpha_0) \oplus E_k(\alpha_1)$:

$$\begin{aligned} f(0, x) &= E_{k'}(E_k(x \oplus E_k(\alpha_0))), \\ f(1, x) &= E_{k'}(E_k(x \oplus E_k(\alpha_1))), \\ f(b, x) &= f(b \oplus 1, x \oplus E_k(\alpha_0) \oplus E_k(\alpha_1)). \end{aligned}$$

More precisely:

$$f(b', x') = f(b, x) \Leftrightarrow x \oplus E_k(\alpha_b) = x' \oplus E_k(\alpha_{b'})$$
$$\Leftrightarrow \begin{cases} x' \oplus x = 0 & \text{if } b' = b \\ x' \oplus x = E_k(\alpha_0) \oplus E_k(\alpha_1) & \text{if } b' \neq b \end{cases}$$

Therefore, an application of Simon's algorithm returns $E_k(\alpha_0) \oplus E_k(\alpha_1)$. This allows to forge messages easily:

1. Query the tag of $\alpha_0 \| m_1$ for an arbitrary block m_1;
2. The same tag is valid for $\alpha_1 \| m_1 \oplus E_k(\alpha_0) \oplus E_k(\alpha_1)$.

In order to break the formal notion of EUF-qCMA security from [BZ13a], we must produce $q+1$ valid tags with only q queries to the oracle. Let $q' = O(n)$ denote the number of quantum queries made to learn $E_k(\alpha_0) \oplus E_k(\alpha_1)$. The attacker will repeat the forgery step $q'+1$ times, in order to produce $2(q'+1)$ messages with valid tags, after a total of $2q'+1$ classical and quantum queries to the cryptographic oracle. Therefore, CBC-MAC is broken by a quantum existential forgery attack.

After some exchange at early stages of the work, an extension of this forgery attack has been found by Santoli and Schaffner [SS16]. Its main advantage is to handle oracles that accept inputs of fixed length, while our attack works for oracles accepting messages of variable length.

5.2.2 Simon's algorithm on slide attacks: Example on key-alternating ciphers

5.2.2.1 Slide attacks

In 1999, Wagner and Biryukov introduced the technique called *slide attack* [BW99]. It can be applied to block ciphers made of r applications of an identical round function R, each one parametrized by the same key K. The attack works independently of the number r of rounds. Intuitively, for the attack to work, R has to be vulnerable to known plaintext attacks.

The attacker collects $2^{n/2}$ encryptions of plaintexts. Amongst these couples of plaintext-ciphertext, with large probability, he gets a "slid" pair, that is, a pair of pairs (P_0, C_0) and (P_1, C_1) such that $R(P_0) = P_1$. This immediately implies that $R(C_0) = C_1$. For the attack to work, the function R needs to allow for an efficient recognition of such pairs, which in turns makes the key extraction from R easy. This attack trivially applies to the key-alternating cipher with blocks of n bits, identical subkeys and no round constants. The complexity is then approximately $2^{n/2}$. The speed-up over exhaustive search given by this attack is then quadratic, similar to the quantum attack based on Grover's algorithm.

5.2.2.2 Applying Simon's algorithm

In [BNP17] improved slide attacks on other constructions different from key-alternating ciphers are presented, but we will detail here, as an example, the simplest one. We consider the attack represented in Figure 5.2. The unkeyed round function is denoted F and the whole encryption

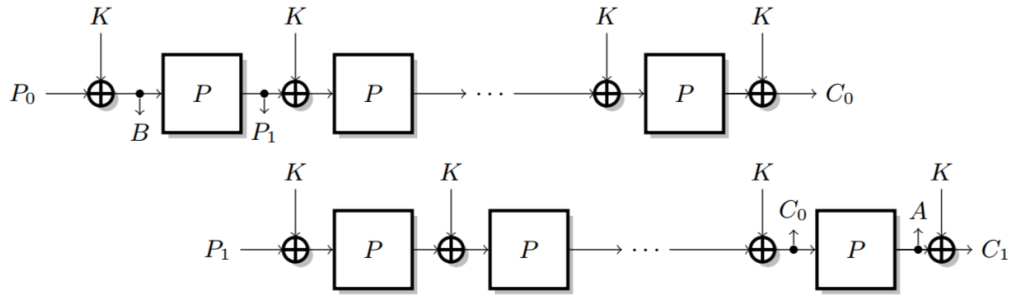


Figure 5.2: The slide attack on the key-alternating cipher

function E_k. We define the following function:

$$f : \{0,1\} \times \{0,1\}^n \to \{0,1\}^{n+1}$$
$$(b,x) \mapsto \begin{cases} F(E_K(x)) \oplus x & \text{if } b = 0 \\ E_K(F(x)) \oplus x & \text{if } b = 1 \end{cases}$$

The main property used in a slide attack is that $E_K(F(x \oplus K)) = F(E_K(x)) \oplus K$, and it satisfies Simon's promise for $s = 1 \| K$ as the function f has been defined so that $f(0,x) = f(1, x \oplus K)$. Indeed, we have

$$f(0,x) = F(E_K(x)) \oplus x = E_K(F(x \oplus K)) \oplus K \oplus x = f(1, x \oplus K)$$

We associate $b = 0$ to the first path, $P = P_0$ and we have that $F(E_K(P_0)) = A$. For $b = 1$ we consider the second path and we have that $P = B$, and then $E_K(F(P)) = C_1$. When P_0 and P_1 form a slide pair, we have that $P_0 \oplus B = K$ and that $A \oplus C_1 = K$. Consequently, we will have a collision through f when P_0 (for $b = 0$) and B (for $b = 1$) correspond to a slid pair, implying that Simon's algorithm returns $P_0 \oplus B = K$, allowing to recover the key with a complexity of about $O(n)$, instead of the $O(2^{n/2})$ of the classical attack.

5.2.3 Conclusion

We have been able to show that symmetric cryptography is far from ready for the quantum world. We have found exponential speed-ups on attacks on symmetric cryptosystems. In consequence, some cryptosystems that are believed to be safe in a classical world become vulnerable in a quantum world.

With the speed-up on slide attacks, we provided the first known exponential quantum speed-up of a classical attack. This attack now becomes very powerful. An interesting follow-up would be to seek other such speed-ups of generic techniques. For authenticated encryption, we have shown that many modes of operations that are believed to be solid and secure in the classical world, become completely broken in the post-quantum world. More constructions might be broken following the same ideas.

5.3 Using Kuperberg's algorithm in symmetric cryptanalysis

Together with my PhD student Xavier Bonnetain, we have studied in [BNP17] the effect of replacing the xors in the previous primitives with modular additions, as recently proposed in [AR17] as a meassure to counter Simon's attacks. Lets first provide some context.

5.3.1 Countering the Simon attacks: new proposal [AR17].

In [AR17], to appear at Eurocrypt 2017, a proposal for countering the Simon attacks is given. The authors propose to replace the common $\mathbb{F}_2^n$ addition, vulnerable to the Simon algorithm, with other operations that imply a harder problem to solve, in the context of such attacks. The most promising of these operations, because of efficiency and implementations issues, and which is already used in several symmetric schemes (*i.e.* [RRY00, Yuv97, NBoS89]), is addition over $\mathbb{Z}/2^n\mathbb{Z}$, i.e. modular addition. The authors claim the quantum hardness of the hidden shift problem as evidence demonstrating the security of their new proposal against quantum chosen plaintext attacks. This modification is proposed for resisting to the attacks on operating modes as well as to the slide attacks.

This approach is a priori an interesting direction to analyze and study. Unfortunately, the authors did not provide a more profound analysis of the impacts of various parameters on the security. Indeed, the complexities of the attacks are no longer $O(n)$ (with n being the state size) when using the modular addition, but we can describe attacks that are still a lot faster than the generic ones, e.g. $O(2^{\sqrt{n}})$ instead of $O(\sqrt{2^n})$.

Classically, a symmetric primitive is considered secure when no attack better than the generic attack exists. While the complexity of the generic exhaustive search attack is exponential ($2^{n/2}$ with Grover's algorithm), the quantum Simon-like attacks on primitives with modular additions have a sub-exponetial complexity of $O(2^{\sqrt{n}})$. This implies a need for a redefinition of *security*, when building *secure* primitives with these counter-measures. Also, concrete proposals providing the dimensions of the primitives needed in order to guarantee the typical security needs (*i.e.* 128 bits) are missing. In our opinion, these were the next steps to follow to decide whether these proposals can be seriously considered, or whether more analysis is needed. Describing in detail the new best quantum attacks on the proposed constructions would be of interest, and actually necessary to provide designs with concrete parameters, which we could then compare to other existing (and quantum-secure) ones.

5.3.2 Studying Kuperberg's algorithm

Kuperberg's algorithm: implementation, verification, improvement, estimation. In [BNP17] we studied Kuperberg's quantum algorithm for hidden shifts in the group $\mathbb{Z}/N\mathbb{Z}$ [Kup05] and its applications in symmetric cryptography. We focused on the original algorithm, and not on the later ones [Reg04, Kup13] because they are far more difficult to simulate with a classical computer. Moreover, we were mainly interested in the complexity in number of queries, and the gain in [Reg04] is only in memory. We limited ourselves to the groups $\mathbb{Z}/2^n\mathbb{Z}$, which are the ones widely used in symmetric cryptography. The original algorithm retrieves one bit of the secret shift at a time and uses a reducibility property to get the next bit. We propose a variant that performs better. We also propose a generalisation for products of cyclic groups ($(\mathbb{Z}/2^p\mathbb{Z})^w$ and its subgroups), and see that the problem is more easily solvable in these groups

than in $\mathbb{Z}/2^{pw}\mathbb{Z}$. This generalisation is common in symmetric primitives, where p represents the number of modular additions done in parallel, and w the size of the words, being $n = pw$.

We have implemented the classical part of these algorithms and simulated them in order to estimate the asymptotic query complexity. We could determine that the concrete complexity of our tweaked version is $2^{1.8\sqrt{n}}$, which is small enough for a practical use on typical parameters of n. We also have adapted the algorithm to the general case where several parallel additions are performed, and provided an estimate of the complexity.

5.3.3 Analysis and conclusions on parameters of possible tweaks

The authors of [AR17] provide some nice ideas for preventing Simon-based attacks. Due to implementation constraints for symmetric primitives, we believe that the most interesting is the use of modular additions, which has already been well investigated [RRY00, Yuv97, NBoS89]. Based on the results presented in the previous sections, we can now correctly size some of the primitives that were broken using Simon-based algorithms, now patched to use modular additions, in order to provide a desired post-quantum security. Considering the complexities of the attacks and of some advanced slide attacks from [BNP17] we have built Tables 5.2 and 5.3 that show how big the parameters of such constructions should be.

Let us point out that we used a slightly unconventional definition of the *security*: we consider a cipher to provide a security of Q bits when no attack of complexity lower than 2^Q exists (the more conventional definition being when no attack better than the generic exhaustive search is known, whose complexity usually is $2^Q = 2^{k/2}$, but not always).

5.3.3.1 Concrete parameters for secure constructions

We recall that our definition of "Quantum security of Q bits" is "no attack with less that 2^Q operations exists".

Table 5.1: Examples of provided the post-quantum security provided by the Even-Mansour construction with usual parameters when using modular addition + or xor addition $\oplus$.

Construction (p / w)	State size	Key size	Provided Quantum security
Even-Mansour$\oplus$ (1 / n)	$n = 128$	$k = 128$	$Q = 8$ bits
Even-Mansour+ (1 / n)	$n = 128$	$k = 128$	$Q = 20$ bits
Even-Mansour$\oplus$ (1 / n)	$n = 256$	$k = 256$	$Q = 9$ bits
Even-Mansour+ (1 / n)	$n = 256$	$k = 256$	$Q = 28.5$ bits
Even-Mansour+ (16 / 16)	$n = 256$	$k = 256$	$Q \leq 24$ bits

5.3.3.2 Discussion

While it seems clear that changing the operation from xor to modular addition increases the security, we showed in Tables 5.2 and 5.3 that applying this modification, in some common primitives, is not enough to provide an acceptable security level. Our estimations indicate that, in order to repair (with modular additions) most of the systems affected by attacks using Simon's algorithm, one has to make their internal state several orders of magnitude larger,

Table 5.2: Summary of constructions and parameters in order to resist the corresponding quantum attacks when using modular additions instead of xor addition.

Construction (p / w)	State size	Key size	Quantum security
Even-Mansour (1 / n)	$n = 5200 = 2^{12.34}$	$k = n$	$Q = 128$ bits
Even-Mansour (1 / n)	$n = 2048 = 2^{11}$	$k = n$	$Q = 80$ bits
Even-Mansour (p / w)	$n = 2^{12.83} - 17.6(\sqrt{p} - \sqrt{w})$	$k = n$	$Q = 128$ bits
LRW (1 / n)	$n = 5200 = 2^{12.34}$	$k \geq 128$	$Q = 128$ bits
LRW (p / w)	$n \geq 2^{12.83} - 17.6(\sqrt{p} - \sqrt{w})$	$k \geq 128$	$Q = 128$ bits
Op. modes (CBC-MAC...) (1 / n)	$n = 5200 = 2^{12.34}$	$k \geq 128$	$Q = 128$ bits
Op. modes (CBC-MAC...) (p / w)	$n \geq 2^{12.83} - 17.6(\sqrt{p} - \sqrt{w})$	$k \geq 128$	$Q = 128$ bits

Table 5.3: Summary of constructions and parameters in order to resist the best corresponding slide quantum attacks.

Construction (p / w)	State size	Key size	Quantum security
Key-alternating Cipher (1 / n)	$n = 5200 = 2^{12.34}$	$k = n$	$Q = 128$ bits
Key-alternating Cipher (p / w)	$n \geq 2^{12.83} - 17.6(\sqrt{p} - \sqrt{w})$	$k = n$	$Q = 128$ bits
2k-DES$\oplus$ (1 / n)	$n = 2^{128}$	$k = n$	$Q = 128$ bits
2k-DES+ (1 / n)	$n = 5200 = 2^{12.34}$	$k = n$	$Q = 128$ bits
2k-DES+ (p / w)	$n \geq 2^{12.83} - 17.6(\sqrt{p} - \sqrt{w})$	$k = 2n$	$Q = 128$ bits
2k-DESX$\oplus$ (1 / n)	$n = 2^{128}$	$k = n$	$Q = 128$ bits
2k-DESX+ (1 / n)	$n = 5200 = 2^{12.34}$	$k = n$	$Q = 128$ bits
2k-DESX+ (p / w)	$n \geq 2^{12.83} - 17.6(\sqrt{p} - \sqrt{w})$	$k = 2n$	$Q = 128$ bits

implying a great disadvantage against other symmetric primitives (in terms of efficiency, cost, size). One might infer that other modifications beyond the substitution of xors by modular additions should be considered, in order to make the affected ciphers safe in a quantum world.

5.4 Perspectives

The main challenge of my ERC project QUASYModo is to redesign symmetric cryptography for the post-quantum world. The final objective is to construct and recommend symmetric primitives secure in the post-quantum world, as well as the tools needed to properly evaluate them. I will continue to work on this toolbox, and when it is ready, I will use it to: 1) analyze existing cryptosystems/primitives, and 2) design new ones for which we will gain confidence in the post-quantum world.

Some other short-term aims are: improvements on linear cryptanalysis using QFT seem possible, try to find better algorithms for solving the same problem as Kupderberg when having several parallel modular additions, providing a quantized version of improved slide attacks, and study the effect of a smaller than the key state for quantum adversaries (starting for instance quantizing sweet-32). I also plan to start working on the design of a block cipher with an internal state size of 256 bits.

Chapter 6

Dedicated cryptanalysis

In this chapter we provide an overview of the dedicated cryptanalysis that I have published in the last eight years.

Importance of dedicated Cryptanalysis

The new emerging needs for symmetric primitives that we described in Section 1.2.4.2 have caused the apparation of many innovative constructions.

For instance, the strong demand for lightweight primitives, which are often risky and have a low security margin (see [BLP+08]), both from the community and the industry, has been met with a huge amount of promising new primitives, with diverse implementation features. Some examples are PRESENT [BKL+07b], CLEFIA [SSA+07], KATAN/KTANTAN [CDK09a], LBlock [WZ11], TWINE [SMMK12], LED [GPPR11], PRINCE [BCG+12], KLEIN [GNL11], Trivium [CP08] and Grain [HJM07].

The need for clearly recommended lightweight ciphers requires that the large number of these potential candidates be narrowed down. In this context, the need for a significant cryptanalysis effort is obvious. This has been proved by the large number of security analyses of the earlier primitives (to cite a few: [LAAZ11, BKLT11, MRTV12, NWW13, CS09, BR10, TSLL11]).

Normally, designers should have already analyzed their proposed cipher with respect to known attacks.[1] So we need to find new, dedicated attacks, in order to adapt to the new constructions. To illutrate this need, a good example is PRINTcipher: despite its similarity with PRESENT, a secure cipher, it is now considered a broken proposal, thanks to new dedicated attacks. Some of my selected papers, at the end of the manuscript, describe some dedicated attacks. Here we list the obtained results.

Regarding hash functions:

1. SHAvite-3-256 [MNPP11] (best known) and 512 [GLM+10] (full round compression function)
2. Luffa [KNPRS10] (best known)
3. ECHO [JNPS11] (best known, 7/8 rounds of the compression function)
4. Grøstl [JNPP12b] (best known, 9/10 rounds for the permutation)
5. Keccak [NPRM11] (first practical cryptanalysis results up to 3/24 rounds).

Regarding ciphers:

[1] Not always. Often, an attack doesn't appear applicable in a straightforward way, because of lack of generalized techniques.

1. Klein [ANPS11, LN15b] (break of the full cipher)
2. Sprout [LN15a] (break of the full cipher)
3. Armadillo2 [ABNP+11b, NPP12b] (break of the full cipher)
4. PRINCE [CFG+15] (best known attack)
5. PICARO [CLN15] (related-key attack on the full cipher)

Chapter 7

Conclusion and Perspectives

Highlights and follow-up work. As a highlight of the research I have done during these last 8 years, I would point out my two personal preferred main contributions:

- Our generalization of several families of cryptanalysis and the algorithmic improvement for the list-merging problem: I believe that this is a fundamental task that should be continued. Indeed, my next immediate steps will be to expand my generalization of (truncated) differential attacks, and to add non-trivial improvements, such as neutral bits. I also plan to provide a simple, compact description of the scenarios where merging or dissection algorithms can be applied. A fundamental part of this generalization work on cryptanalysis families will be the design and development of several implementation tools: a cryptanalisis toolbox. This is very important, as well as making these tools public. I believe that the symmetric community is migrating to an open access of such tools, which is very good news, and I plan on contributing.

- Our recent work on the quantum cryptanalysis of symmetric primitives, and in particular our analysis of the modular addition tweak, show how little we know about quantum attacks for symmetric cryptography. It seems fair to say that the community has spent little effort so far on that subject, but this is changing now, fortunately: more and more researchers start working on it. I firmly believe – and our work demonstrated it – that symmetric cryptanalysts must be deeply involved in this effort, along the specialists in quantum computing and algorithms, in order to obtain results with maximal scope and impact.

Main perspectives. While I plan to continue working on cryptanalysis (classical and quantum), I also think it is important to learn more about key-schedules. This is an important area, that has direct applications when searching ways to increase the key lengths to resist to quantum attacks. As previously pointed out, I also plan to study the effect of having an internal state smaller than the key, for quantum adversaries (starting for instance with quantizing the Sweet-32 attack [BL16]).

After studying the effects of the state size and key-schedules with respect to quantum attacks, I plan to build a block cipher of size 256 bits, with the aim of resisting any possible upcoming attacks. As Joan Daemen pointed out during the Early Symmetric Crypto seminar in Luxembourg in 2017, it might be better to migrate from block ciphers to permutation-based ciphers, which are more similar to stream ciphers or sponge constructions. Therefore, studying the effect of quantum adversaries in these constructions is also important.

Alternative additional directions. I want to keep on working on the design of symmetric primitives for homomorphic encryption and easy-to-mask primitives, both with larger keys, in order to make them secure in a post-quantum world.

An interesting emerging field in symmetric cryptography is the combination of slide attacks with algorithmic cryptanalysis. Many knowledge and security improvements can be obtained from working in this field, which seems particularly interesting.

Security analysis and dedicated cryptanalysis are still needed, particularly for lightweight primitives and for the *Internet of things*. They are moving and expanding quickly, and cryptanalists needs to keep up.

Bibliography

[ABNP+11a] M. A. Abdelraheem, C. Blondeau, M. Naya-Plasencia, M. Videau, and E. Zenner. Cryptanalysis of ARMADILLO2. In *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 308–326. Springer, 2011. (Cited on pages v, xi, 27 and 84.)

[ABNP+11b] M. A. Abdelraheem, C. Blondeau, M. Naya-Plasencia, M. Videau, and E. Zenner. Cryptanalysis of ARMADILLO2. In *ASIACRYPT*, volume 7073 of *LNCS*, pages 308–326. Springer, 2011. (Cited on pages vi, ix, 21 and 56.)

[AHMN13] J. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia. Quark: A lightweight hash. *J. Cryptology*, 26(2):313–339, 2013. (Cited on pages v, xi, 9, 12 and 84.)

[AHMNP10] J. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia. Quark: A lightweight hash. In *Cryptographic Hardware and Embedded Systems - CHES 2010*, volume 6225 of *LNCS*, pages 1–15. Springer, 2010. (Cited on pages v and xi.)

[AIK+00] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita. Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis. In *Selected Areas in Cryptography - SAC 2000*, volume 2012 of *LNCS*, pages 39–56. Springer, 2000. (Cited on page 37.)

[AL13] H. A. Alkhzaimi and M. M. Lauridsen. Cryptanalysis of the SIMON Family of Block Ciphers. Cryptology ePrint Archive, Report 2013/543, 2013. (Cited on page 31.)

[ALLW13] F. Abed, E. List, S. Lucks, and J. Wenzel. Differential and linear cryptanalysis of reduced-round SIMON. Cryptology ePrint Archive, Report 2013/526, 2013. (Cited on page 31.)

[ALWL15] F. Abed, E. List, J. Wenzel, and S. Lucks. Differential cryptanalysis of round-reduced simon and speck. In *FSE 2014*, volume 8540 of *LNCS*, pages 525–545. Springer, 2015. (Cited on page 31.)

[AM15] F. Armknecht and V. Mikhalev. On lightweight stream ciphers with shorter internal states. In *FSE 2015*, volume 9054 of *LNCS*, pages 451–470. Springer, 2015. (Cited on page 15.)

[ANPS11] J-Ph Aumasson, M. Naya-Plasencia, and M. Saarinen. Practical attack on 8 rounds of the lightweight block cipher KLEIN. In *Indocrypt 2011*, volume 7107 of *LNCS*, pages 134–145. Springer, 2011. (Cited on pages v, ix, xi and 56.)

[AR17] G. Alagic and A. Russell. Quantum-Secure Symmetric-Key Cryptography Based on Hidden Shifts. In *Eurocrypt 2017*, 2017. To appear. (Cited on pages xiv, 45, 52 and 53.)

[ARS+15] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for MPC and FHE. In *EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 430–454. Springer, 2015. (Cited on pages v, 7, 9 and 17.)

[AS08] K. Aoki and Y. Sasaki. Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In *Selected Areas in Cryptography - SAC 2008*, volume 5381 of *LNCS*, pages 103–119. Springer, 2008. (Cited on pages 36 and 37.)

[AS09] K. Aoki and Y. Sasaki. Meet-in-the-Middle Preimage Attacks Against Reduced SHA-0 and SHA-1. In *CRYPTO 2009*, volume 5677 of *LNCS*, pages 70–89. Springer, 2009. (Cited on page 37.)

[BBD09] D. J. Bernstein, J. Buchmann, and E. Dahmen. Post-Quantum Cryptography. pages 1–14, 2009. Introductory chapter to book "Post-quantum cryptography". (Cited on pages vii and 46.)

[BBL13] C. Blondeau, A. Bogdanov, and G. Leander. Bounds in shallows and in miseries. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *LNCS*, pages 204–221. Springer, 2013. (Cited on page 41.)

[BBS99] E. Biham, A. Biryukov, and A. Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In *EUROCRYPT 1999*, volume 1592 of *LNCS*, pages 12–23. Springer, 1999. (Cited on page 30.)

[BCG+12] J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, P. Rombouts, S. S. Thomsen, and T. Yalçin. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications. In *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 208–225. Springer, 2012. (Cited on pages viii and 55.)

[BDD+15] A. Bar-On, I. Dinur, O. Dunkelman, V. Lallemand, N. Keller, and B. Tsaban. Cryptanalysis of SP networks with partial non-linear layers. In *EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 315–342. Springer, 2015. (Cited on pages vi, 9 and 19.)

[BDF+11a] D. Boneh, Ö. Dagdelen, M. Fischlin, A. Lehmann, C. Schaffner, and M. Zhandry. Random Oracles in a Quantum World. In *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 41–69. Springer Berlin Heidelberg, 2011. (Cited on page 47.)

[BDF11b] C. Bouillaguet, P. Derbez, and P. Fouque. Automatic Search of Attacks on Round-Reduced AES and Applications. In *CRYPTO 2011*, volume 6841 of *LNCS*, pages 169–187. Springer, 2011. (Cited on page 38.)

[BDP15] A. Biryukov, P. Derbez, and L. Perrin. Differential Analysis and Meet-in-the-Middle Attack Against Round-Reduced TWINE. In *Fast Software Encryption - FSE 2015*, volume 9054 of *LNCS*, pages 3–27. Springer, 2015. (Cited on page 31.)

[BDPA08] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. On the indifferentiability of the sponge construction. In *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 181–197. Springer, 2008. (Cited on page 10.)

[BDPA13] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Keccak. In *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 313–314. Springer, 2013. (Cited on page 6.)

[Ber11] D. J. Bernstein. Post-quantum cryptography. In *Encyclopedia of Cryptography and Security, 2nd Ed.*, pages 949–950. Springer, 2011. (Cited on page 45.)

[BHK+11] G. Brassard, P. Høyer, K. Kalach, M. Kaplan, S. Laplante, and L. Salvail. Merkle puzzles in a quantum world. In *Advances in Cryptology–CRYPTO 2011*, volume 6841 of *LNCS*, pages 391–410. Springer, 2011. (Cited on page 47.)

[BHNS10] B.B. Brumley, R. M. Hakala, K. Nyberg, and S. Sovio. Consecutive S-box Lookups: A Timing Attack on SNOW 3G. In *Information and Communications Security - ICICS 2010*, volume 6476 of *LNCS*. Springer, 2010. (Cited on page 36.)

[BKL+07a] A. Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *LNCS*, pages 450–466. Springer, 2007. (Cited on page 17.)

[BKL+07b] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2007*, LNCS 4727, pages 450–466. Springer Verlag, 2007. (Cited on pages viii and 55.)

[BKL+11] A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varici, and I. Verbauwhede. SPONGENT: A lightweight hash function. In *Cryptographic Hardware and Embedded Systems - CHES 2011*, volume 6917 of *LNCS*, pages 312–325. Springer, 2011. (Cited on page 13.)

[BKLT11] J. Borghoff, L. R. Knudsen, G. Leander, and S. S. Thomsen. Cryptanalysis of PRESENT-Like Ciphers with Secret S-Boxes. In *FSE*, volume 6733 of *LNCS*, pages 270–289. Springer, 2011. (Cited on pages viii and 55.)

[BKR00] M. Bellare, J. Kilian, and P. Rogaway. The Security of the Cipher Block Chaining Message Authentication Code. *J. Comput. Syst. Sci.*, 61(3):362–399, 2000. (Cited on page 49.)

[BKR11] A. Bogdanov, D. Khovratovich, and C. Rechberger. Biclique Cryptanalysis of the Full AES. In *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 344–371. Springer, 2011. (Cited on page 38.)

[BL16] K. Bhargavan and G. Leurent. On the practical (in-)security of 64-bit block ciphers: Collision attacks on HTTP over TLS and openvpn. In E. R. Weippl,

S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 456–467. ACM, 2016. (Cited on page 57.)

[BLNS16] C. Boura, V. Lallemand, M. Naya-Plasencia, and V. Suder. Making the impossible possible. *J. Cryptology*, 2016. to appear. (Cited on pages v, vii, xi, 29, 31, 34 and 37.)

[Blo13] C. Blondeau. Improbable Differential from Impossible Differential: On the Validity of the Model. In *INDOCRYPT*, volume 8250 of *LNCS*, pages 149–160. Springer, 2013. (Cited on page 31.)

[Blo15] C. Blondeau. Impossible differential attack on 13-round Camellia-192. *Inf. Process. Lett.*, 115(9):660–666, 2015. (Cited on page 31.)

[BLP+08] A. Bogdanov, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, and Y. Seurin. Hash Functions and RFID Tags: Mind the Gap. In *CHES 2008*, volume 5154 of *LNCS*, pages 283–299. Springer, 2008. (Cited on pages v, viii, 7, 9 and 55.)

[BM15] C. Blondeau and M. Minier. Analysis of Impossible, Integral and Zero-Correlation Attacks on Type-II Generalized Feistel Networks Using the Matrix Method. In *FSE 2015*, volume 9054 of *LNCS*, pages 92–113. Springer, 2015. (Cited on page 37.)

[BMNPS14] C. Boura, M. Minier, M. Naya-Plasencia, and V. Suder. Improved Impossible Differential Attacks against Round-Reduced LBlock. Cryptology ePrint Archive, Report 2014/279, 2014. (Cited on page 36.)

[BNP17] X. Bonnetain and M. Naya-Plasencia. On concrete quantum security of symmetric primitives with modular additions. 2017. Submitted. (Cited on pages viii, xii, 45, 48, 50, 52 and 53.)

[BNS14] C. Boura, M. Naya-Plasencia, and V. Suder. Scrutinizing and improving impossible differential attacks: Applications to CLEFIA, Camellia, LBlock and Simon. In *ASIACRYPT 2014*, volume 8873 of *LNCS*, pages 179–199. Springer, 2014. (Cited on pages vii, xi, 29, 30, 31, 32, 33, 34, 36, 37 and 84.)

[BR00] John Black and Phillip Rogaway. CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions. In Mihir Bellare, editor, *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, volume 1880 of *LNCS*, pages 197–215. Springer, 2000. (Cited on page 49.)

[BR10] A. Bogdanov and C. Rechberger. A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN. In *Selected Areas in Cryptography - SAC 2010*, volume 6544 of *LNCS*, pages 229–240. Springer, 2010. (Cited on pages viii, 37 and 55.)

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. In *CRYPTO '90*, volume 537 of *LNCS*, pages 2–21. Springer, 1991. (Cited on page 41.)

[BW99] A. Biryukov and D. Wagner. Slide attacks. In *FSE 1999*, volume 1636 of *LNCS*, pages 245–259. Springer, 1999. (Cited on page 50.)

[BZ13a] D. Boneh and M. Zhandry. Quantum-Secure Message Authentication Codes. In *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 592–608. Springer, 2013. (Cited on page 50.)

[BZ13b] D. Boneh and M. Zhandry. Secure signatures and chosen ciphertext security in a quantum computing world. In *CRYPTO 2013*, volume 8043 of *LNCS*, pages 361–379. Springer, 2013. (Cited on page 47.)

[CCF+16] A. Canteaut, S. Carpov, C. Fontaine, T. Lepoint, M. Naya-Plasencia, P. Paillier, and R. Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. In *Fast Software Encryption - FSE 2016*, volume 9783 of *LNCS*, pages 313–333. Springer, 2016. (Cited on pages v, vi, xi, 7, 9, 14 and 84.)

[CDK09a] C. De Cannière, O. Dunkelman, and M. Knezevic. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In *CHES 2009*, LNCS 5747, pages 272–288. Springer Verlag, 2009. (Cited on pages viii and 55.)

[CDK09b] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - a family of small and efficient hardware-oriented block ciphers. In Christophe Clavier and Kris Gaj, editors, *CHES*, volume 5747 of *LNCS*, pages 272–288. Springer, 2009. (Cited on pages 15 and 17.)

[CE85] D. Chaum and J. Evertse. Crytanalysis of DES with a Reduced Number of Rounds: Sequences of Linear Factors in Block Ciphers. In *CRYPTO '85*, volume 218 of *LNCS*, pages 192–211. Springer, 1985. (Cited on page 37.)

[CFG+15] A. Canteaut, T. Fuhr, H. Gilbert, M. Naya-Plasencia, and J. Reinhard. Multiple differential cryptanalysis of round-reduced PRINCE. In *Fast Software Encryption - FSE 2014*, volume 8540 of *LNCS*, pages 591–610. Springer, 2015. (Cited on pages v, ix, xi, 40 and 56.)

[CLN15] A. Canteaut, V. Lallemand, and M. Naya-Plasencia. Related-key attack on full-round PICARO. In *Selected Areas in Cryptography- SAC 2015*, volume 9566 of *LNCS*, pages 86–101. Springer, 2015. (Cited on pages v, ix, xi and 56.)

[CN12] A. Canteaut and M. Naya-Plasencia. Correlation attacks on combination generators. *Cryptography and Communications*, 4(3-4):147–171, 2012. (Cited on pages vii, xi, 29, 30 and 84.)

[CNPV13] A. Canteaut, M. Naya-Plasencia, and B. Vayssière. Sieve-in-the-Middle: Improved MITM techniques. In *CRYPTO 2013 (I)*, volume 8042 of *LNCS*, pages 222–240. Springer, 2013. (Cited on pages vi, vii, xi, 21, 26, 27, 29, 30, 35, 36, 40 and 84.)

[CP08] C. De Cannière and B. Preneel. Trivium. In Matthew J. B. Robshaw and Olivier Billet, editors, *The eSTREAM Finalists*, volume 4986 of *LNCS*, pages 244–266. Springer, 2008. (Cited on pages viii, 17 and 55.)

[CS09] Baudoin Collard and François-Xavier Standaert. A Statistical Saturation Attack against the Block Cipher PRESENT. In *Topics in Cryptology - CT-RSA 2009*, LNCS 5473, pages 195–210. Springer Verlag, 2009. (Cited on pages viii and 55.)

[DDKS12] I. Dinur, O. Dunkelman, N. Keller, and A. Shamir. Efficient Dissection of Composite Problems, with Applications to Cryptanalysis, Knapsacks, and Combinatorial Search Problems. In *CRYPTO 2012*, volume 7417 of *LNCS*, pages 719–740. Springer, 2012. (Cited on pages 25, 26 and 28.)

[Der16] P. Derbez. Note on Impossible Differential Attacks. In *Fast Software Encryption - FSE 2016*, volume 9783 of *LNCS*, pages 416–427. Springer, 2016. (Cited on pages 31 and 35.)

[DF13] P. Derbez and P.-A. Fouque. Exhausting Demirci-Selçuk Meet-in-the-Middle Attacks Against Reduced-Round AES. In *Fast Software Encryption - FSE 2013*, volume 8424 of *LNCS*, pages 541–560. Springer, 2013. (Cited on page 37.)

[DF16] P. Derbez and P.-A. Fouque. Automatic search of meet-in-the-middle and impossible differential attacks. In *CRYPTO 2016*, volume 9815 of *LNCS*, pages 157–184. Springer, 2016. (Cited on page 29.)

[DFJ13] P. Derbez, P.-A. Fouque, and J. Jean. Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting. In *EUROCRYPT'13*, volume 7881 of *LNCS*, pages 371–387. Springer, 2013. (Cited on page 37.)

[DFNS14] I. Damgård, J. Funder, J. B. Nielsen, and L. Salvail. Superposition Attacks on Cryptographic Protocols. In *Information Theoretic Security - 7th International Conference, ICITS 2013, Singapore, November 28-30, 2013, Proceedings*, volume 8317 of *LNCS*, pages 142–161. Springer, 2014. (Cited on page 47.)

[DLMW15] I. Dinur, Y. Liu, W. Meier, and Q. Wang. Optimized interpolation attacks on lowmc. In *ASIACRYPT 2015*, volume 9453 of *LNCS*, pages 535–560. Springer, 2015. (Cited on page 16.)

[DR00] J. Daemen and V. Rijmen. The block cipher Rijndael. In *CARDIS '98*, volume 1820 of *LNCS*, pages 277–284. Springer, 2000. (Cited on page 6.)

[DR02] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002. (Cited on page 2.)

[DR07] J. Daemen and V. Rijmen. Probability distributions of correlation and differentials in block ciphers. *J. Mathematical Cryptology*, 1(3):221–242, 2007. (Cited on page 41.)

[DS09] I. Dinur and A. Shamir. Cube attacks on tweakable black box polynomials. In *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 278–299. Springer, 2009. (Cited on page 30.)

[DSP07] O. Dunkelman, G. Sekar, and B. Preneel. Improved Meet-in-the-Middle Attacks on Reduced-Round DES. In *INDOCRYPT 2007*, volume 4859 of *LNCS*, pages 86–100. Springer, 2007. (Cited on page 37.)

[Dwo05] M. Dworkin. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. NIST Special Publication 800-38B, National Institute for Standards and Technology, May 2005. (Cited on page 49.)

[EMST76] W. R. Ehrsam, C. H. Meyer, J. L. Smith, and W. L. Tuchman. Message verification and transmission error detection by block chaining. *US Patent 4074066*, 1976. (Cited on page 2.)

[FIP01] FIPS 197. Announcing the Advanced Encryption Standard (AES). National Institute for Standards and Technology, Gaithersburg, MD, USA, November 2001. (Cited on page 37.)

[FM14] T. Fuhr and B. Minaud. Match Box Meet-in-the-Middle Attack against KATAN. In *fast Software Encryption - FSE 2014*, volume 8540 of *LNCS*, pages 61–81. Springer, 2014. (Cited on page 15.)

[GGNPS13] B. Gérard, V. Grosso, M. Naya-Plasencia, and F.-X. Standaert. Block ciphers that are easier to mask: How far can we go? In *Cryptographic Hardware and Embedded Systems - CHES 2013*, volume 8086 of *LNCS*, pages 383–399. Springer, 2013. (Cited on pages v, vi, xi, 7, 9, 17 and 84.)

[GHS16] T. Gagliardoni, A. Hülsing, and C. Schaffner. Semantic Security and Indistinguishability in the Quantum World. In *CRYPTO 2016*, volume 9816 of *LNCS*, pages 60–89. Springer, 2016. (Cited on page 47.)

[GLM+10] P. Gauravaram, G. Leurent, F. Mendel, M. Naya-Plasencia, T. Peyrin, C. Rechberger, and M. Schläffer. Cryptanalysis of the 10-round hash and full compression function of SHAvite-3-512. In *Africacrypt 2010*, volume 6055 of *LNCS*, pages 419–436. Springer, 2010. (Cited on pages v, viii, xi and 55.)

[GLRW10] J. Guo, S. Ling, C. Rechberger, and H. Wang. Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2. In *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 56–75. Springer, 2010. (Cited on page 37.)

[GNL11] Z. Gong, S. Nikova, and Y. Wei Law. KLEIN: A New Family of Lightweight Block Ciphers. In *RFIDSec 2011*, volume 7055 of *LNCS*, pages 1–18. Springer, 2011. (Cited on pages viii and 55.)

[GPP11] J. Guo, T. Peyrin, and A. Poschmann. The PHOTON family of lightweight hash functions. In *CRYPTO*, volume 6841 of *LNCS*, pages 222–239. Springer, 2011. (Cited on page 13.)

[GPPR11] J. Guo, T. Peyrin, A. Poschmann, and M. Robshaw. The LED Block Cipher. In *Workshop on Cryptographic Hardware and Embedded Systems 2011 - CHES 2011*, volume 6917 of *LNCS*, pages 326–341. Springer, 2011. (Cited on pages viii and 55.)

[Gro96] L. K. Grover. A fast quantum mechanical algorithm for database search. In *ACM Symposium on the Theory of Computing 1996*, pages 212–219. ACM, 1996. (Cited on pages vii and 46.)

[HJM07] M. Hell, T. Johansson, and W. Meier. Grain: a stream cipher for constrained environments. *IJWMC*, 2(1):86–93, 2007. (Cited on pages viii, 17 and 55.)

[IK03] T. Iwata and K. Kurosawa. OMAC: One-Key CBC MAC. In *Fast Software Encryption - FSE 2003*, volume 2887 of *LNCS*, pages 129–153. Springer, 2003. (Cited on page 49.)

[IS12] T. Isobe and K. Shibutani. All Subkeys Recovery Attack on Block Ciphers: Extending Meet-in-the-Middle Approach. In *Selected Areas in Cryptography - SAC 2012*, volume 7707 of *LNCS*, pages 202–221. Springer, 2012. (Cited on page 37.)

[Iso11] T. Isobe. A Single-Key Attack on the Full GOST Block Cipher. In *FSE 2011*, volume 6733 of *LNCS*, pages 290–305. Springer, 2011. (Cited on page 37.)

[JNP13] J. Jean, M. Naya-Plasencia, and T. Peyrin. Multiple limited-birthday distinguishers and applications. In *In Selected Areas in Cryptography- SAC 2013*, volume 8282 of *LNCS*, pages 533–550. Springer, 2013. (Cited on pages v, vii, xi and 29.)

[JNP14] J. Jean, M. Naya-Plasencia, and T. Peyrin. Improved cryptanalysis of AES-like permutations. *J. Cryptology*, 27(4):772–798, 2014. (Cited on pages v, xi and 84.)

[JNPP12a] J. Jean, M. Naya-Plasencia, and T. Peyrin. Improved rebound attack on the finalist Grøstl. In *Fast Software Encryption - FSE 2012*, volume 7549 of *LNCS*, pages 110–126. Springer, 2012. (Cited on pages v, vi, xi, 21 and 27.)

[JNPP12b] Jérémy Jean, María Naya-Plasencia, and Thomas Peyrin. Improved rebound attack on the finalist Grøstl. In *FSE*, LNCS. Springer, 2012. to appear. (Cited on pages viii, 27 and 55.)

[JNPS11] J. Jean, M. Naya-Plasencia, and M. Schlaffer. Improved analysis of ECHO-256. In *Selected Areas in Cryptography- SAC 2011*, volume 7118 of *LNCS*, pages 19–36. Springer, 2011. (Cited on pages v, vi, viii, xi, 21, 27 and 55.)

[Kap14] M. Kaplan. Quantum attacks against iterated block ciphers. *CoRR*, abs/1410.1434, 2014. (Cited on page 47.)

[KDH12] F. Karakoç, H. Demirci, and A. E. Harmanci. Impossible Differential Cryptanalysis of Reduced-Round LBlock. In *WISTP 2012*, volume 7322 of *LNCS*, pages 179–188. Springer, 2012. (Cited on page 36.)

[KHL+04] J. Kim, S. Hong, S. Lee, J. Hwan Song, and H. Yang. Truncated Differential Attacks on 8-Round CRYPTON. In *ICISC 2003*, volume 2971 of *LNCS*, pages 446–456. Springer, 2004. (Cited on page 37.)

[KKP+04] D. Kwon, J. Kim, S. Park, S. H. Sung, Y. Sohn, J. H. Song, Y. Yeom, E. Yoon, S. Lee, J. Lee, S. Chee, D. Han, and J. Hong. New Block Cipher: ARIA. In *ICISC 2003*, volume 2971 of *LNCS*, pages 432–445. Springer, 2004. (Cited on page 37.)

[KLLN16a] M. Kaplan, G. Leurent, A. Leverrier, and M. Naya-Plasencia. Breaking symmetric cryptosystems using quantum period finding. In *CRYPTO 2016*, volume 9815 of *LNCS*, pages 207–237. Springer, 2016. (Cited on pages vii, xi, xii, 45, 48, 49 and 84.)

[KLLN16b] M. Kaplan, G. Leurent, A. Leverrier, and M. Naya-Plasencia. Quantum differential and linear cryptanalysis. *IACR Trans. Symmetric Cryptol.*, 2016(1):71–94, 2016. (Cited on pages v, vii, xi, xii, 40, 45, 47 and 84.)

[KM10] H. Kuwakado and M. Morii. Quantum distinguisher between the 3-round Feistel cipher and the random permutation. In *Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on*, pages 2682–2685, June 2010. (Cited on page 47.)

[KM12] H. Kuwakado and M. Morii. Security on the quantum-type Even-Mansour cipher. In *2012 International Symposium on Information Theory and its Applications (ISITA 2012)* , pages 312–316, Oct 2012. (Cited on page 47.)

[KMN11] S. Knellwolf, W. Meier, and M. Naya-Plasencia. Conditional Differential Cryptanalysis of Trivium and KATAN. In *SAC 2011*, volume 7118 of *LNCS*, pages 200–212. Springer, 2011. (Cited on pages xi and 16.)

[KMNP10] S. Knellwolf, W. Meier, and M. Naya-Plasencia. Conditional differential cryptanalysis of NLFSR based cryptosystems. In *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 130–145. Springer, 2010. (Cited on pages vii, xi, 16, 29 and 84.)

[KNPRS10] D. Khovratovich, M. Naya-Plasencia, A. Röck, and M. Schläffer. Cryptanalysis of *Luffa* v2 Components. In *Selected Areas in Cryptography - SAC 2012*, volume 6544 of *LNCS*, pages 388–409. Springer, 2010. (Cited on pages v, viii, xi, 36 and 55.)

[Knu95] L. R. Knudsen. Truncated and higher order differentials. In *Fast Software Encryption - FSE 1994*, volume 1008 of *LNCS*, pages 196–211. Springer, 1995. (Cited on page 42.)

[Knu98] L. R. Knudsen. DEAL – A 128-bit cipher. Technical Report, Department of Informatics, University of Bergen, Norway, 1998. (Cited on page 30.)

[KR11] T. Krovetz and P. Rogaway. The software performance of authenticated-encryption modes. In *2011*, volume 6733 of *LNCS*, pages 306–327. Springer, 2011. (Cited on page 2.)

[KRS12] D. Khovratovich, C. Rechberger, and A. Savelieva. Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family. In *FSE 2012*, volume 7549 of *LNCS*, pages 244–263. Springer, 2012. (Cited on page 38.)

[Kup05] G. Kuperberg. A Subexponential-Time Quantum Algorithm for the Dihedral Hidden Subgroup Problem. *SIAM J. Comput.*, 35(1):170–188, 2005. (Cited on page 52.)

[Kup13] Greg Kuperberg. Another Subexponential-time Quantum Algorithm for the Dihedral Hidden Subgroup Problem. In *8th Conference on the Theory of Quantum Computation*, volume 22 of *LIPIcs*, pages 20–34. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013. (Cited on page 52.)

[KW02] L. R. Knudsen and D. Wagner. Integral cryptanalysis. In *Fast Software Encryption - FSE 2002*, volume 2365 of *LNCS*, pages 112–127. Springer, 2002. (Cited on page 17.)

[LAAZ11] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi, and Erik Zenner. A Cryptanalysis of PRINTcipher: The Invariant Subspace Attack. In *Advances in Cryptology - CRYPTO 2011*, volume 6841 of *LNCS*, pages 206–221. Springer, 2011. (Cited on pages viii and 55.)

[LDKK08] J. Lu, O. Dunkelman, N. Keller, and J. Kim. New Impossible Differential Attacks on AES. In *INDOCRYPT'08*, volume 5365 of *LNCS*, pages 279–293. Springer, 2008. (Cited on page 31.)

[LGL+11] Z. Liu, D. Gu, Y. Liu, J. Li, and W. Lei. Linear cryptanalysis of ARIA block cipher. In *Information and Communications Security*, volume 7043 of *LNCS*, pages 242–254. Springer, 2011. (Cited on page 37.)

[Lim99] C. H. Lim. A Revised Version of Crypton - Crypton V1.0. In *Fast Software Encryption - FSE'99*, volume 1636 of *LNCS*, pages 31–45. Springer, 1999. (Cited on page 37.)

[LJF16] X. Li, C.-H. Jin, and F.-W. Fu. Improved Results of Impossible Differential Cryptanalysis on Reduced FOX. *Comput. J.*, 59(4):541–548, 2016. (Cited on page 31.)

[LJWD15] L. Li, K. Jia, X. Wang, and X. Dong. Meet-in-the-Middle Technique for Truncated Differential and Its Applications to CLEFIA and Camellia. In *Fast Software Encryption - FSE 2015*, volume 9054 of *LNCS*, pages 48–70. Springer, 2015. (Cited on page 37.)

[LKKD08] J. Lu, J. Kim, N. Keller, and O. Dunkelman. Improving the Efficiency of Impossible Differential Cryptanalysis of Reduced Camellia and MISTY1. In *CT-RSA 2008*, volume 4964 of *LNCS*, pages 370–386. Springer, 2008. (Cited on page 31.)

[LLG+12] Y. Liu, L. Li, D. Gu, X. Wang, Z. Liu, J. Chen, and W. Li. New Observations on Impossible Differential Cryptanalysis of Reduced-Round Camellia. In *Fast*

Software Encryption - FSE'12, volume 7549 of *LNCS*, pages 90–109. Springer, 2012. (Cited on pages 31, 36 and 37.)

[LN15a] V. Lallemand and M. Naya-Plasencia. Cryptanalysis of full Sprout. In *CRYPTO 2015*, volume 9215 of *LNCS*, pages 663–682. Springer, 2015. (Cited on pages v, vi, ix, xi, 15, 21, 27, 56 and 84.)

[LN15b] V. Lallemand and M. Naya-Plasencia. Cryptanalysis of KLEIN. In *Fast Software Encryption - FSE 2014*, volume 8540 of *LNCS*, pages 451–470. Springer, 2015. (Cited on pages v, vi, ix, xi, 21, 27 and 56.)

[LRW00] H. Lipmaa, P. Rogaway, and D. Wagner. CTR-mode encryption. *Comments to NIST concerning AES modes of operation*, 2000. (Cited on page 2.)

[LS08] S. Li and C. Song. Improved Impossible Differential Cryptanalysis of ARIA. In *ISA 2008*, pages 129–132, 2008. (Cited on page 37.)

[LSZL08] R. Li, B. Sun, P. Zhang, and C. Li. New Impossible Differential Cryptanalysis of ARIA. Cryptology ePrint Archive, Report 2008/227, 2008. (Cited on page 37.)

[LWW+10] Lars Lydersen, Carlos Wiechers, Christoffer Wittmann, Dominique Elser, Johannes Skaar, and Vadim Makarov. Hacking commercial quantum cryptography systems by tailored bright illumination. *Nature photonics*, 4(10):686–689, 2010. (Cited on page 47.)

[Mal14] H. Mala. Private communication, 2014. (Cited on page 37.)

[MB07] A. Maximov and A. Biryukov. Two Trivial Attacks on Trivium. In *Selected Areas in Cryptology - SAC 2007*, volume 4876 of *LNCS*, pages 36–55. Springer, 2007. (Cited on page 14.)

[MDRM10] H. Mala, M. Dakhilalian, V. Rijmen, and M. Modarres-Hashemi. Improved Impossible Differential Cryptanalysis of 7-Round AES-128. In *INDOCRYPT'10*, volume 6498 of *LNCS*, pages 282–291. Springer, 2010. (Cited on pages 31 and 37.)

[MDS11] H. Mala, M. Dakhilalian, and M. Shakiba. Impossible Differential Attacks on 13-Round CLEFIA-128. *J. Comput. Sci. Technol.*, 26(4):744–750, 2011. (Cited on page 36.)

[Min13] M. Minier. Private communication, May 2013. (Cited on page 31.)

[Min16] M. Minier. Improving impossible-differential attacks against Rijndael-160 and Rijndael-224. *Designs, Codes and Cryptography*, pages 1–13, 2016. (Cited on page 31.)

[MJSC16] P. Méaux, A. Journault, F.-X. Standaert, and C. Carlet. Towards stream ciphers for efficient FHE with low-noise ciphertexts. In *EUROCRYPT 2016*, volume 9665 of *LNCS*, pages 311–343. Springer, 2016. (Cited on pages 7 and 17.)

[MNP12] M. Minier and M. Naya-Plasencia. A Related Key Impossible Differential Attack Against 22 Rounds of the Lightweight Block Cipher LBlock. *Inf. Process. Lett.*, 112(16):624–629, 2012. (Cited on page 31.)

[MNPP11] M. Minier, M. Naya-Plasencia, and T. Peyrin. Analysis of reduced-SHAvite-3-256 v2. In *Fast Software Encryption - FSE 2011*, volume 6733 of *LNCS*, pages 68–87. Springer, 2011. (Cited on pages v, viii, xi and 55.)

[MRST09] F. Mendel, C. Rechberger, M. Schläffer, and S. S. Thomsen. The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In *FSE 2009*, volume 5665 of *LNCS*, pages 260–276. Springer, 2009. (Cited on page 30.)

[MRST65] F. Mendel, C. Rechberger, M. Schläffer, and S. S. Thomsen. The Rebound Attack: Cryptanalysis of Reduced WHIRLPOOL and Grøstl. In *Fast Software Encryption - FSE 2009*, volume 1008 of *LNCS*. Springer, 5665. (Cited on page 27.)

[MRTV12] Florian Mendel, Vincent Rijmen, Deniz Toz, and Kerem Varici. Differential analysis of the LED block cipher. Cryptology ePrint Archive, Report 2012/544, 2012. `http://eprint.iacr.org/`. (Cited on pages viii and 55.)

[MSD10] H. Mala, M. Shakiba, and M. Dakhilalian. New impossible differential attacks on reduced-round Crypton. *Computer Standards & Interfaces*, 32(4):222–227, 2010. (Cited on page 37.)

[MSDB09] H. Mala, M. Shakiba, M. Dakhilalian, and G. Bagherikaram. New Results on Impossible Differential Cryptanalysis of Reduced-Round Camellia-128. In *Selected Areas in Cryptography-SAC 2009*, volume 5867 of *LNCS*, pages 281–294. Springer, 2009. (Cited on page 31.)

[NBoS89] NBS National Bureau of Standards. GOST 28147-89. In *Federal Information Processing Standard- Cryptographic Protection - Cryptographic Algorithm*, 1989. (Cited on pages 52 and 53.)

[New17] Nature News. Physicists propose football-pitch-sized quantum computer. 2017. (Cited on page 45.)

[NISa] NIST. SHA-1. *FIPS PUB 180-1*. (Cited on page 6.)

[NISb] NIST. SHA-2. *FIPS PUB 180-2*. (Cited on page 6.)

[NLV11] Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *ACM CCSW 2011*, pages 113–124. ACM, 2011. (Cited on page 14.)

[NP09] M. Naya-Plasencia. Internal collision attack on Maraca. In *Seminar 09031, Symmetric Cryptography*, Dagstuhl Seminar Proceedings, Germany, January 2009. (Cited on page 22.)

[NP11] M. Naya-Plasencia. How to Improve Rebound Attacks. In *CRYPTO 2011*, volume 6841 of *LNCS*, pages 188–205. Springer, 2011. (Cited on pages vi, xi, 21, 26, 27, 30 and 84.)

[NPP12a] M. Naya-Plasencia and T. Peyrin. Practical cryptanalysis of ARMADILLO2. In *Fast Software Encryption - FSE 2012*, volume 7549 of *LNCS*, pages 146–162. Springer, 2012. (Cited on pages v and xi.)

[NPP12b] M. Naya-Plasencia and T. Peyrin. Practical cryptanalysis of ARMADILLO2. In *Fast Software Encryption - FSE 2012*, volume 7549 of *LNCS*, pages 146–162. Springer, 2012. (Cited on pages ix and 56.)

[NPRM11] M. Naya-Plasencia, A. Röck, and W. Meier. Practical analysis of reduced-round Keccak. In *Indocrypt 2011*, volume 7107 of *LNCS*, pages 236–254. Springer, 2011. (Cited on pages v, vi, viii, xi, 21, 27 and 55.)

[NPTV11] M. Naya-Plasencia, D. Toz, and K. Varici. Rebound attack on JH42. In *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 252–269. Springer, 2011. (Cited on pages v, vi, xi, 21, 27 and 84.)

[NWW13] Ivica Nikolic, Lei Wang, and Shuang Wu. Cryptanalysis of round-reduced LED. In *FSE 2013*, LNCS. Springer, 2013. To appear. (Cited on pages viii and 55.)

[PRC12] G. Piret, T. Roche, and C. Carlet. PICARO - A block cipher allowing efficient higher-order side-channel resistance. In *Applied Cryptography and Network Security - 10th International Conference, ACNS 2012*, volume 7341 of *LNCS*, pages 311–328. Springer, 2012. (Cited on pages v, 7 and 9.)

[Reg04] O. Regev. A Subexponential Time Algorithm for the Dihedral Hidden Subgroup Problem with Polynomial Space. *CoRR*, 2004. `http://arxiv.org/abs/quant-ph/0406151`. (Cited on page 52.)

[Riv91] R. Rivest. MD-5. 1991. (Cited on page 6.)

[Rog06] P. Rogaway. Formalizing human ignorance. In *VIETCRYPT 2006*, volume 4341 of *LNCS*, pages 211–228. Springer, 2006. (Cited on page 5.)

[RRY00] R. L. Rivest, M. J. B. Robshaw, and Y. L. Yin. RC6 as the AES. In *AES Candidate Conference*, pages 337–342, 2000. (Cited on pages 52 and 53.)

[RS15] M. Roetteler and R. Steinwandt. A note on quantum related-key attacks. *Information Processing Letters*, 115(1):40–44, 2015. (Cited on page 47.)

[RSA78] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978. (Cited on pages vii and 46.)

[Sas13] Y. Sasaki. Meet-in-the-Middle Preimage Attacks on AES Hashing Modes and an Application to Whirlpool. *IEICE Transactions*, 96-A(1):121–130, 2013. (Cited on page 37.)

[Sha49] C. Shannon. Communication theory of secrecy systems. *Bell System Technical*, 28:656–715, 1949. (Cited on page 3.)

[Sho97] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997. (Cited on pages vii and 46.)

[Sim97] Daniel R Simon. On the power of quantum computation. *SIAM journal on computing*, 26(5):1474–1483, 1997. (Cited on page 47.)

[Sma14] N. P. Smart. Algorithms, key size and parameters report 2014. Technical report, European Union Agency for Network and Information Security, 2014. https://www.enisa.europa.eu/activities/identity-and-trust/library/deliverables/algorithms-key-size-and-parameters-report-2014. (Cited on page 14.)

[SMMK12] T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi. TWINE : A Lightweight Block Cipher for Multiple Platforms. In *Selected Areas in Cryptography-SAC 2012*, volume 7707 of *LNCS*, pages 339–354. Springer, 2012. (Cited on pages viii and 55.)

[SS16] T. Santoli and C. Schaffner. Using Simon's Algorithm to Attack Symmetric-Key Cryptographic Primitives. *arXiv preprint arXiv:1603.07856*, 2016. (Cited on page 50.)

[SSA+07] T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata. The 128-Bit Blockcipher CLEFIA (Extended Abstract). In *Fast Software Encryption - FSE 2007*, volume 4593 of *LNCS*, pages 181–195. Springer, 2007. (Cited on pages viii, 37 and 55.)

[Tea09] CLEFIA Design Team. Comments on the impossible differential analysis of reduced round CLEFIA presented at Inscrypt 2008, Jan. 8, 2009. (Cited on page 31.)

[Tez10] C. Tezcan. The Improbable Differential Attack: Cryptanalysis of Reduced Round CLEFIA. In *INDOCRYPT*, volume 6498 of *LNCS*, pages 197–209. Springer, 2010. (Cited on page 31.)

[TSLL11] X. Tang, B. Sun, R. Li, and C. Li. Impossible differential cryptanalysis of 13-round CLEFIA-128. *Journal of Systems and Software*, 84(7):1191–1196, 2011. (Cited on pages viii and 55.)

[TTS+08] Y. Tsunoo, E. Tsujihara, M. Shigeri, T. Suzaki, and T. Kawabata. Cryptanalysis of CLEFIA using multiple impossible differentials. In *Information Theory and Its Applications. ISITA 2008*, pages 1–6, 2008. (Cited on page 33.)

[Tuc97] W. L. Tuchman. A brief history of the data encryption standard. *ACM Press/Addison-Wesley Publishing Co. New York, NY, USA.*, 1997. (Cited on page 6.)

[Unr15] D. Unruh. Non-interactive zero-knowledge proofs in the quantum random oracle model. In *Eurocrypt 2015*, volume 9057, pages 755–784. Springer, 2015. Preprint on IACR ePrint 2014/587. (Cited on page 47.)

[WWGY14] Y. Wang, W. Wu, Z. Guo, and X. Yu. Differential cryptanalysis and linear distinguisher of full-round Zorro. In *Applied Cryptography and Network Security - 12th International Conference, ACNS 2014*, volume 8479 of *LNCS*, pages 308–323. Springer, 2014. (Cited on page 18.)

[WY05] X. Wang and H. Yu. How to break MD5 and other hash functions. In *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 19–35. Springer, 2005. (Cited on page 6.)

[WYY05] X. Wang, Y. Lisa Yin, and H. Yu. Finding collisions in the full SHA-1. In *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005. (Cited on page 6.)

[WZ11] W. Wu and L. Zhang. Lblock: A lightweight block cipher. In *Applied Cryptography and Network Security - ACNS 2011*, volume 6715 of *LNCS*, pages 327–344. Springer, 2011. (Cited on pages viii, 37 and 55.)

[WZF07] W. Wu, W. Zhang, and D. Feng. Impossible Differential Cryptanalysis of Reduced-Round ARIA and Camellia. *J. Comput. Sci. Technol.*, 22(3):449–456, 2007. (Cited on pages 31 and 37.)

[WZZ08] W. Wu, L. Zhang, and W. Zhang. Improved Impossible Differential Cryptanalysis of Reduced-Round Camellia. In *Selected Areas in Cryptography-SAC 2008*, volume 5381 of *LNCS*, pages 442–456. Springer, 2008. (Cited on page 31.)

[XQL10] F. Xu, B. Qi, and H.-K. Lo. Experimental demonstration of phase-remapping attack in a practical quantum key distribution system. *New Journal of Physics*, 12(11):113026, 2010. (Cited on page 47.)

[YHSL15] Q. Yang, L. Hu, S. Sun, and L.Song. Related-key Impossible Differential Analysis of Full Khudra. *IACR Cryptology ePrint Archive*, 2015:840, 2015. (Cited on page 31.)

[Yuv79] G. Yuval. How to swindle rabin. *Cryptologia*, 3:187–191, 1979. (Cited on page 5.)

[Yuv97] G. Yuval. Treyfer. 1997. (Cited on pages 52 and 53.)

[ZFQ+08] Y. Zhao, C.-H. F. Fung, B. Qi, C. Chen, and H.-K. Lo. Quantum hacking: Experimental demonstration of time-shift attack against practical quantum-key-distribution systems. *Physical Review A*, 78(4):042333, 2008. (Cited on page 47.)

[ZH08] W. Zhang and J. Han. Impossible Differential Analysis of Reduced Round CLEFIA. In *Inscrypt 2008*, volume 5487 of *LNCS*, pages 181–191. Springer, 2008. (Cited on page 31.)

[Zha12] M. Zhandry. How to Construct Quantum Random Functions. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012*, pages 679–687, 2012. (Cited on page 47.)

[Zha15] M. Zhandry. Secure identity-based encryption in the quantum random oracle model. *International Journal of Quantum Information*, 13(04):1550014, 2015. (Cited on page 47.)

[ZWF07] W. Zhang, W. Wu, and D. Feng. New Results on Impossible Differential Cryptanalysis of Reduced AES. In *ICISC'07*, volume 4817 of *LNCS*, pages 239–250. Springer, 2007. (Cited on page 31.)

Appendix A

Curriculum Vitae

Personal data

Date of birth, Nationality	01/09/1981, Spanish
Children	2
Email	maria.naya_plasencia@inria.fr
Web Page	https://www.rocq.inria.fr/secret/Maria.Naya_Plasencia/

Education

November 2009	PhD in Computer Science, Université Pierre et Marie Curie, Paris, France. Title: "Stream ciphers and hash functions: design and cryptanalysis". PhD advisor: Anne Canteaut (Directrice de Recherche Inria). With highest honours.
2006	Master II: Applied algebra, Versailles University, France.
2005	Double degree: Telecommunications engineer ETSIT, Universidad Politécnica de Madrid (Spain) and Télécom SudParis (France).

Current and previous positions

2014-09/-	CR1 Permanent Researcher at Inria, France.
2012-09/2014-09	CR2 Permanent Researcher at Inria, France.
2012-06	*Admitted to both Inria and CNRS section 7 CR2 competitions.*
2011-09/2012-09	Post-doctoral. Versailles University (L. Goubin, A. Joux), France.
2009-12/2011-09	Post-doctoral, ERCIM fellowship "Alain Bensoussan" and scholarship from the Swiss Scientific Foundation. Fachhochschule Nordwestschweiz (W. Meier), Switzerland.
2006–2009	PhD student Inria Paris-Rocquencourt, France.

Research topics

Symmetric cryptography, with special interest in cryptanalysis of block ciphers, hash functions and lightweight primitives. Dedicated and generic attacks. Design of secure and performant symmetric primitives (as Quark and Shabal). Algorithmic interactions on cryptology.

PhD Supervision

Obtaining my permanent position in late 2012, my first opportunity to supervise a PhD was in 2013.

- Virginie Lallemand, from September 2013 to September 2016. Thesis on cryptanalysis of symmetric primitives. Virginie defended her PhD on October 5 of 2016 and is now doing a Post-doctorate with Gregor Leander in Bochum (Germany).

- Xavier Bonnetain has just started his PhD with me as his advisor on September 2016, with a scholarship from École Polytechnique. Thesis on cryptanalysis of symmetric primitives in a post-quantum world.

Internship supervision

- André Schrottenloher, 5 months. MPRI Master II, Paris, 2017. Study of the security of the symmetric primitives in a post-quantum world.

- Xavier Bonnetain, 6 months. MPRI Master II, Paris, 2016. Study of the security of the symmetric primitives in a post-quantum world.

- Virginie Lallemand, 6 months. Cryptis Master II, Limoges University, 2013. Analysis of the "Klein" blockcipher.

- Chloé Pelle, 3rd year 4-month stage, École Centrale de Lille, 2012 (Codirected with Anne Canteaut, who was in sabbatical stay in Copenhagen). Study of lightweight primitives.

- One-month internship in June 2015 of Victoire Dupont de Dinechin from École de mines and HEC, on cube attacks.

PhD Committees

2016	Virginie Lallemand. Université Paris 6, France.
2015	Joëlle Roué. Université Paris 6, France.
2011	César Estébanez Tascón. Universidad Carlos III de Madrid, Spain.

Dissemination of scientific information

- Invited talk at the Colloquium organised by the pre-GDR Sécurité Informatique (http://gdr-securite.irisa.fr/): Colloque Sécurité informatique CNRS du 8 et 9 décembre, on: "Pourquoi essaie-t-on de casser les fonctions cryptographiques ?" (http://colloque-cybersecu.cnrs.fr/).
- La demi-heure de science at Inria: 05/03/2015 https://www.inria.fr/centre/paris/recherche/la-demi-heure-de-science/2015/maria-naya-plasencia-secret-cryptanalyse-le-fondement-de-la-securite

Member of Recruitement Committee

2015-	Inria Paris CSD Committee.(Comité de suivi doctoral)
2015-	Inria Paris Scientific Hiring Committee (Assignement of PhD, post-doctoral and delegation Inria fundings).
2017	Inria Paris CR2 Jury. Young research position
2017	Limoges University. Assistant Professor.
2016	Paris VIII University. Assistant Professor.
2015	Rennes 1 University. Assistant Professor.

Latest Teaching Activities

2015	Courses on symmetric cryptography for the Thales group (120 hours).
2013 and 2014	**Summer schools** lectures on the design and security of cryptographic algorithms in Sibenik, Croatia and Albena, Bulgaria (3 talks).
2011	"Funciones Hash Cryptographicas y la Competicion SHA-3", in **Conferencia Master**, Facultad de Informatica, UcM, Madrid, Spain (2 hours).

Services to the community

Since 2016, Co-editor in chief of ToSC (with Bart Preneel): IACR Transactions on Symmetric Cryptology.

Selected Program Committees:

I have served on 24 international conferences program committees, most notably:

IACR **FSE** (Fast Software Encryption): 2017, 2015, 2013, 2012, 2011; IACR **Crypto** 2014; IACR **EuroCrypt**: 2017, 2016, 2014; IACR **Asiacrypt** 2014; The international workshop Selected Areas in Cryptology- SAC: 2016, 2015, 2014, 2013, 2012; The Cryptographers track CT-RSA 2017, 2016, 2015...

Journal Reviewer

· Journal of Cryptology; Designs, Codes and Cryptography; International Journal of Foundations of Computer Science; Cryptography and Communications - Discrete Structures; IET Information Security; IEEE Transactions on Information Forensics & Security; ...

Organization Committees

· Member of the steering committee of "Groupe Codage et Cryptographie" (200+ researchers), special interest group of the CNRS research working group Computer Science and Mathematics, "GDR-IM".

· Member of the organizing committee of the 9th International Workshop on Coding and Cryptography - WCC 2015, Paris, France (150 attendees).

Responsibilities in Collaborative Research:

· Responsible of task 4 (Symmetric Constructions and attacks) of ANR project CLE (10/2013-01/2016): Cryptography from Learning with errors, coordinated by V. Lyubashevsky.

· Co-organization and animation of the working group on cryptanalysis for the ANR Bloc (Design and Cryptanalysis of Block Ciphers). 15 meetings between January 2013 and January 2015.

· Responsible for Versailles University (2012) of ANR project Saphir2 (Hash functions).

· Participation to status reports D.SYM.11 and D.SYM.7 for ECRYPTI/II: European Network of Excellence (2004-2012) (33 partners from academics and industry).

Major Collaborations

I have around 30 different coauthors from more than 10 different countries.
Invited Stays:

04/2012	Crypto Group, **UCL, Louvain la neuve, Belgium**. Cooperation with **F-X. Standaert**.
03/2012	Crypto group, **Microsoft, Redmond, US**. Cooperation with **D. Khovratovich**.
11/2008	**FHNW, Windisch, Switzerland**. Cooperation with **W. Meier**.
10/2007	**CSIC**, Consejo Superior de Investigaciones cientificas, Madrid, **Spain**. (**A. Fuster**).

Awards

2016	"Stream ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression ". One of the 2 papers invited to JoC from FSE 2016 (IACR).
2015	"On the security of symmetric key ciphers against Quantum adversaries". Best poster award Qcrypt 2015.
2013-2017	Prime d'Excellence Scientifique.
2012	"Improved rebound attack on the finalist Grøstl". One of the 3 papers invited to JoC from FSE 2012 (IACR).
2010	"QUARK: a lightweight hash". One of the 3 papers invited to JoC from CHES 2010 (IACR).

Grant

I have been granted an ERC starting Grant: I am the PI of QUASYModo, that will start in September 2017. We will study the security of symmetric ciphers against quantum adversaries.

Career Breaks

I have been on maternity leave from 10/2013 to 04/2014 and from 11/2015 to 04/2016.

Publication Record

In cryptography, the most important publications appear in conferences. In particular the three most important and selective conferences are Crypto, Eurocrypt and Asiacrypt, followed by the most important conference on symmetric cryptography: FSE (recently converted to the ToSC journal with a new publication model), all organized by the IACR (International Association for Cryptologic Research). The common practice in the cryptographic community with respect to the authors ordering is to use the alphabetical order. This is the case in the publications presented here.

Journal Papers

[1] C. Boura, V. Lallemand, M. Naya-Plasencia, and V. Suder. Making the impossible possible. *J. Cryptology*, 2016. to appear. (Cited on pages v, vii, xi, 29, 31, 34, 37, vi, 27 and 28.)

[2] M. Kaplan, G. Leurent, A. Leverrier, and M. Naya-Plasencia. Quantum differential and linear cryptanalysis. *IACR Trans. Symmetric Cryptol.*, 2016(1):71–94, 2016. (Cited on pages v, vii, xi, xii, 40, 45, 47, 84, 38, 43, 44 and 66.)

[3] J. Jean, M. Naya-Plasencia, and T. Peyrin. Improved cryptanalysis of AES-like permutations. *J. Cryptology*, 27(4):772–798, 2014. (Cited on pages v, xi, 84 and 66.)

[4] A. Canteaut and M. Naya-Plasencia. Correlation attacks on combination generators. *Cryptography and Communications*, 4(3-4):147–171, 2012. (Cited on pages vii, xi, 29, 30, 84, vi, 27 and 66.)

[5] J. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia. Quark: A lightweight hash. *J. Cryptology*, 26(2):313–339, 2013. (Cited on pages v, xi, 9, 12, 84 and 66.)

[6] Marine Minier and María Naya-Plasencia. A related key impossible differential attack against 22 rounds of the lightweight block cipher lblock. *Inf. Process. Lett.*, 112(16):624–629, 2012. (Not cited.)

[7] Anne Canteaut and María Naya-Plasencia. Parity-check relations on combination generators. *IEEE Transactions on Information Theory*, 58(6):3900–3911, 2012. (Not cited.)

Full Papers in International Peer-Reviewed Conference Proceedings

[8] M. Kaplan, G. Leurent, A. Leverrier, and M. Naya-Plasencia. Breaking symmetric cryptosystems using quantum period finding. In *CRYPTO 2016*, volume 9815 of *LNCS*, pages 207–237. Springer, 2016. (Cited on pages vii, xi, xii, 45, 48, 49, 84, 43, 46 and 66.)

[9] A. Canteaut, S. Carpov, C. Fontaine, T. Lepoint, M. Naya-Plasencia, P. Paillier, and R. Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. In *Fast Software Encryption - FSE 2016*, volume 9783 of *LNCS*, pages 313–333. Springer, 2016. (Cited on pages v, vi, xi, 7, 9, 14, 84, 13 and 66.)

[10] V. Lallemand and M. Naya-Plasencia. Cryptanalysis of full Sprout. In *CRYPTO 2015*, volume 9215 of *LNCS*, pages 663–682. Springer, 2015. (Cited on pages v, vi, ix, xi, 15, 21, 27, 56, 84, viii, 19, 25, 54 and 66.)

[11] A. Canteaut, V. Lallemand, and M. Naya-Plasencia. Related-key attack on full-round PICARO. In *Selected Areas in Cryptography- SAC 2015*, volume 9566 of *LNCS*, pages 86–101. Springer, 2015. (Cited on pages v, ix, xi, 56 and 54.)

[12] C. Boura, M. Naya-Plasencia, and V. Suder. Scrutinizing and improving impossible differential attacks: Applications to CLEFIA, Camellia, LBlock and Simon. In *ASIACRYPT 2014*, volume 8873 of *LNCS*, pages 179–199. Springer, 2014. (Cited on pages vii, xi, 29, 30, 31, 32, 33, 34, 36, 37, 84, vi, 27, 28 and 66.)

[13] V. Lallemand and M. Naya-Plasencia. Cryptanalysis of KLEIN. In *Fast Software Encryption - FSE 2014*, volume 8540 of *LNCS*, pages 451–470. Springer, 2015. (Cited on pages v, vi, ix, xi, 21, 27, 56, viii, 19, 25 and 53.)

[14] A. Canteaut, T. Fuhr, H. Gilbert, M. Naya-Plasencia, and J. Reinhard. Multiple differential cryptanalysis of round-reduced PRINCE. In *Fast Software Encryption - FSE 2014*, volume 8540 of *LNCS*, pages 591–610. Springer, 2015. (Cited on pages v, ix, xi, 40, 56, 37 and 54.)

[15] B. Gérard, V. Grosso, M. Naya-Plasencia, and F.-X. Standaert. Block ciphers that are easier to mask: How far can we go? In *Cryptographic Hardware and Embedded Systems - CHES 2013*, volume 8086 of *LNCS*, pages 383–399. Springer, 2013. (Cited on pages v, vi, xi, 7, 9, 17, 84 and 66.)

[16] A. Canteaut, M. Naya-Plasencia, and B. Vayssière. Sieve-in-the-Middle: Improved MITM techniques. In *CRYPTO 2013 (I)*, volume 8042 of *LNCS*, pages 222–240. Springer, 2013. (Cited on pages vi, vii, xi, 21, 26, 27, 29, 30, 35, 36, 40, 84, 19, 23, 25, 33, 37 and 66.)

[17] J. Jean, M. Naya-Plasencia, and T. Peyrin. Multiple limited-birthday distinguishers and applications. In *Selected Areas in Cryptography- SAC 2013*, LNCS. Springer, 2013. (Not cited.)

[18] M. Naya-Plasencia and T. Peyrin. Practical cryptanalysis of ARMADILLO2. In *Fast Software Encryption - FSE 2012*, volume 7549 of *LNCS*, pages 146–162. Springer, 2012. (Cited on pages v and xi.)

[19] J. Jean, M. Naya-Plasencia, and T. Peyrin. Improved rebound attack on the finalist Grøstl. In *Fast Software Encryption - FSE 2012*, volume 7549 of *LNCS*, pages 110–126. Springer, 2012. (Cited on pages v, vi, xi, 21, 27, 19 and 25.)

[20] M. Naya-Plasencia. How to Improve Rebound Attacks. In *CRYPTO 2011*, volume 6841 of *LNCS*, pages 188–205. Springer, 2011. (Cited on pages vi, xi, 21, 26, 27, 30, 84, 19, 23, 25 and 66.)

[21] M. Naya-Plasencia, D. Toz, and K. Varici. Rebound attack on JH42. In *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 252–269. Springer, 2011. (Cited on pages v, vi, xi, 21, 27, 84, 19, 25 and 66.)

[22] M. A. Abdelraheem, C. Blondeau, M. Naya-Plasencia, M. Videau, and E. Zenner. Cryptanalysis of ARMADILLO2. In *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 308–326. Springer, 2011. (Cited on pages v, xi, 27, 84, 25 and 66.)

[23] M. Minier, M. Naya-Plasencia, and T. Peyrin. Analysis of reduced-SHAvite-3-256 v2. In *Fast Software Encryption - FSE 2011*, volume 6733 of *LNCS*, pages 68–87. Springer, 2011. (Cited on pages v, viii, xi, 55 and 53.)

[24] J. Jean, M. Naya-Plasencia, and M. Schlaffer. Improved analysis of ECHO-256. In *Selected Areas in Cryptography- SAC 2011*, volume 7118 of *LNCS*, pages 19–36. Springer, 2011. (Cited on pages v, vi, viii, xi, 21, 27, 55, 19, 25 and 53.)

[25] S. Knellwolf, W. Meier, and M. Naya-Plasencia. Conditional differential cryptanalysis of trivium and KATAN. In *Selected Areas in Cryptography- SAC 2011*, volume 7118 of *LNCS*, pages 200–212. Springer, 2011. (Not cited.)

[26] J-Ph Aumasson, M. Naya-Plasencia, and M. Saarinen. Practical attack on 8 rounds of the lightweight block cipher KLEIN. In *Indocrypt 2011*, volume 7107 of *LNCS*, pages 134–145. Springer, 2011. (Cited on pages v, ix, xi, 56, viii and 53.)

[27] M. Naya-Plasencia, A. Röck, and W. Meier. Practical analysis of reduced-round Keccak. In *Indocrypt 2011*, volume 7107 of *LNCS*, pages 236–254. Springer, 2011. (Cited on pages v, vi, viii, xi, 21, 27, 55, 19, 25 and 53.)

[28] S. Knellwolf, W. Meier, and M. Naya-Plasencia. Conditional differential cryptanalysis of NLFSR based cryptosystems. In *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 130–145. Springer, 2010. (Cited on pages vii, xi, 16, 29, 84, vi, 27 and 66.)

[29] J. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia. Quark: A lightweight hash. In *Cryptographic Hardware and Embedded Systems - CHES 2010*, volume 6225 of *LNCS*, pages 1–15. Springer, 2010. (Cited on pages v and xi.)

[30] M. Naya-Plasencia, A. Röck, J.-Ph. Aumasson, Y. Laigle-Chapuy, G. Leurent, W. Meier, and T. Peyrin. Cryptanalysis of ESSENCE. In *Fast Software Encryption - FSE 2010*, volume 6147 of *LNCS*, pages 134–152. Springer, 2010. Selectivity: 20/71=0.28. (Not cited.)

[31] D. Khovratovich, M. Naya-Plasencia, A. Röck, and M. Schläffer. Cryptanalysis of Luffa v2 components. In *SAC 2010*, volume 6544 of *LNCS*, pages 388–409. Springer, 2010. (Not cited.)

[32] P. Gauravaram, G. Leurent, F. Mendel, M. Naya-Plasencia, T. Peyrin, C. Rechberger, and M. Schläffer. Cryptanalysis of the 10-round hash and full compression function of SHAvite-3-512. In *Africacrypt 2010*, volume 6055 of *LNCS*, pages 419–436. Springer, 2010. (Cited on pages v, viii, xi, 55 and 53.)

[33] J.-Ph. Aumasson and M. Naya-Plasencia. Cryptanalysis of the MCSSHA hash functions. In *WEWoRC 2009 - Third Western European Workshop on Research in Cryptology*, 2009. (Not cited.)

[34] A. Canteaut and M. Naya-Plasencia. Structural weaknesses of permutations with a low differential uniformity and generalized crooked functions. In *Finite Fields and Applications - Selected papers from the 9th International Conference*, volume 518 of *Contemporary Mathematics*, pages 55–71. AMS, 2009. (Not cited.)

[35] K. Matusiewicz, M. Naya-Plasencia, I. Nikolic, Y. Sasaki, and M. Schlaeffer. Rebound attack on the full LANE compression function. In *ASIACRYPT*, volume 5921 of *LNCS*, pages 106–125. Springer, 2009. (Not cited.)

[36] A. Canteaut and M. Naya-Plasencia. Computing the bias of parity-check relations. In *IEEE International Symposium on Information Theory - ISIT 09*, pages 290–294, Seoul, Korea, 2009. IEEE Press. (Not cited.)

[37] J. Aumasson, E. Brier, W. Meier, M. Naya-Plasencia, and T. Peyrin. Inside the hypercube. In *Australasian Conference on Information Security and Privacy - ACISP 2009*, volume 5594 of *LNCS*, pages 202–213. Springer, 2009. (Not cited.)

[38] M. Naya-Plasencia. Cryptanalysis of Achterbahn-128/80 with a new keystream limitation. In *WEWoRC 2007 - Second Western European Workshop in Research in Cryptology*, volume 4945 of *LNCS*, pages 142–152. Springer, 2008. (Not cited.)

[39] M. Naya-Plasencia. Cryptanalysis of Achterbahn-128/80. In *Fast Software Encryption - FSE 2007*, volume 4593 of *LNCS*, pages 73–86. Springer, 2007. (Not cited.)

Selected Invited Presentations

[40] On lightweight block ciphers and their security. In *15th international conference on Cryptology - Indocrypt 2014*, New Delhi, India. Invited talk, December 2014. (Not cited.)

[41] First practical results on reduced-round Keccak and unaligned rebound attack. In *Keccak and SHA-3 day*, Brussels, Belgium, March 2013. (Not cited.)

[42] Invited panelist in the ECRYPTII hash function workshop 2011. In *Panel discussion "Use and misuse of distinguishers"*, Tallinn, Estonia, 2011. with John Kelsey, Bart Preneel, Thomas Ristenpart and Christian Rechberger. (Not cited.)

[43] On impossible differential cryptanalysis. In *Early Symmetric Crypto - ESC 2015*, Luxembourg, January 2015. (Not cited.)

[44] Meet-in-the-middle through an sbox. In *Early Symmetric Crypto - ESC 2013*, Luxembourg, January 2013. (Not cited.)

[45] Improved rebound attack on the finalist Grøstl. In *Seminar 12031, Symmetric Cryptography*, Dagstuhl Seminar Proceedings, Germany, January 2012. (Not cited.)

[46] Conditional Differential Cryptanalysis of NLFSR-based Cryptosystems and Relation with Dynamic Cube Attacks . In *ICS Forum Talk*, HUT, Finland, 2011. (Not cited.)

[47] Internal collision attack on Maraca. In *Seminar 09031, Symmetric Cryptography*, Dagstuhl Seminar Proceedings, Germany, January 2009. (Cited on pages 22 and 20.)

Technical Reports

[48] C. Rechberger, C. Boura, B. Mennik, and M. Naya-Plasencia. Final hash functions status report (D.SYM.11). ECRYPTII. Report on the SHA-3 competition., 2013. (Not cited.)

[49] P. Gauravaram, F. Mendel, M. Naya-Plasencia, V. Rijmen, and D. Toz. Intermediate Status Report (D.SYM.7). ECRYPTII. Report on the SHA-3 competition. (Not cited.)

[50] Shabal, a submission to NIST cryptographic hash algorithm competition. E. Bresson, A. Canteaut, B. Chevallier-Mames, C. Clavier, T. Fuhr, A. Gouget, T. Icart, J.Misarsky, M. Naya-Plasencia, P. Paillier, T. Pornin, J. Reinhard, C. Thuillet and M. Videau. 2008. 144 pages without appendices. (52 citations). (Not cited.)

Appendix B

Selected publications

Here are some of my selected publications: First, the three papers describing in detail the designs of Quark (*QUARK: a lightweight hash (Extended version)* [AHMN13]), Kreyvium (*A Practical Solution for Efficient Homomorphic-Ciphertext Compression* [CCF+16]) and the easy-to-mask Zorro (*Block Ciphers That Are Easier to Mask: How Far Can We Go?* [GGNPS13]). The first two were selected in the three best papers of CHES2010 and FSE 2016 respectively.

Next, the original paper on merging algorithms (*How to Improve Rebound Attacks?* [NP11]), and its application to the rebound attack on the SHA-3 finalist hash function JH (*Rebound Attack on JH42* [NPTV11]), providing the first distinguishers on JH's full internal permutation.

Then, I grouped a few other papers that provided generalized results on some cryptanalisis families. The second one got invited to the *Journal of Cryptology* after being selected in the three best papers of FSE 2012: *Sieve-in-the-Middle: Improved MITM Attacks* [CNPV13], *Improved Cryptanalysis of AES-like Permutations* [JNP14], *Conditional Differential Cryptanalysis of NLFSR-based Cryptosystems* [KMNP10], *Scrutinizing and Improving Impossible Differential Attacks* [BNS14] and *Correlation attacks on combination generators* [CN12].

Next, two papers on post-quantum symmetric cryptanalysis: *Quantum Differential and Linear Cryptanalysis* [KLLN16b] and *Breaking Symmetric Cryptosystems using Quantum Period Finding* [KLLN16a].

Finally, the dedicated cryptanalysis of two ciphers: *Cryptanalysis of Full Sprout* [LN15a] and *Cryptanalysis of ARMADILLO2* [ABNP+11a]. Both use applications of the merging algorithms.

J. Cryptol. (2013) 26: 313–339
DOI: 10.1007/s00145-012-9125-6

Journal of
CRYPTOLOGY

QUARK: A Lightweight Hash*

Jean-Philippe Aumasson
NAGRA, route de Genève 22, 1033 Cheseaux, Switzerland
jeanphilippe.aumasson@gmail.com

Luca Henzen†
UBS AG, Zürich, Switzerland

Willi Meier
FHNW, Windisch, Switzerland

María Naya-Plasencia‡
University of Versailles, Versailles, France

Communicated by Mitsuru Matsui

Received 29 September 2010
Online publication 10 May 2012

Abstract. The need for lightweight (that is, compact, low-power, low-energy) cryptographic hash functions has been repeatedly expressed by professionals, notably to implement cryptographic protocols in RFID technology. At the time of writing, however, no algorithm exists that provides satisfactory security and performance. The ongoing SHA-3 Competition will not help, as it concerns general-purpose designs and focuses on software performance. This paper thus proposes a novel design philosophy for lightweight hash functions, based on the sponge construction in order to minimize memory requirements. Inspired by the stream cipher Grain and by the block cipher KATAN (amongst the lightest secure ciphers), we present the hash function family QUARK, composed of three instances: U-QUARK, D-QUARK, and S-QUARK. As a sponge construction, QUARK can be used for message authentication, stream encryption, or authenticated encryption. Our hardware evaluation shows that QUARK compares well to previous tentative lightweight hash functions. For example, our lightest instance U-QUARK conjecturally provides at least 64-bit security against all attacks (collisions, multicollisions, distinguishers, etc.), fits in 1379 gate-equivalents, and consumes on average 2.44 μW at 100 kHz in 0.18 μm ASIC. For 112-bit security, we propose S-QUARK, which can be implemented with 2296 gate-equivalents with a power consumption of 4.35 μW.

* Extended version of an article appearing at CHES 2010. The specification of QUARK given in this version differs from that in the CHES 2010 proceedings, namely, the parameter n has been increased to address a flaw in the initial analysis (as reported in [59]). This work was partially supported by European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II.

† This work was done when the second author was with ETHZ, Switzerland.

‡ This work was done when the fourth author was with FHNW, Switzerland.

Key words. Hash functions, Lightweight cryptography, Sponge functions, Cryptanalysis, Indifferentiability.

1. Introduction

Known as cryptographers' Swiss Army Knife, hash functions can serve many different purposes, within applications ranging from digital signatures and message authentication codes to secure passwords storage, key derivation, or forensics data identification. It is fair to say that any system that uses some sort of cryptography includes a hash function. These systems include resource-constrained devices implementing cryptographic functions as hardware blocks, such as RFID tags or systems-on-chip for lightweight embedded devices.

In 2006, Feldhofer and Rechberger [35] pointed out the lack of lightweight hash functions for use in RFID protocols, and gave recommendations to encourage the design of such primitives. The situation has not evolved much in four years, despite a growing demand; besides RFID protocols, lightweight hashes are indeed necessary in all applications that need to minimize the amount of hardware and the power and energy consumption.

Despite the need for lightweight hash functions, a dedicated approach to create secure and efficient algorithms remains to be found. New designs are thus of clear practical interest. In this paper, we address this problem and present a novel approach to design lightweight hashes, illustrated with the proposal of a new family of functions, called QUARK.

We expose our design philosophy in Sect. 2, before a complete specification of QUARK in Sect. 3. Then, Sect. 4 presents the rationale behind the QUARK design, and Sect. 5 reports on our preliminary security analysis. Our hardware implementation is presented in Sect. 6.

Related Works The SHA-3 Competition [52] aims to develop a general-purpose hash function, and received as many as 64 original and diverse submissions. Most of them, however, cannot reasonably be called lightweight, as most need more than (say) 10 000 gate equivalents (GE). An exception is CubeHash [11], which can be implemented with 7630 GE in 0.13 µm ASIC [8] to produce digests of up to 512 bits. For comparison, Feldhofer and Wolkerstorfer [36] reported an implementation of MD5 (128-bit digests, 0.35 µm ASIC) with 8001 GE, O'Neill [53] implemented SHA-1 (160-bit digests, 0.18 µm ASIC) with 6122 GE, and the compression function MAME by Yoshida et al. [61] (256-bit digests, 0.18 µm ASIC) fits in 8100 GE. These designs, however, are still too demanding for many low-end devices.

A step towards lightweight hashing is the 2008 work by Bogdanov et al. [23], which presented constructions based on the lightweight block cipher PRESENT [22]. They proposed to instantiate the Davies–Meyer construction (i.e., $E_m(h) \oplus h$, where $E_m(h)$ denotes the encryption of h with key m by the block cipher E) with PRESENT-80, giving a hash function with 64-bit digests. This hash function, called DM-PRESENT, was implemented with 1600 GE in 0.18 µm ASIC.

Another interesting approach was taken with Shamir's SQUASH [57] keyed hash function, which processes short strings only, offers 64-bit preimage resistance, and is expected to need fewer than 1000 GE. However, SQUASH is not collision resistant—as it targets RFID authentication protocols where collision resistance is unnecessary—and so is inappropriate for applications requiring a collision-resistant hash function.

In 2010, reduced versions of the hash function KECCAK (finalist in the SHA-3 Competition) were proposed [14]. For example, a version of KECCAK returning 64-bit digests was implemented with 2520 GE in 0.13 μm ASIC [45].

After the first publication of QUARK at CHES 2010 [5], other lightweight hash designs appeared, based on the sponge construction. These include PHOTON (presented at CRYPTO 2011 [41]) and SPONGENT (presented at CHES 2011 [24]).

At the time of writing, we have not been informed of any third-party results improving on our preliminary security analysis.

2. Design Philosophy

As noted in [23, Sect. 2], designers of lightweight cryptographic algorithms or protocols have to trade off between two opposite design philosophies. The first consists in creating new schemes from scratch, whereas the second consists in reusing available schemes and adapting them to system constraints. While Bogdanov et al. [23] are more in line with the latter approach—as illustrated by their DM-PRESENT proposal—we tend more towards the former.

Although QUARK borrows components from previous works, it integrates a number of innovations that make it unique and that optimize its lightweightness. As explained in this section, QUARK combines

- A sponge construction with a capacity c equal to the digest length n,
- A core permutation inspired by previous primitives, optimized for reduced resources consumption.

We introduce this design strategy as an attempt to optimize its security-performance ratio. Subsequent proposals of lightweight hash functions followed a similar strategy, with PHOTON and SPONGENT respectively building their core permutations on AES- and SERPENT-like algorithms.

2.1. *Separating Digest Length and Security Level*

We observe that the digest length of a hash function has generally been identified with its security level, with (say) n-bit digests being equivalent to n-bit security against preimage attacks. However, this rule restricts the variety of designs, as it forces designers to exclude design paradigms that may otherwise increase usability or performance.

The notion of *capacity*, introduced in the context of sponge functions [13], was a first step towards a separation of digest length and security level, and thus towards more inventive designs (as showed, by the hash family RADIOGATÚN [12]). In particular, the necessity of n-bit (second) preimage resistance is questionable from a pragmatic

standpoint, when one needs to assume that $2^{n/2}$ is an infeasible effort, to avoid birthday collision search. Designers may thus relax the security requirements against (second) preimages—as informally suggested by several researchers in the context of the SHA-3 Competition—so as to propose more efficient algorithms. For example, in [10] the designer of the SHA-3 candidate CubeHash [11] proposed instances with suboptimal preimage resistance (i.e., below 2^n) for efficiency purposes. We believe that lightweight hashes would benefit from separating digest length and security level. For this, we use a *sponge construction* and target a single security level against all attacks, including second preimage attacks, collision attacks, and any differentiating attack (although higher, optimal resistance of 2^n is offered against preimage attacks).

2.2. *Working with Shift Registers*

Shift registers are a well-known construction in digital circuits, generally implemented as a simple cascade of flip-flops. In cryptography, linear or nonlinear feedback shift registers have been widely used as a building block of stream ciphers, thanks to their simplicity and efficiency of implementation (be it in terms of area or power consumption).

In the design of QUARK, we opt for an algorithm based on bit shift registers combined with (nonlinear) Boolean functions, rather than for a design based on S-boxes combined with a linear layer (as PHOTON and SPONGENT). This is motivated by the simplicity of description and of implementation, and by the close-to-optimal area requirements it induces. Indeed, the register serves both to store the internal state (mandatory in any construction) and to perform the operations bringing confusion and diffusion. The only extra circuit is devoted to the implementation of the feedback functions, which combines bits from the registers to compute the new bit fed into the register.

Since good shift register-based algorithms are known, we do not reinvent the wheel and propose a core algorithm inspired from the stream cipher family Grain [43,44] and from the block cipher family KATAN [30], which are arguably the lightest known secure stream cipher and block cipher. Although both these designs are inappropriate for direct reuse in a hash function, both contain excellent design ideas, which we integrate in our lightweight hash QUARK. A goal of this best-of-both approach is to build on solid foundations while at the same time adapting the algorithm to the attack model of a hash function.

To summarize, our approach is not to instantiate classical general-purpose constructions with lightweight components, but rather to make the whole design lightweight by optimizing all its parts: security level, construction, and core algorithm. An outcome of this design philosophy, the hash family QUARK, is described in the next section.

3. Description of the QUARK Hash Family

This section gives a complete specification of QUARK and of its three proposed instances: U-QUARK, D-QUARK, and S-QUARK. In particle physics, the u-quark is lighter than the d-quark, which itself is lighter than the s-quark; our eponym hash functions compare similarly.

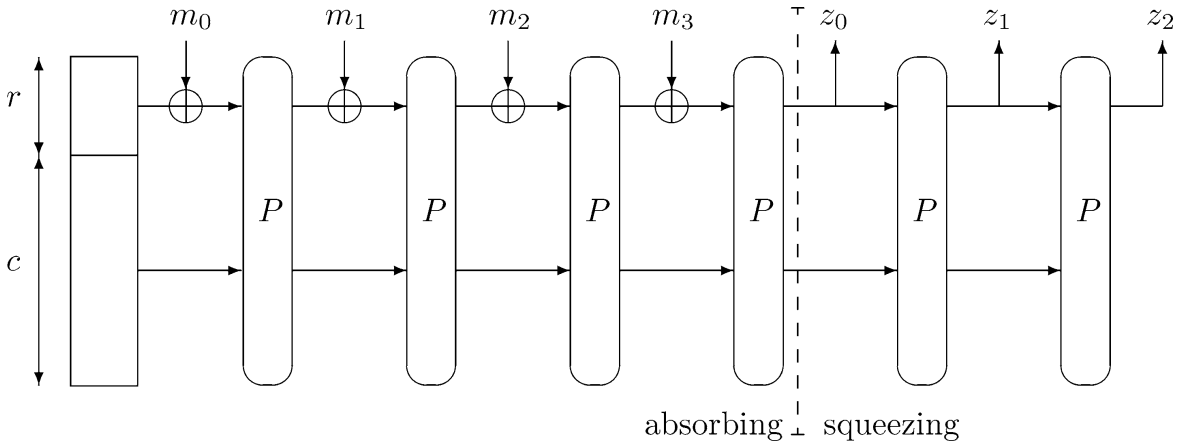


Fig. 1. The sponge construction as used by QUARK, for the example of a 4-block (padded) message.

3.1. *Sponge Construction*

QUARK uses the sponge construction, depicted in Fig. 1, and a b-bit permutation P (that is, a bijective function over $\{0,1\}^b$).

Following the notations introduced in [13], a QUARK instance is parametrized by a *rate* (or block length) r, a *capacity* c, and an *output length* n. The *width* $b = r + c$ of a sponge construction is the size of its internal state. We denote this internal state $s = (s_0, \ldots, s_{b-1})$, where s_0 is referred to as the *first* bit of the state.

Given a predefined initial state of b bits (specified for each instance of QUARK in Appendix A), the sponge construction processes a message m in three steps:

1. **Initialization**: the message is padded by appending a '1' bit followed by the minimal (possibly zero) number of '0' bits to reach a length that is a multiple of r.
2. **Absorbing phase**: the r-bit message blocks are XOR'd with the last r bits of the state (that is, $s_{b-r}, \ldots, s_{b-2}, s_{b-1}$), interleaved with applications of the permutation P. The absorbing phase starts with an XOR between the first block and the state, and it finishes with a call to the permutation P.
3. **Squeezing phase**: the last r bits of the state are returned as output, interleaved with applications of the permutation P, until n bits are returned. The squeezing phase starts with the extraction of r bits, and also finishes with the extraction of r bits.

3.2. *Permutation*

QUARK uses a permutation denoted P, inspired by the stream cipher Grain and by the block cipher KATAN (see Sect. 4.3 for details).

As depicted in Fig. 2, the internal state of P is viewed as three feedback shift registers (FSRs) two nonlinear ones (NFSRs) of $b/2$ bits each, and a linear one (LFSR) of $\lceil \log 4b \rceil$ bits. The state at epoch $t \geq 0$ is thus composed of

- An NFSR X of $b/2$ bits, denoted $X^t = (X_0^t, \ldots, X_{b/2-1}^t)$;
- An NFSR Y of $b/2$ bits, denoted $Y^t = (Y_0^t, \ldots, Y_{b/2-1}^t)$;
- An LFSR L of $\lceil \log 4b \rceil$ bits, denoted $L^t = (L_0^t, \ldots, L_{\lceil \log 4b \rceil - 1}^t)$.

Given a b-bit input, P proceeds in three stages, as described below.

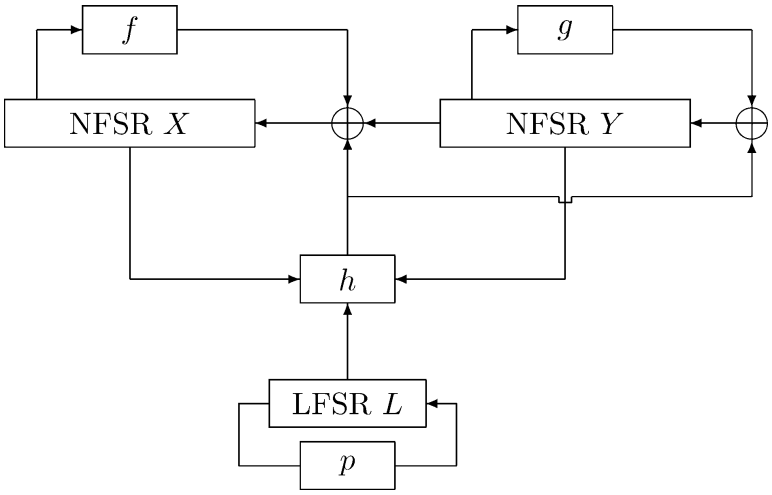


Fig. 2. Diagram of the permutation of QUARK.

3.2.1. *Initialization*

Upon input of the b-bit internal state of the sponge construction $s = (s_0, \ldots, s_{b-1})$, P initializes its internal state as follows:

- X is initialized with the first $b/2$ input bits: $(X_0^0, \ldots, X_{b/2-1}^0) := (s_0, \ldots, s_{b/2-1})$.
- Y is initialized with the last $b/2$ input bits: $(Y_0^0, \ldots, Y_{b/2-1}^0) := (s_{b/2}, \ldots, s_{b-1})$.
- L is initialized to the all-one string: $(L_0^0, \ldots, L_{\lceil \log 4b \rceil - 1}^0) := (1, \ldots, 1)$.

3.2.2. *State Update*

From an internal state (X^t, Y^t, L^t), the next state $(X^{t+1}, Y^{t+1}, L^{t+1})$ is determined by clocking the internal mechanism as follows:

1. The function h is evaluated upon input bits from X^t, Y^t, and L^t, and the result is written h^t:

$$h^t := h\big(X^t, Y^t, L^t\big).$$

2. X is clocked using Y_0^t, the function f, and h^t:

$$\big(X_0^{t+1}, \ldots, X_{b/2-1}^{t+1}\big) := \big(X_1^t, \ldots, X_{b/2-1}^t, Y_0^t + f\big(X^t\big) + h^t\big).$$

3. Y is clocked using the function g and h^t:

$$\big(Y_0^{t+1}, \ldots, Y_{b/2-1}^{t+1}\big) := \big(Y_1^t, \ldots, Y_{b/2-1}^t, g\big(Y^t\big) + h^t\big).$$

4. L is clocked using the function p:

$$\big(L_0^{t+1}, \ldots, L_{\lceil \log 4b \rceil}^{t+1}\big) := \big(L_1^t, \ldots, L_{\lceil \log 4b \rceil - 1}^t, p\big(L^t\big)\big).$$

Table 1. Parameters of the proposed instances of QUARK.

Instance	Rate (r)	Capacity (c)	Width (b)	Rounds ($4b$)	Digest (n)
U-QUARK	8	128	136	544	136
D-QUARK	16	160	176	704	176
S-QUARK	32	224	256	1024	256

3.2.3. *Computation of the Output*

Once initialized, the state of QUARK is updated $4b$ times. The output is defined as the final value of the NFSRs X and Y, using the same bit ordering as for the initialization. That is, the new internal state of the sponge construction is set to

$$s = (s_0, \ldots, s_{b-1}) = \left(X_0^{4b}, X_1^{4b}, \ldots, Y_{b/2-2}^{4b}, Y_{b/2-1}^{4b}\right).$$

3.3. *Proposed Instances*

We propose three different flavors of QUARK: U-QUARK, D-QUARK, and S-QUARK. For each, we give its rate r, capacity c, width b, digest length n, and its functions f, g, and h. For all flavors of QUARK, we have $\lceil \log 4b \rceil = 10$, thus the data-independent LFSR L is of 10 bits. The function p, used by L, is the same for all three instances: given a register L, p returns $L_0 + L_3$.

Table 1 summarizes the parameters of the three instances proposed.

U-QUARK is the lightest flavor of QUARK. It was designed to provide 128-bit preimage resistance and at least 64-bit security against all other attacks, and to admit a parallelization degree of 8. It has parameters $r = 8$, $c = 128$, $b = 136$, $n = 136$.

Function f Given a 68-bit register X, f returns

$$\begin{aligned} &X_0 + X_9 + X_{14} + X_{21} + X_{28} + X_{33} + X_{37} + X_{45} + X_{50} + X_{52} + X_{55} \\ &\quad + X_{55}X_{59} + X_{33}X_{37} + X_9X_{15} + X_{45}X_{52}X_{55} + X_{21}X_{28}X_{33} \\ &\quad + X_9X_{28}X_{45}X_{59} + X_{33}X_{37}X_{52}X_{55} + X_{15}X_{21}X_{55}X_{59} \\ &\quad + X_{37}X_{45}X_{52}X_{55}X_{59} + X_9X_{15}X_{21}X_{28}X_{33} + X_{21}X_{28}X_{33}X_{37}X_{45}X_{52}. \end{aligned}$$

Function g Given a 68-bit register Y, g returns

$$\begin{aligned} &Y_0 + Y_7 + Y_{16} + Y_{20} + Y_{30} + Y_{35} + Y_{37} + Y_{42} + Y_{49} + Y_{51} + Y_{54} \\ &\quad + Y_{54}Y_{58} + Y_{35}Y_{37} + Y_7Y_{15} + Y_{42}Y_{51}Y_{54} + Y_{20}Y_{30}Y_{35} \\ &\quad + Y_7Y_{30}Y_{42}Y_{58} + Y_{35}Y_{37}Y_{51}Y_{54} + Y_{15}Y_{20}Y_{54}Y_{58} \\ &\quad + Y_{37}Y_{42}Y_{51}Y_{54}Y_{58} + Y_7Y_{15}Y_{20}Y_{30}Y_{35} + Y_{20}Y_{30}Y_{35}Y_{37}Y_{42}Y_{51}. \end{aligned}$$

Function h Given 68-bit registers X and Y, and a 10-bit register L, h returns

$$\begin{aligned} &L_0 + X_1 + Y_2 + X_4 + Y_{10} + X_{25} + X_{31} + Y_{43} + X_{56} + Y_{59} \\ &\quad + Y_3X_{55} + X_{46}X_{55} + X_{55}Y_{59} + Y_3X_{25}X_{46} + Y_3X_{46}X_{55} \\ &\quad + Y_3X_{46}Y_{59} + L_0X_{25}X_{46}Y_{59} + L_0X_{25}. \end{aligned}$$

D-QUARK is the second-lightest flavor of QUARK. It was designed to provide 160-bit preimage resistance and at least 80-bit security against all other attacks, and to admit a parallelization degree of 8. It has parameters $r = 16$, $c = 160$, $b = 176$, $n = 176$.

Function f D-QUARK uses the same function f as U-QUARK, but with taps 0, 11, 18, 19, 27, 36, 42, 47, 58, 64, 67, 71, 79 instead of 0, 9, 14, 15, 21, 28, 33, 37, 45, 50, 52, 55, 59, respectively.

Function g D-QUARK uses the same function g as U-QUARK, but with taps 0, 9, 19, 20, 25, 38, 44, 47, 54, 63, 67, 69, 78 instead of 0, 7, 15, 16, 20, 30, 35, 37, 42, 49, 51, 54, 58, respectively.

Function h Given 88-bit registers X and Y, and a 10-bit register L, h returns

$$\begin{aligned} &L_0 + X_1 + Y_2 + X_5 + Y_{12} + Y_{24} + X_{35} + X_{40} + X_{48} + Y_{55} \\ &\quad + Y_{61} + X_{72} + Y_{79} + Y_4X_{68} + X_{57}X_{68} + X_{68}Y_{79} + Y_4X_{35}X_{57} \\ &\quad + Y_4X_{57}X_{68} + Y_4X_{57}Y_{79} + L_0X_{35}X_{57}Y_{79} + L_0X_{35}. \end{aligned}$$

S-QUARK is the heaviest flavor of QUARK. It was designed to provide 224-bit preimage resistance and at least 112-bit security against all other attacks, and to admit a parallelization degree of 16. It has parameters $r = 32$, $c = 224$, $b = 256$, $n = 256$.

Function f S-QUARK uses the same function f as U-QUARK, but with taps 0, 16, 26, 28, 39, 52, 61, 69, 84, 94, 97, 103, 111 instead of 0, 9, 14, 15, 21, 28, 33, 37, 45, 50, 52, 55, 59, respectively.

Function g S-QUARK uses the same function f as U-QUARK, but with taps 0, 13, 28, 30, 37, 56, 65, 69, 79, 92, 96, 101, 109 instead of 0, 7, 15, 16, 20, 30, 35, 37, 42, 49, 51, 54, 58, respectively.

Function h Given 128-bit registers X and Y, and a 10-bit register L, h returns

$$\begin{aligned} &L_0 + X_1 + Y_3 + X_7 + Y_{18} + Y_{34} + X_{47} + X_{58} + Y_{71} + Y_{80} + X_{90} + Y_{91} \\ &\quad + X_{105} + Y_{111} + Y_8X_{100} + X_{72}X_{100} + X_{100}Y_{111} + Y_8X_{47}X_{72} + Y_8X_{72}X_{100} \\ &\quad + Y_8X_{72}Y_{111} + L_0X_{47}X_{72}Y_{111} + L_0X_{47}. \end{aligned}$$

3.4. *Keying* QUARK

As a sponge function, all results known on the sponge construction apply to QUARK. This includes proofs of security for keyed modes of operation, as described in [16,17]. A keyed sponge function processes its input by simply hashing the string composed of the key followed by the said input. The following primitives can then be realized:

- Message authentication code (MAC);
- Pseudorandom generator;
- Stream cipher;
- Random-access stream cipher;
- Key derivation function.

Furthermore, the QUARK instances can easily be modified to operate in the *duplex construction* (a variant of the sponge construction [18]), to allow the realization of functionalities as authenticated encryption or reseedable pseudorandom generators.

4. Design Rationale

This section explains why we opted for a sponge construction and how we chose the internals of the P permutation.

4.1. *Sponge Construction*

The sponge construction [13] is arguably the only real alternative to the classical Merkle–Damgård (MD) construction based on a compression function. Most other known constructions are indeed patched versions of MD, with larger internal state, prefix-free encoding, finalization functions, etc. [7,19,27].

Rather than a (non-injective) compression function, the sponge construction can rely on a *single unkeyed permutation*, and message blocks are integrated with a simple XOR in the internal state. Sponge functions do not require storage of message blocks or of "feedforward" intermediate values as in Davies–Meyer constructions. Nevertheless, the sponge construction needs a larger state to achieve traditional security levels, which partially compensates those memory savings.

The sponge construction was proven to be indifferentiable from a random oracle (up to a bound of approximately $\sqrt{\pi}2^{c/2}$) when instantiated with a random permutation or transformation [13], which is the highest security level a hash construction can achieve. But its most interesting feature is its flexibility: given a fixed permutation P, varying the parameters r, c, and n offers a wide range of trade-offs efficiency/security. This is well illustrated by the interactive page "Tune KECCAK to your requirements" at http://keccak.noekeon.org/tune.html.

Note that during the absorbing phase of QUARK, message blocks are XOR'd to the *last r bits* of the internal state, that is, to the last bits of the Y register. This provides a better diffusion than if the first r bits were used, because differences introduced in the last bits remain in the register, while those in the first quickly disappear due to the bit shifts. During the squeezing phase, digest bits are also extracted from the *last r* bits of the state. The motivation is simple: these are the last bits computed by the permutation; extracting from the first bits would make the computation of the last rounds useless.

4.2. *Separating Digest Length and Security Level*

An originality of QUARK is that its expected security level against (second) preimages differs from its digest length (see Sect. 5.1.2 for a description of the generic attack). In particular, the sponge construction, as used in QUARK, offers a similar security against generic collision attacks and generic second preimage attacks of approximately $2^{c/2}$, and a preimage resistance of approximately 2^c (that is, of $2^{n-r} = 2^c$).

A disadvantage of this approach is that one "wastes" half the digest bits, as far as second preimage resistance is concerned. However, this little penalty brings dramatic performance gains, for it reduces memory requirements by about 50 % compared to classical designs with a same security level. For instance, U-QUARK provides 64-bit security against collisions and second preimages using memory for 146 bits (i.e., the two NFSRs plus the LFSR), while DM-PRESENT provides 64-bit security against preimages but only 32-bit security against collisions with 128 bits of required memory.

Furthermore, the choice of a single security level against all attacks is less confusing for users, who may not be able to determine the security property required for each particular protocol, and then to evaluate the security of the hash function with respect to that property. QUARK provides a single security bound against all attacks, including length extension attacks, multicollision attacks, etc., with increased preimage resistance of 2^c.

4.3. *Permutation Algorithm*

We now justify the choices made to design P. First, we chose an algorithm based on shift registers rather than on S-boxes because in the latter one needs to implement circuits for several Boolean functions (to represent an S-box), rather than a single one in a (serial) implementation of a shift register. Moreover, S-box-based designs typically include a linear transform, which, though cheap to implement, is not necessary in a shift register-based design (as diffusion is performed by the bit shifts within the register). Algorithms based on shift registers also tend to be easier to scale and to implement. To avoid "reinventing the wheel", we borrowed most design ideas from the stream cipher Grain and from the block cipher KATAN, as detailed below.

4.3.1. *Grain*

The Grain family of stream ciphers is composed of Grain-v1, Grain-128, and Grain-128a. The stream cipher Grain-v1 [44] was chosen in 2008 as one of the four "promising new stream ciphers" by the ECRYPT eSTREAM Project. It consists of two 80-bit shift registers combined with three Boolean functions, which makes it one of the lightest designs ever: Good and Benaissa [40] reported an implementation in 0.18 µm ASIC with 1294 GE, for a power consumption of 3.3 µm at 100 kHz. Grain-128 [43] is the 128-bit instance of the Grain family, with 128-bit registers, 128-bit keys, and different Boolean functions. In 2011, a new member of the Grain family was proposed: Grain-128a [1] is an improved version of Grain-128 that incorporates (optional) authentication and countermeasures against known attacks (asymmetric padding, higher nonlinearity).

The main advantages of the Grain ciphers are their simplicity and their performance flexibility (due to the possibility of parallelized implementations). However, a direct reuse of Grain fails to give a secure permutation for a hash function because of "slide

distinguishers" (see Sect. 5.4), of the existence of differential characteristics [29], and of (conjectured) statistical distinguishers for Grain-128 [3,47]. Furthermore, the full Grain-128 can be attacked using advanced cube attacks [32,33]. At the time of writing, no third-party attack on Grain-128a has been published.

4.3.2. *KATAN*

The block cipher family KATAN [30] is inspired by the stream cipher Trivium [28] and builds a keyed permutation with two NFSRs combined with two light quadratic Boolean functions. Its small block sizes (32, 48, and 64 bits) plus the possibility of "burnt-in key" (with the KTANTAN family) lead to very small hardware footprints: 802 GE for KATAN32, and 462 GE for KTANTAN32 [30]. Published third-party cryptanalysis includes shortcut attacks on KTANTAN (unapplicable to KATAN) [21,60], side-channel analysis using algebraic tools [6], and attacks on reduced versions of KATAN [47,48].

KATAN's use of two NFSRs with short feedback delay (unlike Grain's NFSR and LFSR, where feedback delay is at least eight clockings) contributes to a rapid growth of the density and degree of implicit algebraic equations, which complicates differential and algebraic attacks. Another interesting design idea is its use of a LFSR acting both as a counter of the number of rounds, and as an auxiliary input to the inner logic (to simulate two distinct types of rounds). Like Grain, however, KATAN is inappropriate for a direct reuse in a hash function because of its small block size.

4.3.3. *Taking the Best of Both*

Based on the above observations, we decided to borrow the following features from Grain (more precisely, from Grain-v1):

- A mechanism in which each register's update depends on both registers.
- Boolean functions of high degree (up to six, rather than two in KATAN) and high density.

From KATAN, we chose to reuse:

- Two NFSRs instead of an NFSR and an LFSR; Grain's use of a LFSR was motivated by the need to ensure a long period during the keystream generation (where the LFSR is autonomous), but this seems unnecessary for hashing. Moreover, the dissymmetry in such a design is a potential threat for a secure permutation.
- An auxiliary LFSR to act as a counter and to avoid self-similarity of the round function.

Furthermore, we aimed to choose the parallelization degree as a reasonable trade-off between performance flexibility and security. The number of rounds, equal to four times the size of the internal state, was chosen high enough to provide a comfortable security margin against future attacks.

4.3.4. *Choice of the Boolean Functions*

The quality of the Boolean functions in P strongly affects its security. We thus first chose the functions in QUARK according to their individual properties, according to known metrics (see, e.g., [56]). The final choice was made by observing the empirical

Table 2. Properties of the Boolean functions of each QUARK instance (for h, we consider that the parameter L_0 is zero).

Instance	Boolean function	Var.	Deg.	Nonlin. (max)	Resil.
QUARK (all)	f	13	6	3440 (4056)	3
QUARK (all)	g	13	6	3440 (4056)	3
U-QUARK	h	12	3	1280 (2016)	6
D-QUARK	h	15	3	10240 (16320)	9
S-QUARK	h	16	3	20480 (32640)	10

resistance of the *combination* of the three functions to known attacks (see Sects. 5.2–5.3).

The most important properties to consider in the design of Boolean functions for cryptographic applications are

- *Nonlinearity*: the distance to the set of affine functions.
- *Resilience*: the maximum level of correlation immunity, i.e., the maximum number of variables that one can fix and still obtain a balanced function.
- *Algebraic degree*: the maximum degree of a monomial in the algebraic normal form (ANF) of the function.
- *Density*: the proportion of monomials appearing in the ANF.

Nonlinearity and resilience are closely related to the feasibility of attacks based on linear approximations. The degree and density affect the possibility of (higher-order) differential attacks, as they respectively relate to the notions of confusion and diffusion. For efficiency purposes, however, one seldom uses functions of optimal degree and density.

In QUARK, we chose f and g functions similar to the non-linear function of Grain-v1. These functions achieve good, though suboptimal, nonlinearity and resilience (see Table 2). They have degree six and include monomials of each degree below six. An increase of the degree (from two to six) induces only marginal extra cost in terms of hardware gates, since AND logic needs fewer gates than XOR logic (respectively, approximately one and 2.5). The distinct taps for each register break the symmetry of the design. Note that KATAN also employs similar functions for each register's feedback.

As h function, distinct for each flavor of QUARK, we use a function of lower degree than f and g, but with more linear terms to increase the cross-diffusion between the two registers.

4.3.5. *Choice of the Taps*

The taps of f and g, which correspond respectively to indices within the X and Y registers, were chosen with respect to criteria both analytical (invertibility, irregularity of intervals between two consecutive taps) and empirical (measured diffusion and resistance to cube testers and differential attacks). For h, and contrary to Grain, taps are distributed uniformly in X and Y.

For both f, g, and h, no tap is chosen in the last N bits of the register, where N equals eight for U-QUARK and D-QUARK, and 16 for S-QUARK. This allows one to parallelize an implementation of QUARK in N branches by implementing up to N instances of each

Table 3. Security of the proposed instances of QUARK against the standard security notions, in terms of approximate expected number of computations of P.

Instance	Collision resistance	2nd preimage resistance	Preimage resistance
QUARK	$2^{c/2}$	$2^{c/2}$	2^{c}
U-QUARK	2^{64}	2^{64}	2^{128}
D-QUARK	2^{80}	2^{80}	2^{160}
S-QUARK	2^{112}	2^{112}	2^{224}

function in parallel, and thus to compute N updates of the mechanism within a single clock cycle. We chose a lower (maximum) parallelization degree than Grain because the shorter feedback delay contributes to a more rapid growth of the degree and density of the implicit algebraic equations.

5. Preliminary Security Analysis

This section summarizes the known formal security arguments applying to QUARK, as well as our preliminary cryptanalysis results. We applied state-of-the-art cryptanalysis techniques to all flavors of QUARK, including cube attacks and conditional differential attacks, and could obtain results on at most 25 % of P's rounds.

5.1. *The Hermetic Sponge Strategy*

Like the SHA-3 finalist KECCAK [14], QUARK follows the *hermetic sponge strategy*, which consists in adopting the sponge construction with a permutation that should not have exploitable properties. The indifferentiability proof of the sponge construction [13] implies that any non-generic attack on a QUARK hash function leads to a distinguisher for its permutation P (but a distinguisher for P does not necessarily lead to an attack on QUARK). This reduces the security of P to that of the hash function that uses it.

Since QUARK follows the hermetic sponge strategy, the indifferentiability proof in [13] is directly applicable. The proof ensures an expected complexity at least $\sqrt{\pi}2^{c/2}$ against any *differentiating attack*, regardless of the digest length. This covers for example multicollision attacks or herding attacks [46]. Below we give the known refined bounds for the sponge construction regarding the three standard security notions—as described in [42]—and apply them to the parameters of QUARK. Table 3 summarizes the latter results.

5.1.1. *Collision Resistance*

Collisions for the sponge construction can be found by searching collisions on either the n-bit output, or c bits of the internal state (thanks to the possibility of choosing two appropriate r-bit blocks to complete the collision). The collision resistance of a sponge is thus $\min(2^{n/2}, 2^{c/2})$. The proposed instances of QUARK have $c < n$, thus have a collision resistance $2^{c/2}$.

5.1.2. *Second Preimage Resistance*

The generic second preimage attack against QUARK is similar to the generic preimage attack against the hash function CubeHash [11], which was described in [9] and discussed in [2]. It is a meet-in-the-middle attack that searches for a collision on the internal state, starting from the initial state (forwards) and from a subsequent state that leads to the target digest (backwards). For a success chance $1 - 1/e \approx 0.63$, one requires approximately $2^{c/2}$ trials in each direction, since r bits of the state can be controlled by choosing an adequate message block. This is equivalent to more than $2^{c/2+1}$ evaluations of P and thus to more than $b2^{c/2+3}$ clocks of P's mechanism, that is, 2^{74}, 2^{90}, and 2^{123} clocks for U-, D-, and S-QUARK, respectively.

Note that, contrary to CubeHash, the above attack cannot be used to search for preimages of QUARK. This is because one cannot easily determine two distinct final states that yield the same digest, due to the sponge construction—CubeHash does not follow the sponge construction, but a variant of it that allows such an attack.

If n is smaller than $c/2$, however, a second preimage attack has complexity below $2^{c/2}$. We thus have the general formula $\min(2^n, 2^{c/2})$. Our QUARK instances have $n > c/2$, thus offer $2^{c/2}$ security against second preimage attacks.

5.1.3. *Preimage Resistance*

The original proof of security of the sponge construction gave the bound $\min(2^n, 2^{c/2})$ on the (second) preimage resistance of a sponge function. However, no preimage attack proper to sponge functions with complexity $2^{c/2}$ was known. Instead, the expected workload to find a preimage was previously estimated to $2^{n-r} + 2^{c/2}$ in [17, §5.3], although that was not proven optimal. It was later proven [15] that the preimage resistance of a sponge function is essentially $\min(2^{n-r}, 2^c)$: Theorem 2 in [15, §4.2] implies that if the permutation P has no structural flaw, then finding the internal state leading to a given sequence of output blocks has complexity approximately 2^c. The bound on preimage resistance follows from the fact that finding a preimage implies that the state can be recovered. Note that in QUARK, we have $n - r = c$.

The bound above was further refined in [42], which established the bound $\min(2^{\min(b,n)}, \max(2^{\min(b,n)-r}, 2^{c/2}))$. Our QUARK instances have $b = n$, and thus have preimage resistance $\min(2^b, 2^c) = 2^c$. The generic attack consists in searching for the c bits of internal state that squeeze to the $n - r$ target digest, and then performing a meet-in-the-middle to connect the final state to the initial state, as for a second preimage attack.

5.2. *Resistance to Cube Attacks and Cube Testers*

The recently proposed cube attacks [31] and cube testers [4] are higher-order differential cryptanalysis techniques that exploit weaknesses in the algebraic structure of a cryptographic algorithm. Cube testers can be seen as generalized versions of previous monomials tests [34,55]. These techniques are mostly relevant for algorithms based on non-linear components whose ANF has low degree and low density (e.g., the feedback function of an NFSR). Cube testers were for example applied [3] to the stream cipher Grain-128 [43]. Cube attacks/testers are thus tools of choice to attack (reduced versions of) QUARK's permutation, since it resembles to the Grain ciphers, though with an enhanced security.

Table 4. Highest number of rounds t such that the state (X^t, Y^t) could be distinguished from random using a cube tester with the given complexity. Percentage of the total number of rounds is given in parentheses.

Instance	Total rounds	Rounds attacked		
		in 2^8	in 2^{16}	in 2^{24}
U-QUARK	544	109 (20.0 %)	111 (20.4 %)	114 (21.0 %)
D-QUARK	704	144 (20.5 %)	144 (20.5 %)	148 (21.0 %)
S-QUARK	1024	213 (20.8 %)	220 (21.5 %)	222 (21.7 %)

Recall that QUARK targets security against any nontrivial structural distinguisher for its permutation P. We thus applied cube testers rather than cube attacks, for the former are distinguishers rather than key-recovery attacks. We followed a methodology inspired by [3], using bitsliced C implementations of P and an evolutionary algorithm to optimize the parameters of the attack.

In our simplified attack model, the initial state is chosen uniformly at random to apply our distinguishers. Table 4 reports our results, which can be verified using the parameters given in Appendix C.

One observes in Table 4 that all QUARK flavors showed a similar resistance to our cube testers, with a fraction of approximately 21.0 % of the total number of rounds attacked with complexity 2^{24}. It is difficult to extrapolate to higher complexities; the number of rounds attacked cannot be determined analytically to our present knowledge, though heuristical arguments can be given based on previous results [3,4,31]. The number of rounds attackable seems indeed to evolve logarithmically rather than linearly, as a function of the number of variables used. A worst-case assumption (for the designers) is thus that of a linear evolution. Under this assumption, one could attack 126 rounds of U-QUARK in 2^{64} (23.2 % of the total), 162 rounds of D-QUARK in 2^{80} (23.0 %), and 271 rounds of S-QUARK in 2^{112} (26.5 %).

Using an efficient greedy search rather than evolutionary search seems likely to find better cubes, as suggested by a 2010 work by Stankovski [58]: he found a 40-bit cube leading to a distinguisher on 246 rounds, whereas [3] only reached 237 rounds with a cube of same size (an improvement of almost 5 %).

Note that all of Grain-128's 256 rounds could be attacked in [3] in 2^{24}; this result, however, should not be compared to the value 222 reported in Table 4, since the latter attack concerns *any bit of the internal state*, while the former concerns *the first keystream bit* extracted from the internal state after 220 rounds. Our observation of a bias in P after 222 rounds would thus translate into a distinguisher on $222 - 64 = 158$ of the rounds of a stream cipher derived from P. Conversely, one could thus attack $220 + 64 = 284$ rounds of a version of QUARK using Grain-128 in P, since a bias in the output comes from biases in bits of the internal state. The improved results by Stankovski [58] would lead to a distinguisher on $256 + 64 = 320$ rounds of a version of P directly built from Grain-128. Therefore, although S-QUARK uses registers of same length as Grain-128, it is significantly more resistant to cube testers, and shows a comfortable security margin.

5.3. *Resistance to Differential Attacks*

Differential attacks cover all attacks that exploit non-ideal propagation of differences in a cryptographic algorithm (or components thereof). A large majority of attacks on

Table 5. Highest number of rounds t such that the state (X^t, Y^t) could be distinguished from random using a simple differential distinguisher with the given complexity. Percentage of the total number of rounds is given in parentheses.

Instance	Total rounds	Rounds attacked		
		in 2^8	in 2^{16}	in 2^{24}
U-QUARK	544	109 (20.0 %)	116 (21.3 %)	119 (21.9 %)
D-QUARK	704	135 (19.2 %)	145 (20.6 %)	148 (21.0 %)
S-QUARK	1024	206 (20.1 %)	211 (20.6 %)	216 (21.1 %)

hash functions are at least partially differential, starting with the breakthrough results on MD5 and SHA-1. It is thus crucial to analyze the resistance of new designs to differential attacks. We applied a simple search for truncated differential, as well as state-of-the-art conditional differential attacks, as reported below.

5.3.1. *Simple Truncated Differential Attacks*

We first consider a simple attack model where the initial state is assumed chosen uniformly at random and where one seeks differences in the initial state that give biased differences in the state obtained after the (reduced-round) permutation. We focus on high-probability truncated differentials wherein the output difference concerns a small subset of bits (e.g., a single bit). These are sufficient to distinguish the (reduced-round) permutation from a random one, and are easier to find for an adversary than differentials on all the b bits of the state.

First, we observe that it is easy to track differences during the first few rounds, and in particular to find probability-1 (truncated) differential characteristics for reduced-round versions. For example, in U-QUARK, a difference in the bit Y_{29}^0 in the initial state never leads to a difference in the output of f or of h at the 30th round; hence after $(67 + 30) = 97$ rounds, X_0^{97} will be unchanged. Similar examples can be given for 117 rounds of D-QUARK and 188 rounds of S-QUARK. For higher number of rounds, however, it becomes difficult to manually track differences, and so an automated search becomes necessary. As a heuristical indicator of the resistance to differential attacks, we programmed an automated search for high-probability truncated differentials, given an input difference in a single bit. Table 5 presents our results, showing that we could attack approximately as many rounds with truncated differentials as with cube testers (see Table 4).

We expect advanced search techniques to give differential distinguishers for more rounds (e.g., where the sparse difference occurs slightly later in the internal state, as in [29]). However, such methods seem unlikely to apply to the $4b$-round permutation of QUARK. For example, observe that [29] presented a characteristic of probability 2^{-96} for the full 256-round Grain-128; for comparison, S-QUARK makes 1024 rounds, uses more complex feedback functions, and targets a security level of 112 bits; characteristics of probability greater than 2^{-112} are thus highly improbable, even assuming that the adversary can control differences during (say) the first 256 rounds.

5.3.2. *Resistance to Conditional Differential Attacks*

Conditional differential cryptanalysis [47,48] is a technique introduced to analyze NFSR-based algorithms. This analysis is based on a truncated differential or higher-

Table 6. Highest number of rounds t such that the state (X^t, Y^t) could be distinguished from random using a conditional differential distinguisher with the given complexity. Percentage of the total number of rounds is given in parentheses.

Instance	Total rounds	Rounds attacked		
		in 2^2	in 2^{21}	in 2^{27}
U-QUARK	544	111 (20.4 %)	123 (22.6 %)	136 (25.0 %)
D-QUARK	704	117 (16.6 %)	151 (21.4 %)	159 (22.6 %)
S-QUARK	1024	206 (20.1 %)	233 (22.8 %)	237 (23.1 %)

order differential where the attacker controls the first rounds with some conditions of different types. It was applied to (reduced versions of) KATAN, KTANTAN, Grain-v1, and Grain-128 to mount key-recovery attacks and distinguishing attacks. Unsurprisingly, conditional differential cryptanalysis applies to reduced versions of QUARK's permutation P. Roughly speaking, these attacks start with a random initial state, then impose some conditions on the internal variables, and the samples are generated from some bits that will not modify the said conditions. We obtained the best results by using first order differentials on a single output bit of P.

Table 6 shows improved results compared to cube attacks and simple differential attacks (cf. Tables 4 and 5), but are still very far from $4b$ rounds (the slightly lower percentage of rounds attacked of D-QUARK can be explained by its low ratio parallelization degree over state size, although this point would require further investigation). We can expect that some conditions and differences would improve our results, but it seems unlikely that they will reach even $2b$ rounds of QUARK's permutation.

5.4. *Resistance to Slide Resynchronization Attacks*

Suppose that the initial state of the LFSR of QUARK is not the all-one string, but instead is determined by the input of P—that is, P is redefined to accept $(b + 10)$ rather than b input bits. It is then straightforward to distinguish the modified P from a random transform: pick a first initial state (X^0, Y^0, L^0), and consider the second initial state $(X'^0, Y'^0, L'^0) = (X^1, Y^1, L^1)$, i.e., the state obtained after clocking the first state once. Since all rounds are identical, the shift will be preserved between the two states, leading to final states (X^{4b}, Y^{4b}, L^{4b}) and $(X'^{4b}, Y'^{4b}, L'^{4b}) = (X^{4b+1}, Y^{4b+1}, L^{4b+1})$. One thus obtains two input/output pairs satisfying a nontrivial relation, which is a distinguisher for the modified P considered. The principle of the attack is that of slide attacks on block ciphers [20]; we thus call the above a *slide distinguisher*.

The above idea is at the basis of "slide resynchronization" attacks on Grain-v1 and Grain-128 [29,49], which are related-key attacks using as relation a rotation of the key, to simulate a persistent shift between two internal states.

To avoid the slide distinguisher, we use a trick previously used in KATAN: making each round dependent on a bit coming from a LFSR initialized to a fixed value, in order to simulate two distinct types of rounds. It is thus impossible to have two valid initial states shifted by one or more clocks, and such that the shift persists through the $4b$ rounds.

5.5. *Resistance to Side-Channel Attacks*

Implementations of hash functions are potential targets of side-channel attacks in keyed settings when the adversary's goal is to obtain information on the key (previous works include DPA on HMAC-SHA-2 [51], and template attacks on HMAC-SHA-1 [38]). Without keys, side-channel attacks can also be a threat; for example, fault injection can force a message to successfully pass the integrity check, DPA can be used to obtain information on an unknown message (e.g., a password) hashed multiple times with distinct salts, etc.

Like most cryptographic algorithms (including PRESENT [54]), an unprotected implementation of QUARK is likely to be vulnerable to side-channel attacks, in particular to DPA (see [37] for a DPA of Grain). Protected implementations are expected to need at least thrice more gates than the implementations reported below, due to the overhead imposed by countermeasures such as hiding and masking.

6. Hardware Implementation

This section reports our hardware implementation of the QUARK instances. Note that QUARK is not optimized for software (be it 64- or 8-bit processors), and other designs are preferable for such platforms (such as PHOTON [41]). We thus focus our evaluation on hardware efficiency.

Our results arise from pure simulations, and are thus not supported by real measurements on a fabricated chip. However, we believe that this evaluation gives a fair and reliable overview of the overall VLSI performance of QUARK.

6.1. *Architectures*

Three characteristics make QUARK particularly attractive for lightweight hashing: first, the absence in its sponge construction of "feedforward" values, which normally would require additional dedicated memory components; second, its use of shift registers, which are straightforward to implement in hardware; and third, the possibility of several space/time implementation trade-offs. Based on the two extremal trade-off choices, we designed two architecture variants of U-QUARK, D-QUARK, and S-QUARK:

- **Serial architecture**: Only one permutation module, hosting the circuit for the functions f, g, and h, is implemented. Each clock cycle, the bits of the registers X, Y, and L are shifted by one. These architectures correspond to the most compact designs. They contain the minimal circuitry needed to handle incoming messages and to generate the correct output digests.
- **Parallel architecture**: The number of the implemented permutation modules corresponds to the parallelization degree given in Sect. 3.3. The bits in the registers are accordingly shifted. These architectures increase the number of rounds computed per cycle—and therefore the throughput—at extra area costs.

In addition to the three feedback shift registers, each design has a dedicated controller module that handles the sponge process. This module is made up of a finite-state machine and of two counters for the round and the output digest computation. After processing all message blocks during the absorbing phase, the controller switches automat-

ically to the squeezing phase (computation of the digest), if no further r-bit message blocks are given. This implies that the message has been externally padded.

6.2. *Methodology*

We described the serial and parallel architectures of each QUARK instance in functional VHDL, and synthesized the code with Synopsys Design Vision-2009.06 targeting the UMC 0.18 µm 1P6M CMOS technology with the FSA0A_C cell library from Faraday Technology Corporation. We used the generic process (at typical conditions), instead of the low-leakage for two reasons: first, the leakage dissipation is not a big issue in 0.18 µm CMOS, and second, for such small circuits the leakage power is about two orders of magnitude smaller than the total power. To provide a thorough and more reliable analysis, we extended the implementation up to the back-end design. Place and route have been carried out with the help of Cadance Design Systems Velocity-9.1. In a square floorplan, we set a 98 % row density, i.e., the utilization of the core area. Two external power rings of 1.2 µm were sufficient for power and ground distribution. In this technology, six metal layers are available for routing. However, during the routing phase, the fifth and the sixth layers were barely used. The design flow has been placement, clock tree synthesis, and routing with intermediate timing optimizations.

Each architecture was implemented at the target frequency of 100 kHz. As noted in [23,30], this is a typical operating frequency of cryptographic modules in RFID systems. Power simulation was measured for the complete design under real stimuli simulations (two consecutive 512-bit messages) at 100 kHz. The switching activity of the circuit's internal nodes was computed generating Value Change Dump (VCD) files. These were then used to perform statistical power analysis in the velocity tool. Besides the mean value, we also report the peak power consumption, which is a limiting parameter in RFID systems (a maximum of 27 µW is suggested in [36]). Table 7 reports the performance metrics obtained from our simulations at 100 kHz. To give an overview of the best speed achievable, we also implemented the parallel architectures increasing the timing constraints (see Table 8).

6.3. *Results and Discussion*

As reported in Table 7, each of the three serial designs needs fewer than 2300 GE, thus making 112-bit security affordable for restricted-area environments. Particularly appealing for ultra-compact applications is the U-QUARK function, which offers 64-bit security but requires only 1379 GE and dissipates less than 2.5 µW. To the best of our knowledge, U-QUARK is lighter than all previous designs with comparable security claims. We expect an instance of QUARK with 256-bit security (e.g., with $r = 64, c = 512$) to fit in 4500 GE.

Note that in the power results of the QUARK circuits, the single contributions of the mean power consumption are 68 % of internal, 30 % of switching, and 2 % of leakage power. Also important is that the peak value exceeds maximally 27 % of the mean value.

The maximum speed achieved by the parallel cores is 357 Mbps with S-QUARK×16 clocked with a period of 1.4 ns (see Table 8). At this frequency, the values of the power dissipation increase up to 30–60 mW. The leakage component does not contribute significantly. Indeed, 38 % of the total power is devoted to switching, with the rest for

Table 7. Compared hardware performance of PRESENT-based (post-synthesis), KECCAK, and QUARK (post-layout) lightweight hash functions. Note that the QUARK results are post-layout figures. Security is expressed in bits (e.g., "64" in the "Pre." column means that preimages can be found within approximately 2^{64} calls to the function). Throughput and power consumption are given for a frequency of 100 kHz, assuming a long message.

Hash function	Parameters[a]			Security		Area[b]	Lat.	Thr.	Power [µW]	
	n	c	r	Pre	Col	[GE]	[cycles]	[kbps]	Mean	Peak
u-QUARK	136	128	8	128	64	1379	544	1.47	2.44	2.96
u-QUARK×8	136	128	8	128	64	2392	68	11.76	4.07	4.84
d-QUARK	176	160	16	160	80	1702	704	2.27	3.10	3.95
d-QUARK×8	176	160	16	160	80	2819	88	18.18	4.76	5.80
s-QUARK	256	224	32	224	112	2296	1024	3.13	4.35	5.53
s-QUARK×16	256	224	32	224	112	4640	64	50.00	8.39	9.79
Implementations of PRESENT-based hashes from [23] (0.18 µm)										
DM-PRESENT-80	64	64	80	64	32	1600	547	14.63	1.83	–
DM-PRESENT-80	64	64	80	64	32	2213	33	242.42	6.28	–
DM-PRESENT-128	64	64	128	64	32	1886	559	22.90	2.94	–
DM-PRESENT-128	64	64	128	64	32	2530	33	387.88	7.49	–
H-PRESENT-128	128	128	64	128	64	2330	559	11.45	6.44	–
H-PRESENT-128	128	128	64	128	64	4256	32	200.00	8.09	–
Implementations[c] of KECCAK[200] from [14, §9.4] (0.13 µm)										
KECCAK[72,128]	200	128	72	128	64	1300	3870	1.86	–	–
KECCAK[40,160]	200	160	40	160	80	1300	3870	1.03	–	–
Implementations of KECCAK[200] from [45] (0.13 µm)										
KECCAK[72,128]	200	128	72	128	64	2520	900	8.00	5.60	–
KECCAK[72,128]	200	128	72	128	64	4900	900	400.00	27.60	–
KECCAK[40,160]	200	160	40	160	80	2520	900	4.44	5.60	–
KECCAK[40,160]	200	160	40	160	80	4900	900	222.22	27.60	–
Implementations of PHOTON from [42] (0.18 µm)										
PHOTON-128/16/16	128	128	16	112	64	1122	996	1.61	2.29	–
PHOTON-128/16/16	128	128	16	112	64	1708	156	10.26	3.45	–
PHOTON-160/36/36	160	160	36	124	80	1396	1332	2.70	2.74	–
PHOTON-160/36/36	160	160	36	124	80	2117	180	20.00	4.35	–
PHOTON-224/32/32	224	224	32	112	64	1736	1716	1.86	4.01	–
PHOTON-224/32/32	224	224	32	112	64	2786	204	15.69	6.50	–
Implementations of SPONGENT from [24] (0.13 µm)										
SPONGENT-128	128	128	8	120	64	1060	2380	0.34	2.20	–
SPONGENT-128	128	128	8	120	64	1687	70	11.43	3.58	–
SPONGENT-160	176	160	16	144	80	1329	3960	0.40	2.85	–
SPONGENT-160	176	160	16	144	80	2190	90	17.78	4.47	–
SPONGENT-224	240	224	16	208	112	1728	7200	0.22	3.73	–
SPONGENT-224	240	224	16	208	112	2903	120	13.33	5.97	–

[a]For the non-sponge PRESENT-based functions, n, c, r are respectively the lengths of a digest, internal state, and message block.

[b]For QUARK implementations, one GE is the area of a 2-input drive-one NAND gate, i.e., in the target 0.18 µm technology, 9.3744 μm^2.

[c]These implementations use external memory to store the internal state.

Table 8. Maximum-speed performances of our parallel implementations of QUARK, compared with the implementation of KECCAK[200] in [14].

Hash function	Block [bits]	Area [GE]	Lat. [cycles]	Freq. [MHz]	Thr. [Mbps]	Power [µW]	
						Mean	Peak
U-QUARK×8	8	3032	68	714	84.0	30.46	37.01
D-QUARK×8	16	3561	88	714	129.8	37.14	43.35
S-QUARK×16	32	6220	64	714	357.0	65.34	75.27
KECCAK[$r = 40$, $c = 160$][a]	40	1600	3870	714	7.4	–	–

[a]This implementation uses external memory to store the internal state, and is on 0.13 µm technology.

internal power. We do not exclude the possibility to reach higher speed ratios with different architectures. In practice, the latency could be further reduced by implementing more permutation modules. Due to the tap configuration in the function f, g, and h, this would also increase the circuit complexity (i.e., more area and lower frequency), which was outside the scope of this analysis.

6.3.1. *Comparison to Previous Designs*

As reported in Table 7, the functions DM-PRESENT-80/128 and H-PRESENT-128 also offer time/space implementation trade-offs. For a same second preimage resistance of at least 64 bits, U-QUARK fits in a smaller area (1379 vs. 1600 GE), and even the 80-bit-secure D-QUARK does not need more GE than DM-PRESENT-128. In terms of throughput, however, QUARK underperforms PRESENT-based designs. Not only, the figures provided in Tables 7 and 8 are computed for a generic long-size message, omitting the latency of the squeezing phase. Indeed, since QUARK has a smaller rate than the digest size, the complete hash value is only generated after additional executions of the permutation P. In the case of small-size messages, this behavior penalizes QUARK, and more generally the sponge construction, with respect to the PRESENT-based hash functions. The significantly smaller speed values may be due to QUARK's higher security margin (note that 26 of the 31 rounds of PRESENT, as a block cipher, were attacked [25], suggesting a thin security margin against distinguishers in the "open key" model of hash functions). Moreover, QUARK provides at least 64-bit preimage resistance, while both DM-PRESENT versions are limited to a 32-bit collision resistance, making collision search practical.

Compared to the small versions of KECCAK, which follow the same design philosophy as QUARK, the latter have lower area requirements for a given security level (even when comparing 0.18 µm with 0.13 µm technologies). Implementations in [45] also suggest a higher power consumption than QUARK.

6.3.2. *Comparison to Subsequent Designs*

The lightweight hash functions PHOTON and SPONGENT, which appeared after the publication of QUARK, also use a sponge construction (or a slightly modified version thereof). However, they build P on highly optimized block cipher-like constructions,

and seem to allow slightly more compact implementations than QUARK (note, however, that SPONGENT implementations were realized on 0.13 µm technology, against 0.18 µm for QUARK and PHOTON). Interestingly, each of the three functions has unique characteristics, and none seems to dominate on all aspects; for example, SPONGENT and PHOTON have slightly lower footprints; however, the former has a significantly lower throughput than QUARK and PHOTON, while the latter appears to have a lower security margin.

Acknowledgements

We would like to thank the KECCAK team for many helpful comments on a preliminary version of this article, Hidenori Kuwakado for noticing an error in the first C implementation of QUARK, and the reviewers of the Journal of Cryptology for their insightful feedback.

Willi Meier is supported by the Hasler foundation www.haslerfoundation.ch under project number 08065. María Naya-Plasencia is supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center of the Swiss National Science Foundation under grant number 5005-67322, and partially by the French Agence Nationale de la Recherche through the SAPHIR2 project under Contract ANR-08-VERS-014.

Appendix A. Initial States

The initial states of each instance of QUARK are chosen as the first bits of their SHA-256 digest (e.g., U-QUARK's initial state corresponds to the first 136 bits of SHA-256("u-quark")). The hexadecimal values below present the IV from s_0 to s_{b-1} (cf. Sect. 3.2). That is, the initial state of U-QUARK has $X_0 = 1, X_1 = 1, X_2 = 0, X_3 = 1$, which corresponds to the first hexadecimal digit D (1101 in binary).

U-QUARK: `D8DACA44414A099719C80AA3AF065644DB`
D-QUARK: `CC6C4AB7D11FA9BDF6EEDE03D87B68F91BAA706C20E9`
S-QUARK: `397251CEE1DE8AA73EA26250C6D7BE128CD3E79DD718C24B8A19D09C2492DA5D`

Appendix B. Test Values

Using the same endianness convention as in Appendix A, we give below the intermediate values of the internal state when hashing the empty message (that is, after padding, the blocks `80`, `8000`, `80000000` for U-QUARK to S-QUARK).

U-QUARK

Initial state after XOR with the message block `80`:
`D8DACA44414A099719C80AA3AF0656445B`

State after applying the only permutation of the absorbing phase:
9A03A9DEFBB9ED3867DAB18EC039276212

States after each permutation of the squeezing phase:
4C983B073679AD44498C7DED5B5A3EC16B
CD18A9431D86D59100F114398B45869375
DE2DA1946E4D047A641F31EF8A884E13BC
61A3BF954EC85422ADAF58349D485D2CAB
A526ABB27ABD03661D3E04876FCB7B6423
C47103489721DEF7E7F67F6952F4180A14
FA5671E806083DB70885867946CE0BC947
25C149CA3418D1F86FDC4A195827174250
47A44A6590C7A05B8A3B641B262ECB2ED0
FE3D800B292D9DC5E766BAFD9F1CD36A8B
DC21EF190455FD30B84F8012ACC03E72A3
865D7978420A74F7F1901C7724F97FE013
50C180B068D3CD04CE25F1DDAB868E9DBB
62347472491643FABB8051344C4CA38CD8
89C3B410F2EBE58E8CCC9AB056A5E50A00

Digest returned:
126B75BCAB23144750D08BA313BBD800A4

D-QUARK

Initial state after XOR with the message block 8000:
CC6C4AB7D11FA9BDF6EEDE03D87B68F91BAA706CA0E9

State after applying the only permutation of the absorbing phase:
E1AFDDED75F72D33AE3F60D3A1A9E9FA759AC6F082C7

States after each permutation of the squeezing phase:
D013143E679FAEC7A2B6EB458498FED5DC498145F380
7D9E93000F8A30236E8FD3E85BE3C096705E2FD6E231
FC595197C3415152DB7FF0E246CD4AB92E98D3C2578E
FA0E4CF5390554A0841F15310C908C4066F8CF162FF4
9498279CAA9AC4C293245DB08CA40BAF61FB32EFC2A4
ADAD6159EAB1656B022A4B06E4454A4B025426B302E1
D30C6F301AD93B8A07E212732B7B6C7B0DA1FDC38BF3
12F7B39AFC823FB430892B89D0F6CBAADF36A46C7AEA
0D6B4D0554F5F343BCB26AAC85CC8019BC486AF88477

Digest returned:
82C7F380E231578E2FF4C2A402E18BF37AEA8477298D

S-QUARK

Initial state after XOR with the message block 80000000:
397251CEE1DE8AA73EA26250C6D7BE128CD3E79DD718C24B8A19D09CA492DA5D

State after applying the only permutation of the absorbing phase:
3D63F54100A7BC5135692F3BDE1563F7998A6965FE6D26AB40262D2003256214

States after each permutation of the squeezing phase:
12603448212FCAF31D611E986F6C9C10C42E1DD79D91B74407ECE15AB92E811C
FFDBEED704CC5D6BE6CCF7E32A9F563278DAA52D38C870588E84DBEA321AE86B
5804FEAD1E4357EC99D9B6D98624F4F649A9FAF384C434D7C79988A0AB4B0E7A
3B2EFFC05882C5BCA5A191FD20945445AC1C1A660B1B8FAD0F746670E9C22C42
7B2184B713EE554B914D66447D76F725340199622EE4F768069F2C07882FCCDE
D69E1FA2067F8A54606D81F9DE212D51C48B3C4C12CFF9EE013740118C22BFF6

Digest returned:
03256214B92E811C321AE86BAB4B0E7AE9C22C42882FCCDE8C22BFF6A0A1D6F1

Appendix C. Index Sets for Cube Testers

Below we give the 24-bit index sets used to obtain the results in Table 4:

U-QUARK:
{0, 5, 6, 8, 15, 17, 18, 23, 29, 33, 34, 42, 44, 45, 57, 58, 72, 78, 99, 101, 114, 118, 120, 126}
D-QUARK:
{0, 22, 24, 25, 31, 32, 34, 39, 43, 48, 55, 58, 62, 69, 82, 90, 94, 101, 128, 133, 140, 141, 143, 159}
S-QUARK:
{12, 19, 23, 30, 36, 37, 38, 43, 46, 49, 51, 56, 57, 59, 66, 91, 95, 97, 162, 170, 182, 185, 219, 249}

References

[1] M. Ågren, M. Hell, T. Johansson, W. Meier, A new version of Grain-128 with authentication, in *ECRYPT Symmetric Key Encryption Workshop 2011* (2011). Available at http://skew2011.mat.dtu.dk/
[2] J.-P. Aumasson, E. Brier, W. Meier, M. Naya-Plasencia, T. Peyrin, Inside the hypercube, in *ACISP*, ed. by C. Boyd, J. Manuel González Nieto. LNCS, vol. 5594 (Springer, Berlin, 2009), pp. 202–213
[3] J.-P. Aumasson, I. Dinur, L. Henzen, W. Meier, A. Shamir, Efficient FPGA implementations of highly-dimensional cube testers on the stream cipher Grain-128, in *SHARCS* (2009)
[4] J.-P. Aumasson, I. Dinur, W. Meier, A. Shamir, Cube testers and key recovery attacks on reduced-round MD6 and Trivium, in *FSE*, ed. by O. Dunkelman. LNCS, vol. 5665 (Springer, Berlin, 2009), pp. 1–22
[5] J.-P. Aumasson, L. Henzen, W. Meier, M. Naya-Plasencia, Quark: a lightweight hash, in *Mangard and Standaert [50]* (2010), pp. 1–15
[6] G.V. Bard, N. Courtois, J. Nakahara, P. Sepehrdad, B. Zhang, Algebraic, AIDA/cube and side channel analysis of KATAN family of block ciphers, in *Gong and Gupta [39]* (2010), pp. 176–196
[7] M. Bellare, T. Ristenpart, Multi-property-preserving hash domain extension and the EMD transform, in *ASIACRYPT*, ed. by X. Lai, K. Chen. LNCS, vol. 4284 (Springer, Berlin, 2006), pp. 299–314
[8] M. Bernet, L. Henzen, H. Kaeslin, N. Felber, W. Fichtner, Hardware implementations of the SHA-3 candidates Shabal and CubeHash, in *CT-MWSCAS* (IEEE, New York, 2009)

[9] D.J. Bernstein, CubeHash appendix: complexity of generic attacks. Submission to NIST, 2008. http://cubehash.cr.yp.to/submission/generic.pdf
[10] D.J. Bernstein, CubeHash parameter tweak: 16 times faster, 2009. http://cubehash.cr.yp.to/submission/tweak.pdf
[11] D.J. Bernstein, CubeHash specification (2.B.1). Submission to NIST (Round 2), 2009. http://cubehash.cr.yp.to/submission2/spec.pdf
[12] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, RADIOGATÚN, a belt-and-mill hash function, in *Second NIST Cryptographic Hash Function Workshop* (2006). http://radiogatun.noekeon.org/
[13] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, On the indifferentiability of the sponge construction, in *EUROCRYPT*, ed. by N.P. Smart. LNCS, vol. 4965 (Springer, Berlin, 2008), pp. 181–197
[14] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, Keccak sponge function family main document (version 2.1). Submission to NIST (Round 2), 2010. http://keccak.noekeon.org/Keccak-main-2.1.pdf
[15] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, Sponge-based pseudo-random number generators, in *Mangard and Standaert [50]* (2010), pp. 33–47
[16] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, On the security of the keyed sponge construction, in *ECRYPT Symmetric Key Encryption Workshop 2011* (2011). Available at http://skew2011.mat.dtu.dk/
[17] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, Sponge functions. http://sponge.noekeon.org/SpongeFunctions.pdf
[18] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, Duplexing the sponge: single-pass authenticated encryption and other applications. Cryptology ePrint Archive, Report 2011/499, 2011
[19] E. Biham, O. Dunkelman, A framework for iterative hash functions—HAIFA. Cryptology ePrint Archive, Report 2007/278, 2007
[20] A. Biryukov, D. Wagner, Slide attacks, in *FSE*, ed. by L. Knudsen. LNCS, vol. 1636 (Springer, Berlin, 1999), pp. 245–259
[21] A. Bogdanov, C. Rechberger, A 3-subset meet-in-the-middle attack: cryptanalysis of the lightweight block cipher KTANTAN. Cryptology ePrint Archive, Report 2010/532, 2010
[22] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, C. Vikkelsoe, PRESENT: an ultra-lightweight block cipher, in *CHES*, ed. by P. Paillier, I. Verbauwhede. LNCS, vol. 4727 (Springer, Berlin, 2007), pp. 450–466
[23] A. Bogdanov, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, Hash functions and RFID tags: mind the gap, in *CHES*, ed. by E. Oswald, P. Rohatgi. LNCS, vol. 5154 (Springer, Berlin, 2008), pp. 283–299
[24] A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varici, I. Verbauwhede, SPONGENT: a lightweight hash function, in *CHES*, ed. by B. Preneel, T. Takagi. LNCS, vol. 6917 (Springer, Berlin, 2011), pp. 312–325
[25] J.Y. Cho, Linear cryptanalysis of reduced-round PRESENT, in *CT-RSA*, ed. by J. Pieprzyk. LNCS, vol. 5985 (Springer, Berlin, 2010), pp. 302–317
[26] C. Clavier, K. Gaj (eds.), *Cryptographic Hardware and Embedded Systems—CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6–9, 2009, Proceedings*. LNCS, vol. 5747 (Springer, Berlin, 2009)
[27] J.-S. Coron, Y. Dodis, C. Malinaud, P. Puniya, Merkle–Damgård revisited: how to construct a hash function, in *CRYPTO*, ed. by V. Shoup. LNCS, vol. 3621 (Springer, Berlin, 2005), pp. 430–448
[28] C. De Cannière, B. Preneel, Trivium, in *New Stream Cipher Designs*. LNCS, vol. 4986 (Springer, Berlin, 2008), pp. 84–97
[29] C. De Cannière, Ö. Kücük, B. Preneel, Analysis of Grain's initialization algorithm, in *SASC 2008* (2008)
[30] C. De Cannière, O. Dunkelman, M. Knezevic, KATAN and KTANTAN—a family of small and efficient hardware-oriented block ciphers, in *Clavier and Gaj [26]* (2009), pp. 272–288
[31] I. Dinur, A. Shamir, Cube attacks on tweakable black box polynomials, in *EUROCRYPT*, ed. by A. Joux. LNCS, vol. 5479 (Springer, Berlin, 2009), pp. 278–299
[32] I. Dinur, A. Shamir, Breaking Grain-128 with dynamic cube attacks. Cryptology ePrint Archive, Report 2010/570, 2010
[33] I. Dinur, T. Güneysu, C. Paar, A. Shamir, R. Zimmermann, An experimentally verified attack on full Grain-128 using dedicated reconfigurable hardware, in *ASIACRYPT*, ed. by D.H. Lee, X. Wang. LNCS, vol. 7073 (Springer, Berlin, 2011), pp. 327–343

[34] H. Englund, T. Johansson, M.S. Turan, A framework for chosen IV statistical analysis of stream ciphers, in *INDOCRYPT*, ed. by K. Srinathan, C. Pandu Rangan, M. Yung. LNCS, vol. 4859 (Springer, Berlin, 2007), pp. 268–281
[35] M. Feldhofer, C. Rechberger, A case against currently used hash functions in RFID protocols, in *OTM Workshops (1)*, ed. by R. Meersman, Z. Tari, P. Herrero. LNCS, vol. 4277 (Springer, Berlin, 2006), pp. 372–381
[36] M. Feldhofer, J. Wolkerstorfer, Strong crypto for RFID tags—a comparison of low-power hardware implementations, in *ISCAS 2007* (IEEE, New York, 2007), pp. 1839–1842
[37] W. Fischer, B.M. Gammel, O. Kniffler, J. Velten, Differential power analysis of stream ciphers, in *SASC 2007* (2007)
[38] P.-A. Fouque, G. Leurent, D. Réal, F. Valette, Practical electromagnetic template attack on HMAC, in *Clavier and Gaj [26]* (2009), pp. 66–80
[39] G. Gong, K.C. Gupta (eds.), *Progress in Cryptology—INDOCRYPT 2010—11th International Conference on Cryptology in India*, Hyderabad, India, December 12–15, 2010. LNCS, vol. 6498 (Springer, Berlin, 2010)
[40] T. Good, M. Benaissa, Hardware performance of eSTREAM phase-III stream cipher candidates, in *SASC* (2008)
[41] J. Guo, T. Peyrin, A. Poschmann, The PHOTON family of lightweight hash functions, in *CRYPTO*, ed. by P. Rogaway. LNCS, vol. 6841 (Springer, Berlin, 2011), pp. 222–239
[42] J. Guo, T. Peyrin, A. Poschmann, The PHOTON family of lightweight hash functions (2011). Available on https://sites.google.com/site/photonhashfunction/. Full version of [41]
[43] M. Hell, T. Johansson, A. Maximov, W. Meier, A stream cipher proposal: Grain-128, in *IEEE International Symposium on Information Theory (ISIT 2006)* (2006)
[44] M. Hell, T. Johansson, W. Meier, Grain: a stream cipher for constrained environments. *Int. J. Wirel. Mob. Comput.* **2**(1), 86–93 (2007)
[45] E.B. Kavun, T. Yalcin, A lightweight implementation of Keccak hash function for radio-frequency identification applications, in *RFIDSec*, ed. by S.B.O. Yalcin. LNCS, vol. 6370 (Springer, Berlin, 2010), pp. 258–269
[46] J. Kelsey, T. Kohno, Herding hash functions and the Nostradamus attack, in *EUROCRYPT*, ed. by S. Vaudenay. LNCS, vol. 4004 (Springer, Berlin, 2006), pp. 183–200
[47] S. Knellwolf, W. Meier, M. Naya-Plasencia, Conditional differential cryptanalysis of NLFSR-based cryptosystems, in *ASIACRYPT*, ed. by M. Abe. LNCS, vol. 6477 (Springer, Berlin, 2010), pp. 130–145
[48] S. Knellwolf, W. Meier, M. Naya-Plasencia, Conditional differential cryptanalysis of Trivium and KATAN, in *Selected Areas in Cryptography*, ed. by A. Miri, S. Vaudenay. LNCS, vol. 7118 (Springer, Berlin, 2012), pp. 200–212
[49] Y. Lee, K. Jeong, J. Sung, S. Hong, Related-key chosen IV attacks on Grain-v1 and Grain-128, in *ACISP*, ed. by Y. Mu, W. Susilo, J. Seberry. LNCS, vol. 5107 (Springer, Berlin, 2008), pp. 321–335
[50] S. Mangard, F.-X. Standaert (eds.), *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop*, Santa Barbara, CA, USA, August 17–20, 2010. LNCS, vol. 6225 (Springer, Berlin, 2010)
[51] R.P. McEvoy, M. Tunstall, C.C. Murphy, W.P. Marnane, Differential power analysis of HMAC based on SHA-2, and countermeasures, in *WISA*, ed. by S. Kim, M. Yung, H.-W. Lee. LNCS, vol. 4867 (Springer, Berlin, 2007), pp. 317–332
[52] NIST, Cryptographic hash algorithm competition. http://www.nist.gov/hash-competition
[53] M. O'Neill, Low-cost SHA-1 hash function architecture for RFID tags, in *Workshop on RFID Security RFIDsec* (2008)
[54] M. Renauld, F.-X. Standaert, Combining algebraic and side-channel cryptanalysis against block ciphers, in *30th Symposium on Information Theory in the Benelux* (2009), pp. 97–104. http://www.dice.ucl.ac.be/~fstandae/68.pdf
[55] M.-J.O. Saarinen, Chosen-IV statistical attacks on eStream ciphers, in *SECRYPT*, ed. by M. Malek, E. Fernández-Medina, J. Hernando (INSTICC Press, Setubal, 2006), pp. 260–266
[56] P. Sarkar, S. Maitra, Construction of nonlinear boolean functions with important cryptographic properties, in *EUROCRYPT*, ed. by B. Preneel. LNCS, vol. 1807 (Springer, Berlin, 2000), pp. 485–506
[57] A. Shamir, SQUASH—a new MAC with provable security properties for highly constrained devices such as RFID tags, in *FSE*, ed. by K. Nyberg. LNCS, vol. 5086 (Springer, Berlin, 2008), pp. 144–157

[58] P. Stankovski, Greedy distinguishers and nonrandomness detectors, in *Gong and Gupta [39]* (2010), pp. 210–226
[59] G. Van Assche, Errata for Keccak presentation. Email sent to the NIST SHA-3 mailing list on Feb. 7, 2011, on behalf of the Keccak team
[60] L. Wei, C. Rechberger, J. Guo, H. Wu, H. Wang, S. Ling, Improved meet-in-the-middle cryptanalysis of KTANTAN (poster), in *ACISP*, ed. by U. Parampalli, P. Hawkes. LNCS, vol. 6812 (Springer, Berlin, 2011), pp. 433–438
[61] H. Yoshida, D. Watanabe, K. Okeya, J. Kitahara, H. Wu, O. Kucuk, B. Preneel, MAME: a compression function with reduced hardware requirements, in *ECRYPT Hash Workshop 2007* (2007)

Stream ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression

Anne Canteaut[1], Sergiu Carpov[2], Caroline Fontaine[3⋆], Tancrède Lepoint[4⋆⋆], María Naya-Plasencia[1], Pascal Paillier[4⋆⋆], and Renaud Sirdey[2]

[1] Inria, France, {anne.canteaut,maria.naya_plasencia}@inria.fr
[2] CEA LIST, France, {sergiu.carpov,renaud.sirdey}@cea.fr
[3] CNRS/Lab-STICC and Telecom Bretagne and UEB, caroline.fontaine@telecom-bretagne.eu
[4] CryptoExperts, France, {tancrede.lepoint,pascal.paillier}@cryptoexperts.com

Abstract. In typical applications of homomorphic encryption, the first step consists for Alice to encrypt some plaintext m under Bob's public key pk and to send the ciphertext $c = \mathsf{HE}_{\mathsf{pk}}(m)$ to some third-party evaluator Charlie. This paper specifically considers that first step, *i.e.* the problem of transmitting c as efficiently as possible from Alice to Charlie. As previously noted, a form of compression is achieved using hybrid encryption. Given a symmetric encryption scheme E, Alice picks a random key k and sends a much smaller ciphertext $c' = (\mathsf{HE}_{\mathsf{pk}}(k), \mathsf{E}_k(m))$ that Charlie decompresses homomorphically into the original c using a decryption circuit $\mathcal{C}_{\mathsf{E}^{-1}}$.

In this paper, we revisit that paradigm in light of its concrete implementation constraints; in particular E is chosen to be an additive IV-based stream cipher. We investigate the performances offered in this context by Trivium, which belongs to the eSTREAM portfolio, and we also propose a variant with 128-bit security: Kreyvium. We show that Trivium, whose security has been firmly established for over a decade, and the new variant Kreyvium have an excellent performance.

Keywords. Stream Ciphers, Homomorphic cryptography, Ciphertext compression, Trivium

1 Introduction

Since the breakthrough result of Gentry [Gen09] achieving fully homomorphic encryption (FHE), many works have been published on simpler and more efficient schemes based on homomorphic encryption. Because they allow arbitrary computations on encrypted data, FHE schemes suddenly opened the way to exciting new applications, in particular cloud-based services in several areas (see *e.g.* [NLV11,GLN12,LLN14]).

Compressed encryption. In these cloud applications, it is often assumed that some data is sent encrypted under a homomorphic encryption (HE) scheme to the cloud to be processed in a way or another. It is thus typical to consider, in the first step of these applications, that a user (Alice) encrypts some data m under some other user's public key pk (Bob) and sends some homomorphic ciphertext $c = \mathsf{HE}_{\mathsf{pk}}(m)$ to a third-party evaluator in the Cloud (Charlie). The roles of Alice and Bob are clearly distinct, even though they might be played by the same entity in some applications.

However, all HE schemes proposed so far suffer from a very large ciphertext expansion; the transmission of c between Alice and Charlie is therefore a very significant bottleneck in practice. The problem of reducing the size of c as efficiently as possible has first been considered in [NLV11] wherein m is encrypted with a symmetric encryption scheme E under some key k randomly chosen by Alice, who then sends a much smaller ciphertext $c' = (\mathsf{HE}_{\mathsf{pk}}(k), \mathsf{E}_k(m))$ to Charlie. Given c', Charlie then exploits the homomorphic property of HE and recovers the original

$$c = \mathsf{HE}_{\mathsf{pk}}(m) = \mathcal{C}_{\mathsf{E}^{-1}}\left(\mathsf{HE}_{\mathsf{pk}}(k), \mathsf{E}_k(m)\right)$$

by homomorphically evaluating the decryption circuit $\mathcal{C}_{\mathsf{E}^{-1}}$. This can be assimilated to a *compression method* for homomorphic ciphertexts, c' being the result of applying a *compressed encryption scheme* to

⋆ This work has received a French governmental support granted to the COMIN Labs excellence laboratory and managed by the National Research Agency in the "Investing for the Future" program under reference ANR-10-LABX-07-01.

⋆⋆ This work has been supported in part by the European Union's H2020 Programme under grant agreement number ICT-644209 and the French FUI project CRYPTOCOMP.

the plaintext m and c being recovered from c' using a *ciphertext decompression procedure*. In that approach obviously, the new encryption rate $|c'|/|m|$ becomes asymptotically close to 1 for long messages, which leaves no significant margin for improvement. However, the paradigm of ciphertext compression leaves totally open the question of how to choose E in a way that minimizes the decompression overhead, while preserving the same security level as originally intended.

Prior art. The cost of a homomorphic evaluation of several symmetric primitives has been investigated, including several optimized implementations of AES [GHS12,CCK+13,DHS14], and of the lightweight block ciphers Simon [LN14] and Prince [DSES14]. Usually very simple, lightweight block ciphers seem natural candidates for efficient evaluations in the encrypted domain. However, they may also lead to *much worse* performances than a homomorphic evaluation of, say, AES. Indeed, contemporary HE schemes use *noisy* ciphertexts, where a fresh ciphertext includes a noise component which grows along with homomorphic operations. Usually a homomorphic multiplication increases the noise by much larger proportions than a homomorphic addition. The maximum allowable level of noise (determined by the system parameters) then depends mostly on the multiplicative depth of the circuit. Many lightweight block ciphers balance out their simplicity by a large number of rounds, *e.g.* KATAN and KTANTAN [CDK09], with the effect of considerably increasing their multiplicative depth. This type of design is therefore prohibitive in a HE context. Still Prince appears to be a much more suitable block cipher for homomorphic evaluation than AES (and than Simon), because it specifically targets applications that require a low latency; it is designed to minimize the cost of an unrolled implementation [BCG+12] rather than being designed to optimize *e.g.* silicon area.

At Eurocrypt 2015, Albrecht, Rechberger, Schneider, Tiessen and Zohner observed that the usual criteria that rule the design of lightweight block ciphers are not appropriate when designing a symmetric encryption scheme with a low-cost homomorphic evaluation [ARS+15]. Indeed, both the number of rounds and the number of binary multiplications required to evaluate an Sbox have to be taken into account. Minimizing the number of rounds is a crucial issue for low-latency ciphers like Prince, while minimizing the number of multiplications is a requirement when designing a block cipher for efficient masked implementations (see *e.g.* [GLSV14]).

These two criteria have been considered together for the first time by Albrecht *et al.* in the recent design of a family of block ciphers called LowMC [ARS+15] with very small multiplicative size and depth[5]. However, the proposed instances of LowMC, namely LowMC-80 and LowMC-128, have recently had some security issues [DLMW15]. They actually present some weaknesses inherent in their low multiplicative complexity. Indeed, the algebraic normal forms (*i.e.*, the multivariate polynomials) describing the encryption and decryption functions are sparse and have a low degree. This type of features is usually exploited in algebraic attacks, cube attacks and their variants, *e.g.* [CP02,CM03,DS09,ADMS09]. While these attacks are rather general, the improved variant used for breaking LowMC [DLMW15], named interpolation attack [JK97], specifically applies to block ciphers. Indeed it exploits the sparse algebraic normal form of some intermediate bit within the cipher using that this bit can be evaluated both from the plaintext in the forward direction and from the ciphertext in the backward direction. This technique leads to several attacks including a key-recovery attack against LowMC-128 with time complexity 2^{118} and data complexity 2^{73}, implying that the cipher does not provide the expected 128-bit security level.

Our contributions. We emphasize that beyond the task of designing a HE-friendly block cipher, revisiting the whole compressed encryption scheme (in particular its internal mode of operation) is what is really needed in order to take these concrete HE-related implementation constraints into account.

First, we identify that homomorphic decompression is subject to an *offline phase* and an *online phase*. The offline phase is plaintext-independent and therefore can be performed in advance, whereas the online phase completes decompression upon reception of the plaintext-dependent part of the compressed ciphertext. Making the online phase as quick as technically doable leads us to choose **an additive IV-based stream cipher to implement** E. However, we note that the use of a lightweight block cipher as the building-block of that stream cipher usually provides a security level limited to $2^{n/2}$ where n is the block size [Rog11], thus limiting the number of encrypted blocks to (typically) less than 2^{32} (*i.e.* 32GB for 64-bit blocks).

[5] It is worth noting that in a HE context, reducing the multiplicative size of a symmetric primitive might not be the first concern (while it is critical in a multiparty computation context, which also motivated the work of Albrecht *et al.* [ARS+15]), whereas minimizing the multiplicative depth is of prime importance.

As a result, we propose our own candidate for E: the keystream generator Trivium [CP08], which belongs to the eSTREAM portfolio of recommended stream ciphers, and a new proposal called *Kreyvium*, which shares the same internal structure but allows for bigger keys of 128 bits[6]. The main advantage of Kreyvium over Trivium is that it provides 128-bit security (instead of 80-bit) with the same multiplicative depth, and inherits the same security arguments. It is worth noticing that the design of a variant of Trivium which guarantees a 128-bit security level has been raised as an open problem for the last ten years, see e.g. [Eni14, p. 30]. Beside a higher security level, it also accommodates longer IVs, so that it can encrypt up to $46 \cdot 2^{128}$ plaintext bits under the same key, with multiplicative depth only 12. Moreover, both Trivium and Kreyvium are resistant against the interpolation attacks used for breaking LowMC since these ciphers do not rely on a permutation which would enable the attacker to compute backwards.

We implemented our construction and instantiated it with Trivium, Kreyvium and LowMC in CTR-mode. Our results show that the promising performances attained by the HE-dedicated block cipher LowMC can be achieved with well-known primitives whose security has been firmly established for over a decade.

Organization of the paper. We introduce a general model and a generic construction to compress homomorphic ciphertexts in Sec. 2. Our construction using Trivium and Kreyvium is described in Sec. 3. Subsequent experimental results are presented in Sec. 4.[7]

2 A Generic Design for Efficient Decompression

In this section, we describe our model and generic construction to transmit compressed homomorphic ciphertexts between Alice and Charlie. We use the same notation as in the introduction: Alice wants to send some plaintext m, encrypted under Bob's public key pk (of an homomorphic encryption scheme HE) to a third party evaluator Charlie.

2.1 Offline/Online Phases in Ciphertext Decompression

Most practical scenarios would likely find it important to distinguish between three distinct phases within the homomorphic evaluation of $\mathcal{C}_{\mathsf{E}^{-1}}$:

1. an *offline key-setup* phase which only depends on Bob's public key and can be performed once and for all before Charlie starts receiving compressed ciphertexts encrypted under Bob's key;
2. an *offline decompression* phase which can be performed only based on some plaintext-independent material found in the compressed ciphertext;
3. an *online decompression* phase which aggregates the result of the offline phase with the plaintext-dependent part of the compressed ciphertext and (possibly very quickly) recovers the decompressed ciphertext c.

As such, our general-purpose formulation $c' = (\mathsf{HE}_{\mathsf{pk}}(k), \mathsf{E}_k(m))$ does not allow to make a clear distinction between these three phases. In our context, it is much more relevant to reformulate the encryption scheme as an IV-based encryption scheme where the encryption and decryption process are both deterministic but depend on an IV:

$$\mathsf{E}_k(m) \stackrel{\text{def}}{=} \left(IV, \mathsf{E}'_{k,IV}(m)\right) \ .$$

Since the IV has a limited length, it can be either transmitted during an offline preprocessing phase, or may alternately correspond to a state which is maintained by the server. Now, to minimize the latency

[6] Independently from our results, another variant of Trivium named Trivi-A has been proposed [CCHN15]. It handles larger keys but uses longer registers. It then needs more rounds for mixing the internal state, which means that it is much less adapted to our setting than Kreyvium.

[7] In App. D, we also present a second candidate for E that relies on a completely different technique based on the observation that multiplication in binary fields is $\mathbb{F}_2$-bilinear, making it possible to homomorphically exponentiate field elements with a log-log-depth circuit. We also report a random oracle based proof that compressed ciphertexts are semantically secure under an appropriate complexity assumption. We show, however, that this second approach remains disappointingly impractical.

of homomorphic decompression for Charlie, the online phase should be reduced to a minimum. The most appropriate choice in this respect consists in using an additive IV-based stream cipher Z so that

$$\mathsf{E}'_{k,IV}(m) = Z(k, IV) \oplus m \ .$$

In this reformulation, the decompression process is clearly divided into a offline precomputation stage which only depends on pk, k and IV, and an online phase which is plaintext-dependent. The online phase is thus reduced to a mere XOR between the plaintext-dependent part of the ciphertext $\mathsf{E}'_{k,IV}(m)$ and the HE-encrypted keystream $\mathsf{HE}(Z(k, IV))$, which comes essentially for free in terms of noise growth in HE ciphertexts. All expensive operations (*i.e.* homomorphic multiplications) are performed during the offline decompression phase where $\mathsf{HE}(Z(k, IV))$ is computed from $\mathsf{HE}(k)$ and IV.

2.2 Our Generic Construction

We devise a generic construction based on a homomorphic encryption scheme HE with plaintext space $\{0, 1\}$, an expansion function G mapping ℓ_{IV}-bit strings to strings of arbitrary size, and a fixed-size parametrized function F with input size ℓ_x, parameter size ℓ_k and output size N. The construction is depicted on Fig. 1.

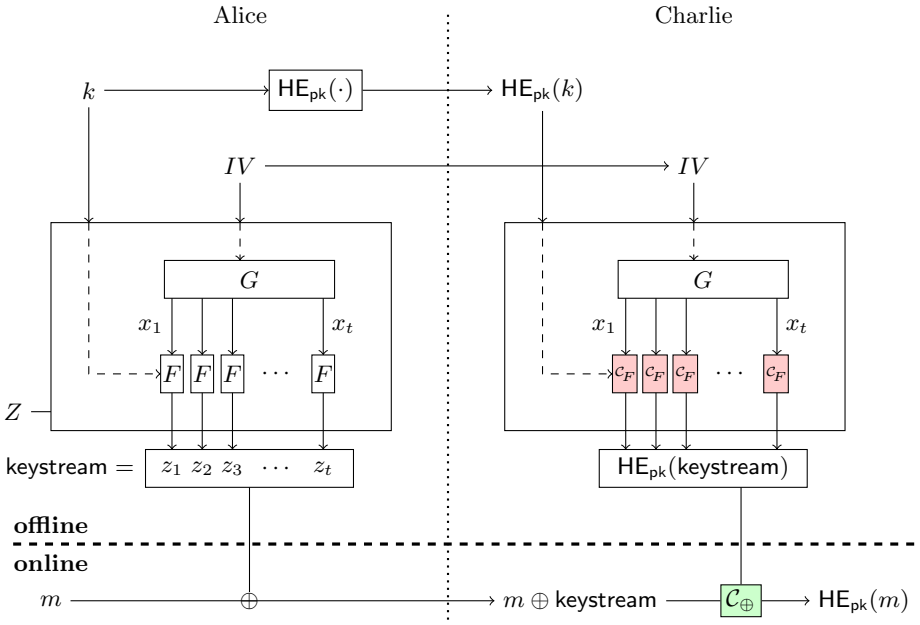


Fig. 1. Our generic construction. The multiplicative depth of the circuit is equal to the depth of $\mathcal{C}_F$. This will be the bottleneck in our protocol and we want the multiplicative depth of F to be as small as possible. With current HE schemes, the circuit $\mathcal{C}_\oplus$ is usually very fast (addition of ciphertexts) and has a negligible impact on the noise in the ciphertext.

Compressed encryption. Given an ℓ_m-bit plaintext m, Bob's public key pk and $IV \in \{0, 1\}^{\ell_{IV}}$, the compressed ciphertext c' is computed as follows:

1. Set $t = \lceil \ell_m / N \rceil$,
2. Set $(x_1, \ldots, x_t) = G(IV; t\ell_x)$,
3. Randomly pick $k \leftarrow \{0, 1\}^{\ell_k}$,
4. For $1 \leq i \leq t$, compute $z_i = F_k(x_i)$,
5. Set $\mathsf{keystream}$ to the ℓ_m leftmost bits of $z_1 \,||\, \ldots \,||\, z_t$,
6. Output $c' = (\mathsf{HE}_{\mathsf{pk}}(k), m \oplus \mathsf{keystream})$.

Ciphertext decompression. Given c' as above, Bob's public key pk and $IV \in \{0,1\}^{\ell_{IV}}$, the ciphertext decompression is performed as follows:

1. Set $t = \lceil \ell_m / N \rceil$,
2. Set $(x_1, \ldots, x_t) = G(IV; t\ell_x)$,
3. For $1 \leq i \leq t$, compute $\mathsf{HE}_{\mathsf{pk}}(z_i) = \mathcal{C}_F\left(\mathsf{HE}_{\mathsf{pk}}(k), x_i\right)$ with some circuit $\mathcal{C}_F$,
4. Deduce $\mathsf{HE}_{\mathsf{pk}}(\mathsf{keystream})$ from $\mathsf{HE}_{\mathsf{pk}}(z_1), \ldots, \mathsf{HE}_{\mathsf{pk}}(z_t)$,
5. Compute $c = \mathsf{HE}_{\mathsf{pk}}(m) = \mathcal{C}_{\oplus}\left(\mathsf{HE}_{\mathsf{pk}}(\mathsf{keystream}), m \oplus \mathsf{keystream}\right)$.

The circuit $\mathcal{C}_{\oplus}$ computes $\mathsf{HE}(a \oplus b)$ given $\mathsf{HE}(a)$ and b where a and b are bit-strings of the same size. In our construction, the cost of decompression per plaintext block is *fixed* and roughly equals one single evaluation of the circuit $\mathcal{C}_F$; most importantly, the multiplicative depth of the decompression circuit is also fixed, and set to the depth of $\mathcal{C}_F$.

How secure are compressed ciphertexts? From a high-level perspective, compressed homomorphic encryption is just hybrid encryption and relates to the generic KEM-DEM construct. However it just cannot inherit from the general security results attached to the KEM-DEM framework [AGKS05,HK07] since taking some HE scheme to implement the KEM part does not even fulfill the basic requirements that the KEM be IND-CCA or even IND-CCCA. It is usual that HE schemes succeed in achieving CPA security but often grossly fail to realize any form of CCA1 security, to the point of admitting simple key recovery attacks [CT15]. Therefore common KEM-DEM results just do not apply here.

On the other hand, CPA security is arguably strong enough for compressed homomorphic encryption, given that in practice Alice may always provide a signature $\sigma(c')$ together with c' to Charlie to ensure origin and data authenticity. Thus, the right level of security requirement on the compressed encryption scheme itself seems to be just IND-CPA for concrete use. However, it is not known what minimal security assumptions to require from a homomorphic KEM and a general-purpose DEM to yield a KEM-DEM scheme that is provably IND-CPA. As a result of that, evidence that CPA security is reached may only be provided on a case-by-case basis given a specific embodiment.

Instantiating the paradigm. The rest of the paper focuses on how to choose the expansion function G and function F so that the homomorphic evaluation of $\mathcal{C}_F$ is as fast (and its multiplicative depth as low) as possible. In our approach, the value of IV is assumed to be shared between Alice and Charlie and needs not be transmitted along with the compressed ciphertext. For instance, IV is chosen to be an absolute constant such as $IV = 0^{\ell}$ where $\ell = \ell_{IV} = \ell_x$. Another example is to take for $IV \in \{0,1\}^{\ell}$ a synchronized state that is updated between transmissions. Also, the expansion function G is chosen to implement a counter in the sense of the NIST description of the CTR mode [Nat01], for instance

$$G(IV; t\ell) = (IV, IV \boxplus 1, \ldots, IV \boxplus (t-1)) \quad \text{where} \quad a \boxplus b = (a+b) \bmod 2^{\ell} .$$

Finally, F is chosen to follow a specific design to ensure both an appropriate security level and a low multiplicative depth. We focus in Section 3 on the keystream generator corresponding to Trivium, and on a new variant, called *Kreyvium*.

Interestingly, the output of an iterated PRF used in counter mode is computationally indistinguishable from random [BDJR97, Th. 13]. Hence, under the assumption that Trivium or Kreyvium is a PRF[8], the keystream $z_1 \,||\, \ldots \,||\, z_t$ produced by our construction is also indistinguishable. However, this is insufficient to prove that the compressed encryption scheme is semantically secure (IND-CPA), because the adversary also sees $\mathsf{HE}_{\mathsf{pk}}(k)$ during the IND-CPA game, which cannot be proven not to make the keystream distinguishable. Although the security of this approach is empiric, Section 3 provides a strong rationale for the Kreyvium design and makes it the solution with the smallest homomorphic evaluation latency known so far.

Why not using a block cipher for F? Although not specifically in these terms, the use of lightweight block ciphers like PRINCE and SIMON has been proposed in the context of compressed homomorphic ciphertexts *e.g.* [LN14,DSES14]. However a complete encryption scheme based on the ciphers has not been defined. This is a major issue since the security provided by all classical modes of operation (including all variants

[8] Note that this equivalent to say that Kreyvium instantiated with a random key and mapping the IV's to the keystream is secure [BG07, Sec. 3.2].

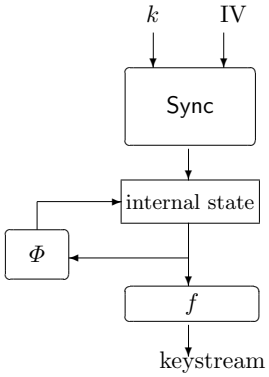
of CBC, CTR, CFB, OFB, OCB...) is inherently limited to $2^{n/2}$ where n is the block size [Rog11] (this is also emphasized in *e.g.* [KL14, p. 95]). Only a very few modes providing *beyond-birthday* security have been proposed [Iwa06,Yas11,LST12] but they induce a higher implementation cost and their security is usually upper-bounded by $2^{2n/3}$.

In other words, the use of a block cipher operating on 64-bit blocks like PRINCE or SIMON-32/64 implies that the number of blocks encrypted under the same key should be significantly less that 2^{32} (*i.e.* 32GB for 64-bit blocks). Therefore, only block ciphers with a large enough block size, like the LowMC instantiation with a 256-bit block proposed in [ARS+15], are suitable in applications which may require the encryption of more than 2^{32} bits under the same key.

3 Trivium and Kreyvium, Two Low-Depth Stream Ciphers

Since an additive stream cipher is the optimal choice, we now focus on keystream generation, and on its homomorphic evaluation. An IV-based keystream generator is decomposed into:

- a resynchronization function, Sync, which takes as input the IV and the key (possibly expanded by some precomputation phase), and outputs some n-bit initial state;
- a transition function Φ which computes the next state of the generator;
- a filtering function f which computes a keystream segment from the current internal state.



Since generating N keystream bits may require a circuit of depth up to

$$(\mathrm{depth}(\mathsf{Sync}) + N\,\mathrm{depth}(\Phi) + \mathrm{depth}(f))\,,$$

the best design strategy for minimizing this value consists in choosing a transition function with a small depth. The extreme option is to choose for Φ a linear function as in the CTR mode where the counter is implemented by an LFSR. An alternative strategy that we will investigate consists in choosing a nonlinear transition whose depth does not increase too fast when it is iterated. In App. B, the reader may find a discussion on the influence of Sync on the multiplicative depth of the circuit depending on which quantity should be encrypted under the HE scheme.

Size of the internal state. A major specificity of our context is that a large internal state can be easily handled. Indeed, in most classical stream ciphers, the internal-state size usually appears as a bottleneck because the overall size of the quantities to be stored highly influences the number of gates in the implementation. This is not the case in our context. It might seem, a priori, that increasing the size of the internal state automatically increases the number of nonlinear operations (because the number of inputs of Φ increases). But, this is not the case if a part of this larger internal state is used, for instance, for storing the secret key. This strategy can be used for increasing the security at no implementation cost. Indeed, the complexity of all generic attacks aiming at recovering the internal state of the generator is $\mathcal{O}(2^{n/2})$ where n is the size of the secret part of the internal state even if some part is not updated during the keystream generation. For instance, the time-memory-data-tradeoff attacks in [Bab95,Gol97,BS00] aim at inverting the function which maps the internal state of the generator to the first keystream bits. But precomputing some values of this function must be feasible by the attacker, which is not the case if the filtering or transition function depends on some secret material. On the other hand, the size n' of the non-constant secret part of the internal state determines the data complexity for finding a collision on the internal state: the length of the keystream produced from the same key is limited to $2^{n'/2}$. But, if the transition function or the filtering function depends on the IV, this limitation corresponds to the maximal keystream length produced from the same key/IV pair. It is worth noticing that many attacks require a very long keystream generated from the same key/IV pair and do not apply in our context since the keystream length is strictly limited by the multiplicative depth of the circuit.

3.1 Trivium in the HE setting

Trivium [CP08] is one of the seven stream ciphers recommended by the eSTREAM project after a 5-year international competition [ECR05]. Due to the small number of nonlinear operations in its transition function, it appears as a natural candidate in our context.

Description. Trivium is a synchronous stream cipher with a key and an IV of 80 bits each. Its internal state is composed of three registers of sizes 93, 84 and 111 bits, having an internal state size of 288 bits in total. Here, we use for the internal state the notation introduced by the designers: the leftmost bit of the 93-bit register is s_1, and its rightmost one is s_{93}; the leftmost bit of the register of size 84 is s_{94} and the rightmost s_{177}; the leftmost bit of register of size 111 is s_{178} and the rightmost s_{288}. The initialization and the generation of an N-bit Keystream are described below.

$(s_1, s_2, \ldots, s_{93}) \leftarrow (K_0, \ldots, K_{79}, 0, \ldots, 0)$
$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (IV_0, \ldots, IV_{79}, 0, \ldots, 0)$
$(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (0, \ldots, 0, 1, 1, 1)$
for $i = 1$ to $1152 + N$ **do**
  $t_1 \leftarrow s_{66} + s_{93}$
  $t_2 \leftarrow s_{162} + s_{177}$
  $t_3 \leftarrow s_{243} + s_{288}$
    if $i > 1152$ **do**
      output $z_{i-1152} \leftarrow t_1 + t_2 + t_3$
    end if
  $t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171}$
  $t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{264}$
  $t_3 \leftarrow t_3 + s_{286} \cdot s_{287} + s_{69}$
  $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
  $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
  $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$
end for

No attack better than an exhaustive key search is known so far on the full Trivium. It can therefore be considered as a secure cipher. The family of attacks that seems to provide the best result on round-reduced versions is the cube attack and its variants [DS09,ADMS09,FV13]. They recover some key bits (resp. provide a distinguisher on the keystream) if the number of initialization rounds is reduced to 799 (resp. 885) rounds out of 1152. The highest number of initialization rounds that can be attacked is 961: in this case, a distinguisher exists for a class of weak keys [KMN11].

Multiplicative depth. It is easy to see that the multiplicative depth grows quite slowly with the number of iterations. An important observation is that, in the internal state, only the first 80 bits in Register 1 (the keybits) are initially encrypted under the HE and that, as a consequence, performing hybrid clear and encrypted data calculations is possible (this is done by means of the following simple rules: $0 \cdot [x] = 0$, $1 \cdot [x] = [x]$, $0 + [x] = [x]$ and $1 + [x] = [1] + [x]$, where the square brackets denote encrypted bits and where in all but the latter case, a homomorphic operation is avoided which is specially desirable for multiplications). This optimization allows for instance to increase the number of bits which can be generated (after the 1152 blank rounds) at depth 12 from 42 to 57 (*i.e.*, a 35% increase). Then, the relevant quantity in our context is the multiplicative depth of the circuit which computes N keystream bits from the 80-bit key. The proof of the following proposition is given in the App. C.

Proposition 1. *In Trivium, the keystream length $N(d)$ which can be produced from the 80-bit key with a circuit of multiplicative depth d, $d \geq 4$, is given by*

$$N(d) = 282 \times \left\lfloor \frac{d}{3} \right\rfloor + \begin{cases} 81 & \text{if } d \equiv 0 \bmod 3 \\ 160 & \text{if } d \equiv 1 \bmod 3 \\ 269 & \text{if } d \equiv 2 \bmod 3 \end{cases}.$$

3.2 Kreyvium

Our first aim is to offer a variant of Trivium with 128-bit key and IV, without increasing the multiplicative depth of the corresponding circuit. Besides a higher security level, another advantage of this variant is that the number of possible IVs, and then the maximal length of data which can be encrypted under the same key, increases from $2^{80}N_{\mathsf{trivium}}(d)$ to $2^{128}N_{\mathsf{kreyvium}}(d)$. Increasing the key and IV-size in Trivium is a challenging task, mentioned as an open problem in [Eni14, p. 30] for instance. In particular, Maximov and Biryukov [MB07] pointed out that increasing the key-size in Trivium without any additional modification cannot be secure due to some attack with complexity less than 2^{128}. A first attempt in this direction has been made in [MB07] but the resulting cipher accommodates 80-bit IV only, and its multiplicative complexity is higher than in Trivium since the number of AND gates is multiplied by 2.

Description. Our proposal, Kreyvium, accommodates a key and an IV of 128 bits each. The only difference with the original Trivium is that we have added to the 288-bit internal state a 256-bit part corresponding to the secret key and the IV. This part of the state aims at making both the filtering and transition functions key- and IV-dependent. More precisely, these two functions f and Φ depend on the key bits and IV bits, through the successive outputs of two shift-registers K^* and IV^* initialized by the key and by the IV respectively. The internal state is then composed of five registers of sizes 93, 84, 111, 128 and 128 bits, having an internal state size of 544 bits in total, among which 416 become unknown to the attacker after initialization.

We will use the same notation as the description of Trivium, and for the additional registers we use the usual shift-register notation: the leftmost bit is denoted by K^*_{127} (or IV^*_{127}), and the rightmost bit (*i.e.*, the output) is denoted by K^*_0 (or IV^*_0). Each one of these two registers are rotated independently from the rest of the cipher. The generator is described below, and depicted on Fig. 2.

$(s_1, s_2, \ldots, s_{93}) \leftarrow (K_0, \ldots, K_{92})$
$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (IV_0, \ldots, IV_{83})$
$(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (IV_{84}, \ldots, IV_{127}, 1, \ldots, 1, 0)$
$(K^*_{127}, K^*_{126}, \ldots, K^*_0) \leftarrow (K_0, \ldots, K_{127})$
$(IV^*_{127}, IV^*_{126}, \ldots, IV^*_0) \leftarrow (IV_0, \ldots, IV_{127})$
for $i = 1$ to $1152 + N$ **do**
  $t_1 \leftarrow s_{66} + s_{93}$
  $t_2 \leftarrow s_{162} + s_{177}$
  $t_3 \leftarrow s_{243} + s_{288} + \boldsymbol{K^*_0}$
  if $i > 1152$ **do**
    output $z_{i-1152} \leftarrow t_1 + t_2 + t_3$
  end if
  $t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171} + \boldsymbol{IV^*_0}$
  $t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{264}$
  $t_3 \leftarrow t_3 + s_{286} \cdot s_{287} + s_{69}$
  $t_4 \leftarrow K^*_0$
  $t_5 \leftarrow IV^*_0$
  $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
  $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
  $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$
  $(K^*_{127}, K^*_{126}, \ldots, K^*_0) \leftarrow (t_4, K^*_{127}, \ldots, K^*_1)$
  $(IV^*_{127}, IV^*_{126}, \ldots, IV^*_0) \leftarrow (t_5, IV^*_{127}, \ldots, IV^*_1)$
end for

Related ciphers. KATAN [CDK09] is a lightweight block cipher with a lot in common with Trivium. It is composed of two registers, whose feedback functions are very sparse, and have a single nonlinear term. The key, instead of being used for initializing the state, is introduced by XORing two key information-bits per round to each feedback bit. The recently proposed stream cipher Sprout [AM15], inspired by Grain but with much smaller registers, also inserts the key in a similar way: instead of using the key for initializing the state, one key information-bit is XORed at each clock to the feedback function. We can see the parallelism between these two ciphers and our newly proposed variant. In particular, the previous security analysis on KATAN shows that this type of design does not introduce any clear

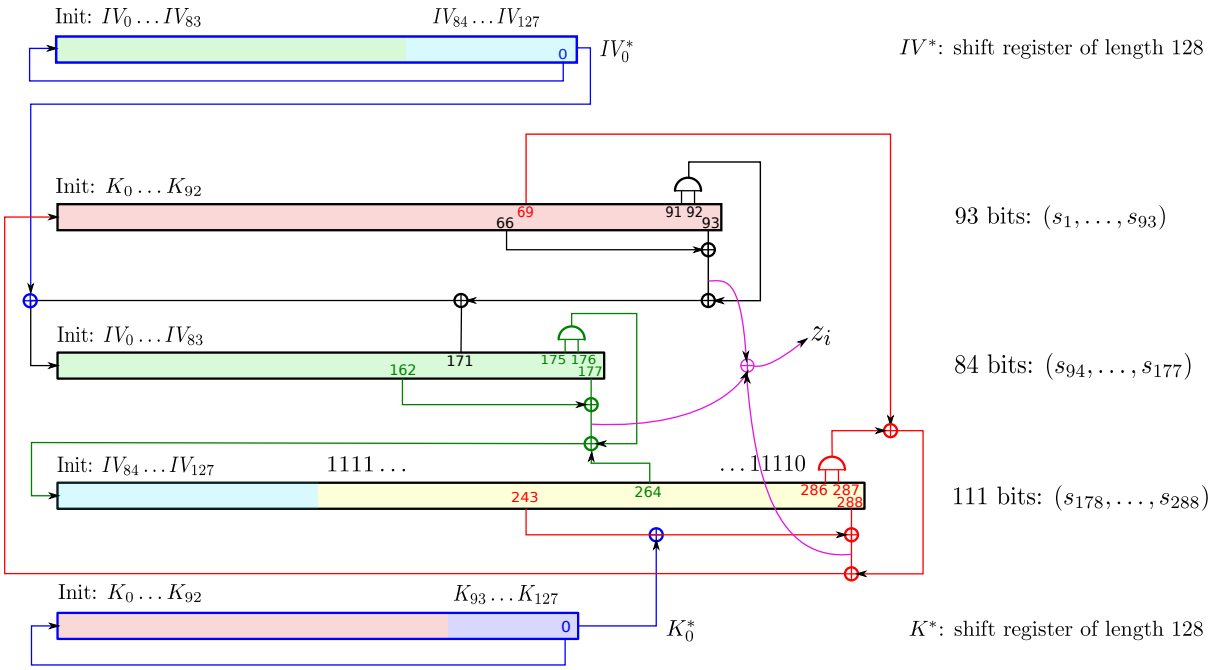


Fig. 2. Kreyvium. The three registers in the middle correspond to the original Trivium. The modifications defining Kreyvium correspond to the two registers in blue.

weakness. Indeed, the best attacks on round-reduced versions of KATAN so far [FM14] are meet-in-the-middle attacks, that exploit the knowledge of the values of the first and the last internal states (due to the block-cipher setting). As this is not the case here, such attacks, as well as the recent interpolation attacks against LowMC [DLMW15], do not apply. The best attacks against KATAN, when excluding MitM techniques, are conditional differential attacks [KMN10,KMN11].

Design rationale. In Kreyvium, we have decided to XOR the keybit K_0^* to the feedback function of the register that interacts with the content of $(s_1, \dots, s_{63})$ the later, since $(s_1, \dots, s_{63})$ is initialized with some key bits. The same goes for the IV^* register. Moreover, as the keybits that start entering the state are the ones that were not in the initial state, all the keybits affect the state at the earliest.

We also decided to initialize the state with some keybits and with all the IV bits, and not with a constant value, as this way the mixing will be performed quicker. Then we can expect that the internal-state bits after initialization are expressed as more complex and less sparse functions in the key and IV bits.

Our change of constant is motivated by the conditional differential attacks from [KMN11]: the conditions needed for a successful attack are that 106 bits from the IV or the key are equal to '0' and a single one needs to be '1'. This suggests that values set to zero "encourage" non-random behaviors, leading to our new constant. In other words, in Trivium, an all-zero internal state is always updated in an all-zero state, while an all-one state will change through time. The 0 at the end of the constant is added for preventing slide attacks.

Multiplicative depth. Exactly as for Trivium, we can compute the number of keystream bits which can be generated from the key at a given depth. The only difference with Trivium is that the first register now contains 93 key bits instead of 80. For this reason, the optimization using hybrid plaintext/ciphertext calculations is a bit less interesting: for any fixed depth $d \geq 4$, we can generate 11 bits less than with Trivium.

Proposition 2. *In Kreyvium, the keystream length $N(d)$ which can be produced from the 128-bit key with a circuit of multiplicative depth d, $d \geq 4$, is given by*

$$N(d) = 282 \times \left\lfloor \frac{d}{3} \right\rfloor + \begin{cases} 70 & \text{if } d \equiv 0 \bmod 3 \\ 149 & \text{if } d \equiv 1 \bmod 3 \\ 258 & \text{if } d \equiv 2 \bmod 3 \end{cases}.$$

Security analysis. We investigate in more detail how all the known attacks on Trivium, and some other techniques, can apply to Kreyvium.

TMDTO. TMDTO attacks aiming at recovering the initial state of the cipher do not apply since the size of the secret part of the internal state (416 bits) is much larger than twice the key-size. As discussed at the beginning of Section 3, the size of the whole secret internal state has to be taken into account, even if the additional 128-bit part corresponding to K^* is independent from the rest of the state. On the other hand, TMDTO aiming at recovering the key have complexity larger than exhaustive key search (even without any restriction on the precomputation time) since the key and the IV have the same size [HS05,CLP05].

Internal-state collision. As discussed in Section 3, a distinguisher may be built if the attacker is able to find two colliding internal states, since the two keystream sequences produced from colliding states are identical. Finding such a collision requires around 2^{144} keystream bits generated from the same key/IV pair, which is much longer than the maximal keystream length allowed by the multiplicative depth of the circuit. But, for a given key, two internal states colliding on all bits except on IV^* lead to two keystreams which have the same first 69 bits since IV^* affects the keystream only 69 clocks later. Moreover, if the difference between the two values of IV^* when the rest of the state collides lies in the leftmost bit, then this difference will affect the keystream bits $(69 + 128) = 197$ clocks later. This implies that, within around 2^{144} keystream bits generated from the same key, we can find two identical runs of 197 consecutive bits which are equal. However, this property does not provide a valid distinguisher because a random sequence of length 2^{144} blocks is expected to contain much more collisions on 197-bit runs. Therefore, the birthday-bound of 2^{144} bits provides a limit on the number of bits produced from the same key/IV pair, not on the bits produced from the same IV.

Cube attacks [DS09,FV13] and cube testers [ADMS09]. As previously pointed out, they provide the best attacks for round-reduced Trivium. In our case, as we keep the same main function, but we have two additional XORs per round, thus a better mixing of the variables, we can expect the relations to get more involved and hamper the application of previously defined round-reduced distinguishers. One might wonder if the fact that more variables are involved could ease the attacker's task, but we point out here that the limitation in the previous attacks was not the IV size, but the size of the cubes themselves. Therefore, having more variables available is of no help with respect to this point. We can conclude that the resistance of Kreyvium to these types of attacks is at least the resistance of Trivium, and even better.

Conditional differential cryptanalysis. Because of its applicability to both Trivium and KATAN, the attack from [KMN11] is definitely of interest in our case. In particular, the highest number of blank rounds is reached if some conditions on two registers are satisfied at the same time (and not only conditions on the register controlled by the IV bits in the original Trivium). In our case, as we have IV bits in two registers, it is important to elucidate whether an attacker can take advantage of introducing differences in two registers simultaneously. First, let us recall that we have changed the constant to one containing mostly 1. We previously saw that the conditions that favor the attacks are values set to zero in the initial state. In Trivium, per design, we have $(108+4+13) = 125$ bits already fixed to zero in the initial state, 3 are fixed to one and the others can be controlled by the attacker in the weak-key setting (and the attacker will force them to be zero most of the time). Now, instead, we have 64 bits forced to be 1, 1 equal to zero, and $(128+93) = 221$ bits of the initial state controlled by the attacker in the weak-key setting, plus potentially 21 additional bits from the key still not used, that will be inserted during the first rounds. We can conclude that, while in Trivium is possible in the weak-key setting, to introduce zeros in the whole initial state but in 3 bits, in Kreyvium, we will never be able to set to zero 64 bits, implying that applying the techniques from [KMN11] becomes much harder. Additionally, as in the discussion on cube attacks, we can also hope here that we get more involved relations that will provide a better resistance against these attacks.

Algebraic attacks. Several algebraic attacks have been proposed against Trivium, aiming at recovering the 288-bit internal state at the beginning of the keystream generation (i.e., at time $t = 1153$) from the knowledge of the keystream bits. The most efficient attack of this type is due to Maximov and Biryukov [MB07]. It exploits the fact that the 22 keystream bits at time $3t'$, $0 \leq t' < 22$, are determined

by all bits of the initial state at indexes divisible by 3 (starting from the leftmost bit in each register). Moreover, once all bits at positions $3i$ are known, then guessing that the outputs of the three AND gates at time $3t'$ are zero provides 3 linear relations between the bits of the internal state and the keystream bits. The attack then consists of an exhaustive search for some bits at indexes divisible by 3. The other bits in such positions are then deduced by solving the linear system derived from the keystream bits at positions $3t'$. Once all these bits have been determined, the other 192 bits of the initial state are deduced from the other keystream equations. This process must be iterated until the guess for the outputs of the AND gates is correct. In the case of Trivium, the outputs of at least 125 AND gates must be guessed in order to get 192 linear relations involving the 192 bits at indexes $3i+1$ and $3i+2$. This implies that the attack has to be repeated $(4/3)^{125} = 2^{52}$ times. From these guesses, we get many linear relations involving the bits at positions $3i$ only, implying that only an exhaustive search with complexity 2^{32} for the other bits at positions $3i$ is needed. Therefore, the overall complexity of the attack is around $2^{32} \times 2^{52} = 2^{84}$. A similar algorithm can be applied to Kreyvium, but the main difference is that every linear equation corresponding to a keystream bit also involves one key bit. Moreover, the key bits involved in the generation of any 128 consecutive output bits are independent. It follows that each of the first 128 linear equations introduces a new unknown in the system to solve. For this reason, it is not possible to determine all bits at positions $3i$ by an exhaustive search on less than 96 bits like for Trivium. Moreover, the outputs of more than 135 AND gates must be guessed for obtaining enough equations on the remaining bits of the initial state. Therefore the overall complexity of the attack exceeds $2^{96} \times 2^{52} = 2^{148}$ and is much higher that the cost of the exhaustive key search. It is worth noticing that the attack would have been more efficient if only the feedback bits, and not the keystream bits, would have been dependent on the key. In this case, 22 linear relations independent from the key would have been available to the attacker.

4 Experimental Results

In this section, we discuss and compare the practicality of our generic construction when instantiated with Trivium, Kreyvium and the HE-dedicated cipher LowMC. The expansion function G implements a mere counter, and the aforementioned algorithms are used to instantiate the function F that produces N bits of keystream per iteration—*cf.* Prop. 1 and 2.[9]

HE framework. In our experiments, we considered two HE schemes: the BGV scheme [BGV14] and the FV scheme [FV12] (a scale-invariant version of BGV). The BGV scheme is implemented in the library HElib [HS14] and has become *de facto* a standard benchmarking library for HE applications. Similarly, the FV scheme was previously used in several HE benchmarkings [FSF+13,LN14,CDS15], is conceptually simpler than the BGV scheme, and is one of the most efficient HE schemes.[10] Additionally, batching was used [SV14], *i.e.* the HE schemes were set up to encrypt vectors in an SIMD fashion (component-wise operations, and rotations via the Frobenius endomorphism). The number of elements that can be encrypted depends on the number of terms in the factorization modulo 2 of the cyclotomic polynomial used in the implementation. This batching allowed us to perform several Trivium/Kreyvium/LowMC in parallel in order to increase the throughput.

Parameter selection for subsequent homomorphic processing. In all the previous works on the homomorphic evaluation of symmetric encryption schemes, the parameters of the underlying HE scheme were selected for the exact multiplicative depth required and not beyond [GHS12,CLT14,LN14,DSES14,ARS+15]. This means that once the ciphertext is decompressed, no further homomorphic computation can actually be performed by Charlie – this makes the claimed timings considerably less meaningful in a real-world context.

In this work, we benchmarked both parameters for the exact multiplicative depth and parameters able to handle circuits of the minimal multiplicative depth plus 7 to allow further homomorphic processing

[9] Note that these propositions only hold when hybrid clear and encrypted data calculations are possible between IV and HE ciphertexts. This explains the slight differences in the number of keystream bits per iteration (column "N") between Tab. 1 and 2.

[10] In our experiments, we used the Armadillo compiler implementation of FV [CDS15]. This source-to-source compiler turns a C++ algorithm into a Boolean circuit, optimizes it, and generates an OpenMP parallel code which can then be combined with a HE scheme.

Table 1. Latency and throughput for the algorithms using HElib on a single core of a mid-end 48-core server (4 x AMD Opteron 6172 processors with 64GB of RAM).

Algorithm	security level κ	N	used × depth	#slots	latency sec.	throughput bits/min
Trivium-12	80	45	12	600	1417.4	1143.0
			19	720	4420.3	439.8
Trivium-13	80	136	13	600	3650.3	1341.3
			20	720	11379.7	516.3
Kreyvium-12	128	42	12	504	1715.0	740.5
			19	756	4956.0	384.4
Kreyvium-13	128	124	13	682	3987.2	1272.6
			20	480	12450.8	286.8
LowMC-128	$? \leq 118$	256	13	682	3608.4	2903.1
			20	480	10619.6	694.3
LowMC-128 [ARS+15]	$? \leq 118$	256	13	682	3368.8	3109.6
			20	480	9977.1	739.0

by Charlie (which is obviously what is expected in applications of homomorphic encryption). We chose 7 because, in practice, numerous applications use algorithms of multiplicative depth smaller than 7 (see *e.g.* [GLN12,LLN14]). In what follows we compare the results we obtain using Trivium, Kreyvium and also the LowMC cipher. For LowMC, we benchmarked not only our own implementation but also the LowMC implementation of [ARS+15] available at `https://bitbucket.org/malb/lowmc-helib`. Minor changes to this implementation were made in order to obtain an equivalent parametrization of HElib. The main difference between the latter implementations is that the implementation from [ARS+15] uses an optimized method for multiplying a Boolean vector and a Boolean matrix, namely the "Method of Four Russians". This explains why our implementation is approximately 6% slower, as it performs 2–3 times more ciphertext additions.

Experimental results using HElib. For sake of comparison with [ARS+15], we ran our implementations and their implementation of LowMC on a single core using HElib. The results are provided in Tab. 1. We recall that the latency refers to the time required to perform the entire homomorphic evaluation whereas the throughput is the number of blocks processed per time unit.

Experimental results using FV. On Tab. 2, we present the benchmarks when using the FV scheme. The experiments were performed using either a single core (in order to compare with BGV) or on all the cores of the machine the tests were performed on. The execution time acceleration factor between 48-core parallel and sequential executions is given in the column "Speed gain". While good accelerations (at least 25 times) were obtained for Trivium and Kreyvium algorithms, the acceleration when using LowMC is significantly smaller ($\sim$ 10 times). This is due to the huge number of operations in LowMC that created memory contention and huge slowdown in memory allocation.

Interpretation. First, we would like to recall that LowMC-128 must be considered in a different category because of the existence of a key-recovery attack with time complexity 2^{118} and data complexity 2^{73} [DLMW15]. However, it has been included in the table in order to show that the performances achieved by Trivium and Kreyvium are of the same order of magnitude. An increase in the number of rounds of LowMC-128 (typically by 4 rounds) is needed in order to achieve 128-bit security, but this would have a non-negligible impact on its homomorphic evaluation performance, as it would require to increase the depth of the cryptosystem supporting the execution. For instance, a back-of-the-envelope estimation for four additional rounds leads to a degradation of its homomorphic execution performances by a factor of about 2 to 3 (more computations with larger parameters), making the approach in this paper much more competitive.

Table 2. Latency of our construction when using the FV scheme on a mid-end 48-core server (4 x AMD Opteron 6172 processors with 64GB of RAM).

Algorithm	security level κ	N	used $\times$ depth	latency (sec.) 1 core	latency (sec.) 48 cores	Speed gain
Trivium-12	80	57	12	681.5	26.8	$\times$ 25.4
			19	2097.1	67.6	$\times$ 31.0
Trivium-13	80	136	13	888.2	33.9	$\times$ 26.2
			20	2395.0	77.2	$\times$ 31.0
Kreyvium-12	128	46	12	904.4	35.3	$\times$ 25.6
			19	2806.3	82.4	$\times$ 34.1
Kreyvium-13	128	125	13	1318.6	49.7	$\times$ 26.5
			20	3331.4	97.9	$\times$ 34.0
LowMC-128	$? \leq 118$	256	14	1531.1	171.0	$\times$ 9.0
			21	3347.8	329.0	$\times$ 10.2

It is worth noticing that the minimal multiplicative depth for which valid LowMC output ciphertexts were obtained was 14 for the FV scheme and 13 for the BGV scheme (the theoretical multiplicative depth is 12 but the high number of additions in LowMC explains this difference[11]).

Our results show that Trivium and Kreyvium have a smaller latency than LowMC, but have a slightly smaller throughput. As already emphasized in [LN14], real-world applications of homomorphic encryption (which are often cloud-based applications) should be implemented in a transparent and user-friendly way. In the context of our approach, the latency of the offline phase is still an important parameter aiming at an acceptable experience for the end-user even when a sufficient amount of homomorphic keystream could not be precomputed early enough because of overall system dimensioning issues.

Also Trivium and Kreyvium are more parallelizable than LowMC is. Therefore, our work shows that the promising performances obtained by the recently proposed *HE-dedicated cipher* LowMC can also be achieved with Trivium, a well-analyzed stream cipher, and a variant aiming at achieving 128 bits of security. Last but not least, we recall that our construction was aiming at compressing the size of transmissions between Alice and Charlie. We support an encryption rate $|c'|/|m|$ that becomes asymptotically close to 1 for long messages, *e.g.* for $\ell_m = 1$GB message length, our construction instantiated with Trivium (resp. Kreyvium), yields an expansion rate of 1.08 (resp. 1.16).

5 Conclusion

Our work shows that the promising performances obtained by the recently proposed HE-dedicated cipher LowMC can also be achieved with Trivium, a well-known primitive whose security has been thoroughly analyzed, *e.g.* [MB07,DS09], and [ADMS09,FV13,KMN11]. The 10-year analysis effort from the whole community, initiated by the eSTREAM competition, enables us to gain confidence in its security. Also our variant Kreyvium, with a 128-bit security, benefits from the same analysis since the core of the cipher is essentially the same.

From a more fundamental perspective, one may wonder how many multiplicative levels are *strictly necessary* to achieve a secure compressed encryption scheme, irrespective of any performance metric such as the number of homomorphic bit multiplications to perform in the decompression circuit. We already know that a multiplicative depth of $\lceil \log \kappa \rceil + 1$ is achievable for κ-bit security (*cf.* App. D). Can one do better or prove that this is a lower bound?

[11] We would like to emphasize that the multiplicative depth is only an *approximation* of the homomorphic depth required to absorb the noise generated by the execution of a given algorithm [LP13]. This approximation neglects the noise induced by additions and thus does not hold for too addition-intensive algorithms such as those in the LowMC family.

However, the provable security of a KEM-DEM construct where the KEM is homomorphic remains an open question. In particular, assuming the KEM part is just IND-CPA, what would be the minimum security requirements expected from the DEM part to yield an IND-CPA construction?

References

ADMS09. Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. In *FSE*, volume 5665 of *LNCS*, pages 1–22. Springer, 2009.

AGKS05. Masayuki Abe, Rosario Gennaro, Kaoru Kurosawa, and Victor Shoup. Tag-KEM/DEM: A New Framework for Hybrid Encryption and A New Analysis of Kurosawa-Desmedt KEM. In *EUROCRYPT*, volume 3494 of *LNCS*, pages 128–146. Springer, 2005.

AM15. Frederik Armknecht and Vasily Mikhalev. On Lightweight Stream Ciphers with Shorter Internal States. In *FSE*, volume 9054 of *LNCS*, pages 451–470. Springer, 2015.

AMOR14. Gora Adj, Alfred Menezes, Thomaz Oliveira, and Francisco Rodríguez-Henríquez. Computing Discrete Logarithms in $\mathbb{F}\_3^{6*137}$ using Magma. *IACR Cryptology ePrint Archive*, 2014:57, 2014.

ARS+15. Martin Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *EUROCRYPT*, volume 9056 of *LNCS*, pages 430–454. Springer, 2015.

Bab95. Steve Babbage. A space/time trade-off in exhaustive search attacks on stream ciphers. In *European Convention on Security and Detection*, number 408. IEEE Conference Publication, 1995.

BCG+12. Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications. In *ASIACRYPT*, volume 7658 of *LNCS*, pages 208–225. Springer, 2012.

BDJR97. Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A Concrete Security Treatment of Symmetric Encryption. In *FOCS*, pages 394–403. IEEE Computer Society, 1997.

BG07. Côme Berbain and Henri Gilbert. On the Security of IV Dependent Stream Ciphers. In *FSE*, volume 4593 of *LNCS*, pages 254–273. Springer, 2007.

BGJT14. Razvan Barbulescu, Pierrick Gaudry, Antoine Joux, and Emmanuel Thomé. A Heuristic Quasi-Polynomial Algorithm for Discrete Logarithm in Finite Fields of Small Characteristic. In *EUROCRYPT*, volume 8441 of *LNCS*, pages 1–16. Springer, 2014.

BGV14. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption without Bootstrapping. *TOCT*, 6(3):13, 2014.

Bod07. Marco Bodrato. Towards Optimal Toom-Cook Multiplication for Univariate and Multivariate Polynomials in Characteristic 2 and 0. In *WAIFI*, volume 4547 of *LNCS*, pages 116–133. Springer, 2007.

BS00. Alex Biryukov and Adi Shamir. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In *ASIACRYPT*, volume 1976 of *LNCS*, pages 1–13. Springer, 2000.

CCHN15. Avik Chakraborti, Anupam Chattopadhyay, Muhammad Hassan, and Mridul Nandi. TriviA: A fast and secure authenticated encryption scheme. In *CHES*, volume 9293 of *Lecture Notes in Computer Science*, pages 330–353. Springer, 2015.

CCK+13. Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi, and Aaram Yun. Batch Fully Homomorphic Encryption over the Integers. In *EUROCRYPT*, volume 7881 of *LNCS*, pages 315–335. Springer, 2013.

CDK09. Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In *CHES*, volume 5747 of *LNCS*, pages 272–288. Springer, 2009.

CDS15. Sergiu Carpov, Paul Dubrulle, and Renaud Sirdey. Armadillo: a compilation chain for privacy preserving applications. In *ACM CCSW*, 2015.

CLP05. Christophe De Cannière, Joseph Lano, and Bart Preneel. Comments on the rediscovery of time memory data tradeoffs. Technical report, eSTREAM - ECRYPT Stream Cipher Project, 2005.

CLT14. Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Scale-Invariant Fully Homomorphic Encryption over the Integers. In *PKC*, volume 8383 of *LNCS*, pages 311–328. Springer, 2014.

CM03. Nicolas Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. In *EUROCRYPT*, volume 2656 of *LNCS*, pages 345–359. Springer, 2003.

CP02. Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In *ASIACRYPT*, volume 2501 of *LNCS*, pages 267–287. Springer, 2002.

CP08. Christophe De Cannière and Bart Preneel. Trivium. In *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *LNCS*, pages 244–266. Springer, 2008.

CT15. Massimo Chenal and Qiang Tang. On Key Recovery Attacks Against Existing Somewhat Homomorphic Encryption Schemes. In *LATINCRYPT*, volume 8895 of *LNCS*, pages 239–258. Springer, 2015.

DHS14. Yarkin Doröz, Yin Hu, and Berk Sunar. Homomorphic AES Evaluation using NTRU. *IACR Cryptology ePrint Archive*, 2014:39, 2014.

DLMW15. Itai Dinur, Yunwen Liu, Willi Meier, and Qingju Wang. Optimized Interpolation Attacks on LowMC. *IACR Cryptology ePrint Archive*, 2015:418, 2015.

DS09. Itai Dinur and Adi Shamir. Cube Attacks on Tweakable Black Box Polynomials. In *EUROCRYPT*, volume 5479 of *LNCS*, pages 278–299. Springer, 2009.

DSES14. Yarkin Doröz, Aria Shahverdi, Thomas Eisenbarth, and Berk Sunar. Toward Practical Homomorphic Evaluation of Block Ciphers Using Prince. In *WAHC*, volume 8438 of *LNCS*, pages 208–220. Springer, 2014.

ECR05. ECRYPT - European Network of Excellence in Cryptology. The eSTREAM Stream Cipher Project. http://www.ecrypt.eu.org/stream/, 2005.

Eni14. Algorithms, key size and parameters report 2014. Technical report, ENISA - European Union Agency for Network and Information Security, 2014.

FM14. Thomas Fuhr and Brice Minaud. Match Box Meet-in-the-Middle Attack against KATAN. In *FSE*, volume 8540 of *LNCS*, pages 61–81. Springer, 2014.

FSF+13. Simon Fau, Renaud Sirdey, Caroline Fontaine, Carlos Aguilar, and Guy Gogniat. Towards practical program execution over fully homomorphic encryption schemes. In *IEEE International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 284–290, 2013.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

FV13. Pierre-Alain Fouque and Thomas Vannet. Improving Key Recovery to 784 and 799 Rounds of Trivium Using Optimized Cube Attacks. In *FSE*, volume 8424 of *LNCS*, pages 502–517. Springer, 2013.

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178. ACM, 2009.

GHS12. Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic Evaluation of the AES Circuit. In *CRYPTO*, volume 7417 of *LNCS*, pages 850–867. Springer, 2012.

GKZ14. Robert Granger, Thorsten Kleinjung, and Jens Zumbrägel. Breaking '128-bit Secure' Supersingular Binary Curves - (Or How to Solve Discrete Logarithms in $\mathbb{F}_{2^{4\cdot 1223}}$ and $\mathbb{F}_{2^{12\cdot 367}}$). In *CRYPTO, Part II*, volume 8617 of *LNCS*, pages 126–145. Springer, 2014.

GLN12. Thore Graepel, Kristin E. Lauter, and Michael Naehrig. ML Confidential: Machine Learning on Encrypted Data. In *ICISC*, volume 7839 of *LNCS*, pages 1–21. Springer, 2012.

GLSV14. Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations. In *FSE*, volume 8540 of *LNCS*, pages 18–37. Springer, 2014.

Gol97. Jovan Dj. Golic. Cryptanalysis of alleged A5 stream cipher. In *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 239–255. Springer-Verlag, 1997.

HK07. Dennis Hofheinz and Eike Kiltz. Secure Hybrid Encryption from Weakened Key Encapsulation. In *CRYPTO*, volume 4622 of *LNCS*, pages 553–571. Springer, 2007.

HS05. Jin Hong and Palash Sarkar. New Applications of Time Memory Data Tradeoffs. In *ASIACRYPT*, volume 3788 of *LNCS*, pages 353–372. Springer, 2005.

HS14. Shai Halevi and Victor Shoup. Algorithms in HElib. In *CRYPTO, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 554–571, 2014.

Iwa06. Tetsu Iwata. New Blockcipher Modes of Operation with Beyond the Birthday Bound Security. In *FSE*, volume 4047 of *LNCS*, pages 310–327. Springer, 2006.

JK97. Thomas Jakobsen and Lars R. Knudsen. The interpolation attack on block ciphers. In *FSE*, volume 1267 of *LNCS*, pages 28–40. Springer, 1997.

JP14. Antoine Joux and Cécile Pierrot. Improving the Polynomial time Precomputation of Frobenius Representation Discrete Logarithm Algorithms - Simplified Setting for Small Characteristic Finite Fields. In *ASIACRYPT, Part I*, volume 8873 of *LNCS*, pages 378–397. Springer, 2014.

KL14. Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. Chapman and Hall/CRC Press, 2014.

KMN10. Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional Differential Cryptanalysis of NLFSR-Based Cryptosystems. In *ASIACRYPT*, volume 6477 of *LNCS*, pages 130–145. Springer, 2010.

KMN11. Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional Differential Cryptanalysis of Trivium and KATAN. In *SAC*, volume 7118 of *LNCS*, pages 200–212. Springer, 2011.

LLN14. Kristin Lauter, Adriana López-Alt, and Michael Naehrig. Private Computation on Encrypted Genomic Data. In *LATINCRYPT*, LNCS, 2014.

LN14. Tancrède Lepoint and Michael Naehrig. A Comparison of the Homomorphic Encryption Schemes FV and YASHE. In *AFRICACRYPT*, volume 8469 of *LNCS*, pages 318–335. Springer, 2014.

LP13. Tancrède Lepoint and Pascal Paillier. On the Minimal Number of Bootstrappings in Homomorphic Circuits. In *WAHC*, volume 7862 of *LNCS*, pages 189–200. Springer, 2013.

LST12. Will Landecker, Thomas Shrimpton, and R. Seth Terashima. Tweakable Blockciphers with Beyond Birthday-Bound Security. In *CRYPTO*, volume 7417 of *LNCS*, pages 14–30. Springer, 2012.

MB07. Alexander Maximov and Alex Biryukov. Two Trivial Attacks on Trivium. In *SAC*, volume 4876, pages 36–55. Springer, 2007.

Nat01. National Institute of Standards and Technology. NIST Special Publication 800-38A — Recommendation for Block Cipher Modes of Operation. NIST Special Publication 800-38A, 2001.

NLV11. Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *ACM CCSW*, pages 113–124. ACM, 2011.

Pin89. Antonio Pincin. A new algorithm for multiplication in finite fields. *IEEE Transactions on Computers*, 38(7):1045–1049, 1989.

Rog11. Phillip Rogaway. Evaluation of some blockcipher modes of operation. Cryptrec, 2011.

SV14. Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Des. Codes Cryptography*, 71(1):57–81, 2014.

Yas11. Kan Yasuda. A New Variant of PMAC: Beyond the Birthday Bound. In *CRYPTO*, volume 6841 of *LNCS*, pages 596–609. Springer, 2011.

A Number of AND and XOR gates in Trivium and Kreyvium

A more thorough analysis of the number of AND and XOR gates in the different circuits is provided in Table 3. The keystream length is the maximum possible for a given multiplicative depth. It is lower for the BGV scheme (batched) because the IV is no more a boolean string so less circuit optimization are possible. For the FV scheme (non-batched) the table gives the number of executed gates in the worst case. The actual number of executed gates can be lower as it depends on the employed IV.

Table 3. Number of AND and XOR gates to homomorphically evaluate in Trivium and Kreyvium for FV and BGV schemes.

Algorithm	λ	FV			BGV		
		#ANDs	#XORs	keystream	#ANDs	#XORs	keystream
Trivium-12	80	3237	15019	57	3183	14728	45
Trivium-13	80	3474	16537	136	3474	16537	136
Kreyvium-12	128	3311	18081	46	3288	17934	42
Kreyvium-13	128	3564	19878	125	3561	19866	124

B Which quantity must be encrypted under the HE?

In order to limit the multiplicative depth of the decryption circuit, we may prefer to transmit a longer secret $\tilde{k}$, from which more calculations can be done at a small multiplicative depth. Typically, for a block cipher, the sequence formed by all round-keys can be transmitted to the server. In this case, the key scheduling does not have to be taken into account in the homomorphic evaluation of the decryption function. Similarly, stream ciphers offer several such trade-offs between the encryption rate and the encryption throughput. The encryption rate, *i.e.*, the ratio between the size of $c' = (\mathsf{HE}_{\mathsf{pk}}(k), \mathsf{E}_k(m))$ and the plaintext size ℓ_m, is defined as

$$\rho = \frac{|c'|}{\ell_m} = \frac{|E_k(m)|}{\ell_m} + \frac{|\tilde{k}| \times (\text{HE expansion rate})}{\ell_m} .$$

The extremal situation obviously corresponds to the case where the message encrypted under the homomorphic scheme is sent directly, *i.e.*, $c' = \mathsf{HE}_{\mathsf{pk}}(m)$. The multiplicative depth here is 0, as no decryption needs to be performed. In this case, ρ corresponds to the HE expansion rate.

The following alternative scenarios can then be compared.

1. Only the secret key is encrypted under the homomorphic scheme, *i.e.*, $\tilde{k} = k$. Then, since we focus on symmetric encryption schemes with rate 1, we get

$$\rho = 1 + \frac{\ell_k \times (\text{HE expansion rate})}{\ell_m}$$

which is the smallest encryption rate we can achieve for an ℓ_k-bit security. In a nonce-based stream cipher, ℓ_m is limited by the IV size ℓ_{IV} and by the maximal keystream length $N(d)$ which can be produced for a fixed multiplicative depth $d \geq \text{depth}(\mathsf{Sync}) + \text{depth}(f)$. Then, the minimal encryption rate is achieved for messages of any length $\ell_m \leq 2^{\ell_{IV}} N(d)$.

2. An intermediate case consists in transmitting the initial state of the generator, *i.e.*, the output of Sync. Then, the number of bits to be encrypted by the HE increases to the size n of the internal state, while the number of keystream bits which can be generated from a given initial state with a circuit of depth d corresponds to $N(d + \text{depth}(\mathsf{Sync}))$. Then, we get

$$\rho = 1 + \frac{n \times (\text{HE expansion rate})}{N(d + \text{depth}(\mathsf{Sync}))} \ ,$$

for any message length. The size of the internal state is at least twice the size of the key. Therefore, this scenario is not interesting, unless the number of plaintext bits ℓ_m to be encrypted under the same key is smaller than twice $N(d + \text{depth}(\mathsf{Sync}))$.

C Proofs of Propositions 1 and 2

We first observe that, within any register in Trivium, the degree of the leftmost bit is greater than or equal to the degrees of the other bits in the register. It is then sufficient to study the evolution of the leftmost bits in the three registers. Let $t_i(d)$ denotes the first time instant (starting from $t = 1$) where the leftmost bit in Register i is computed by a circuit of depth d. The depth of the feedback bit in Register i can increase from d to $(d+1)$ if either a bit of depth $(d+1)$ reaches a XOR gate in the feedback function, or a bit of depth d reaches one of the inputs of the AND gate. From the distance between the leftmost bit and the first bit involved in the feedback (resp. and the first entry of the AND gate) in each register, we derive that

$$\begin{aligned}
t_1(d+1) &= \min(t_3(d+1) + 66, t_3(d) + 109) \\
t_2(d+1) &= \min(t_1(d+1) + 66, t_1(d) + 91) \\
t_3(d+1) &= \min(t_2(d+1) + 69, t_2(d) + 82)
\end{aligned}$$

In Trivium, the first key bits K_{78} and K_{79} enter the AND gate in Register 1 at time $t = 13$ (starting from $t = 1$), implying $t_2(1) = 14$. Then, $t_3(1) = 83$ and $t_1(1) = 149$. This leads to

$$t_1(4) = 401, t_2(4) = 296 \text{ and } t_3(4) = 335 \ .$$

From $d = 3$, the differences $t_i[d+1] - t_i[d]$ are large enough so that the minimum in the three recurrence relation corresponds to the right-hand term. We then deduce that, for $d \geq 4$,

– if $d \equiv 1 \bmod 3$,

$$t_1(d) = 282 \times \frac{(d-1)}{3} + 119, \ t_2(d) = 282 \times \frac{(d-1)}{3} + 14, \ t_3(d) = 282 \times \frac{(d-1)}{3} + 53.$$

– if $d \equiv 2 \bmod 3$,

$$t_1(d) = 282 \times \frac{(d-2)}{3} + 162, \ t_2(d) = 282 \times \frac{(d-2)}{3} + 210, \ t_3(d) = 282 \times \frac{(d-2)}{3} + 96.$$

– if $d \equiv 0 \bmod 3$,

$$t_1(d) = 282 \times \frac{(d-3)}{3} + 205, \ t_2(d) = 282 \times \frac{(d-3)}{3} + 253, \ t_3(d) = 282 \times \frac{(d-3)}{3} + 292.$$

The degree of the keystream produced at time t corresponds to the minimum between the degrees of the bit at position 66 in Register 1, the bit at position 69 in Register 2 and the bit at position 66 in Register 3. Then, for $d > 3$,

$$N(d) = \min(t_1(d+1) + 64, t_2(d+1) + 67, t_3(d+1) + 64) \ .$$

This leads to, for any $d \geq 4$,

$$N(d) = 282 \times \left\lfloor \frac{d}{3} \right\rfloor + \begin{cases} 81 & \text{if } d \equiv 0 \bmod 3 \\ 160 & \text{if } d \equiv 1 \bmod 3 \\ 269 & \text{if } d \equiv 2 \bmod 3 \end{cases} .$$

In Kreyvium, the recurrence relations defining the $t_i(d)$ are the same. The only difference is that the first key bits now enter the AND gate in Register 1 at time $t = 1$, implying $t_2(1) = 2$. Then, $t_3(1) = 71$, $t_1(1) = 137$ and $t_3[2] = 85$. The situation is then similar to Trivium, except that we start from

$$t_1(4) = 390, t_2(4) = 285 \text{ and } t_3(4) = 324 \ .$$

These three values are equal to the values obtained with Trivium minus 11. This fixed difference then propagated, leading to, for any $d \geq 4$,

- if $d \equiv 1 \bmod 3$,

$$t_1(d) = 282 \times \frac{(d-1)}{3} + 108, \ t_2(d) = 282 \times \frac{(d-1)}{3} + 3, \ t_3(d) = 282 \times \frac{(d-1)}{3} + 42.$$

- if $d \equiv 2 \bmod 3$,

$$t_1(d) = 282 \times \frac{(d-2)}{3} + 151, \ t_2(d) = 282 \times \frac{(d-2)}{3} + 199, \ t_3(d) = 282 \times \frac{(d-2)}{3} + 85.$$

- if $d \equiv 0 \bmod 3$,

$$t_1(d) = 282 \times \frac{(d-3)}{3} + 194, \ t_2(d) = 282 \times \frac{(d-3)}{3} + 242, \ t_3(d) = 282 \times \frac{(d-3)}{3} + 281.$$

We eventually derive that, for Kreyvium, for any $d \geq 4$,

$$N(d) = 282 \times \left\lfloor \frac{d}{3} \right\rfloor + \begin{cases} 70 & \text{if } d \equiv 0 \bmod 3 \\ 149 & \text{if } d \equiv 1 \bmod 3 \\ 258 & \text{if } d \equiv 2 \bmod 3 \end{cases} .$$

D Another Approach: Using Discrete Logs on Binary Fields

We now introduce a second, discrete-log based embodiment of the generic compressed encryption scheme of Section 2.2. We recall that the homomorphic encryption scheme $\mathsf{HE}_{\mathsf{pk}}(\cdot)$ is assumed to encrypt separately each plaintext bit. For $h \in \mathbb{F}_{2^n}$, we identify h with the vector of its coefficients and therefore by $\mathsf{HE}_{\mathsf{pk}}(h)$, we mean the vector composed of the encrypted coefficients of h.

This approach attempts to achieve provable security while ensuring a low-depth circuit $\mathcal{C}_F$. For this, we require G to be a PRNG and IV to be chosen at random at encryption time and transmitted within c'. This allows us to prove that c' is semantically secure under a well-defined complexity assumption. Simultaneously, we use exponentiation in a binary field to instantiate F, which yields a circuit $\mathcal{C}_F$ of depth $\lceil \log \ell_k \rceil$. Performance estimations, however, show that Approach 2 is rather impractical.

D.1 Description

In this approach, the operating mode picks a fresh $IV \leftarrow \{0,1\}^{\ell_{IV}}$ for each compressed ciphertext. The expansion function G is instantiated by some PRNG that we will view as a random oracle in the security proof. Also, we set

$$\ell_x = N = n \ ,$$

and therefore F maps n-bit inputs to n-bit outputs under ℓ_k-bit parameters. Given $k \in \{0,1\}^{\ell_k}$ and $x \in \{0,1\}^n$, $F_k(x)$ views x as a field element in $\mathbb{F}_{2^n}$ and k as an ℓ_k-bit integer, computes $z = x^k$ over $\mathbb{F}_{2^n}$, views z as an n-bit string and outputs z. This completes the description of the compressed encryption scheme.

D.2 A log-log-depth exponentiation circuit over $\mathbb{F}_{2^n}$

We describe a circuit $\mathcal{C}_{\exp}$ which, given a field element $h \in \mathbb{F}_{2^n}$ and an encrypted exponent $\mathsf{HE}_{\mathsf{pk}}(k)$ with $k \in \{0,1\}^{\ell_k}$, computes $\mathsf{HE}_{\mathsf{pk}}(h^k)$ and has multiplicative depth at most $\lceil \log \ell_k \rceil$.

Stricto sensu, $\mathcal{C}_{\exp}$ is not just a Boolean circuit evaluated homomorphically, as it combines computations in the clear, homomorphic $\mathbb{F}_2$-arithmetic on encrypted bits, and $\mathbb{F}_2$-arithmetic on mixed cleartext/encrypted bits.

$\mathcal{C}_{\exp}$ uses implicitly some irreducible polynomial p to represent $\mathbb{F}_{2^n}$ and we denote by $\oplus$ and $\otimes_p$ the field operators. The basic idea here is that for any $a, b \in \mathbb{F}_{2^n}$, computing $\mathsf{HE}(a \otimes_p b)$ from $\mathsf{HE}(a), \mathsf{HE}(b)$ requires *only* 1 *multiplicative level*, simply because $\otimes_p$ is $\mathbb{F}_2$-bilinear. Therefore, knowing p and the characteristics of HE, we can efficiently implement a bilinear operator on encrypted binary vectors to compute

$$\mathsf{HE}(a \otimes_p b) = \mathsf{HE}(a) \ \otimes_p^{\mathsf{HE}} \ \mathsf{HE}(b) \ .$$

A second useful observation is that for any $a \in \mathbb{F}_{2^n}$ and $\beta \in \{0,1\}$, there is a multiplication-free way to deduce $\mathsf{HE}(a^\beta)$ from a and $\mathsf{HE}(\beta)$. When $\beta = 1$, a^β is just a and $a^\beta = 1_{\mathbb{F}_{2^n}} = (1, 0, \ldots, 0)$ otherwise. Therefore to construct a vector $v = (v_0, \ldots, v_{n-1}) = \mathsf{HE}(a^\beta)$, it is enough to set

$$v_i := \begin{cases} \mathsf{HE}(0) & \text{if } a_i = 0 \\ \mathsf{HE}(\beta) & \text{if } a_i = 1 \end{cases}$$

for $i = 1, \ldots, n-1$ and

$$v_0 := \begin{cases} \mathsf{HE}(\beta \oplus 1) & \text{if } a_0 = 0 \\ \mathsf{HE}(1) & \text{if } a_i = 1 \end{cases}$$

where it does not matter that the same encryption of 0 be used multiple times. Let us denote this procedure as

$$\mathsf{HE}(a^\beta) = \mathsf{L}_a\left(\mathsf{HE}(\beta)\right) \ .$$

Now, given as input $h \in \mathbb{F}_{2^n}$, $\mathcal{C}_{\exp}$ first computes in the clear $h_i = h^{2^i}$ for $i = 0, \ldots, \ell_k - 1$. Since

$$h^k = h_0^{k_0} \otimes_p h_1^{k_1} \otimes_p \cdots \otimes_p h_{\ell_k-1}^{k_{\ell_k-1}} \ ,$$

one gets

$$\begin{aligned} \mathsf{HE}(h^k) &= \mathsf{HE}\left(h_0^{k_0}\right) \ \otimes_p^{\mathsf{HE}} \ \mathsf{HE}\left(h_1^{k_1}\right) \ \otimes_p^{\mathsf{HE}} \ \cdots \ \otimes_p^{\mathsf{HE}} \ \mathsf{HE}\left(h_{\ell_k-1}^{k_{\ell_k-1}}\right) \\ &= \mathsf{L}_{h_0}\left(\mathsf{HE}\left(k_0\right)\right) \ \otimes_p^{\mathsf{HE}} \ \mathsf{L}_{h_1}\left(\mathsf{HE}\left(k_1\right)\right) \ \otimes_p^{\mathsf{HE}} \ \cdots \ \otimes_p^{\mathsf{HE}} \ \mathsf{L}_{h_{\ell_k-1}}\left(\mathsf{HE}\left(k_{\ell_k-1}\right)\right) \ . \end{aligned}$$

Viewing the ℓ_k variables as the leaves of a binary tree, $\mathcal{C}_{\exp}$ therefore requires at most $\lceil \log \ell_k \rceil$ levels of homomorphic multiplications to compute and return $\mathsf{HE}_{\mathsf{pk}}(h^k)$.

D.3 Security Results

Given some homomorphic encryption scheme HE and security parameters κ, n, ℓ_k, we define a family of decision problems $\{\mathsf{DP}_t\}_{t>0}$ as follows.

Definition 1 (Decision Problem DP_t). *Let $\mathsf{pk} \leftarrow \mathsf{HE.KeyGen}(1^\kappa)$ be a random public key, $k \leftarrow \{0,1\}^{\ell_k}$ a random ℓ_k-bit integer and $g_1, \ldots, g_t, g'_1, \ldots, g'_t \leftarrow \mathbb{F}_{2^n}$, $2t$ random field elements. Distinguish the distributions*

$$\begin{aligned} D_{t,1} &= \left(\mathsf{pk}, \mathsf{HE}_{\mathsf{pk}}(k), g_1, \ldots, g_t, g_1^k, \ldots, g_t^k\right) \quad \text{and} \\ D_{t,0} &= \left(\mathsf{pk}, \mathsf{HE}_{\mathsf{pk}}(k), g_1, \ldots, g_t, g'_1, \ldots, g'_t\right) \ . \end{aligned}$$

Theorem 1. *Viewing G as a random oracle, the compressed encryption scheme described above is semantically secure (IND-CPA), unless breaking DP_t is efficient, for messages of bit-size ℓ_m with $(t-1)n < \ell_m \leq tn$.*

Proof (Sketch). A random-oracle version of the PRNG function G is an oracle that takes as input a pair (IV, ℓ) where $IV \in \{0,1\}^{\ell_{IV}}$ and $\ell \in \mathbb{N}^*$, and returns an ℓ-bit random string. It is also imposed to the oracle that $G(IV;\ell_1)$ be a prefix of $G(IV;\ell_2)$ for any IV and $\ell_1 \leq \ell_2$.

We rely on the real-or-random flavor of the IND-CPA security game and build a reduction algorithm $\mathcal{R}$ that uses an adversary $\mathcal{A}^G$ against the scheme to break DP_t as follows. $\mathcal{R}$ is given as input some $(\mathsf{pk}, \mathsf{HE}_{\mathsf{pk}}(k), g_1, \ldots, g_t, \tilde{g}_1, \ldots, \tilde{g}_t)$ sampled from $D_{t,b}$ and has to guess the bit b. $\mathcal{R}$ runs $\mathcal{A}^G(\mathsf{pk})$ and receives some challenge plaintext $m^\star \in \{0,1\}^{\ell_m}$ where $(t-1)n < \ell_m \leq tn$. $\mathcal{R}$ makes use of its input to build a compressed ciphertext c' as follows:

1. Set $\mathsf{keystream}$ to the ℓ_m leftmost bits of $\tilde{g}_1 \,||\, \ldots \,||\, \tilde{g}_t$,
2. Pick a random $IV^\star \leftarrow \{0,1\}^{\ell_{IV}}$,
3. Abort if $G(IV^\star;\ell')$ is already defined for some ℓ',
4. Set $G(IV^\star;tn)$ to $g_1 \,||\, \ldots \,||\, g_t$.
5. Set $c' = (\mathsf{HE}_{\mathsf{pk}}(k), IV^\star, m^\star \oplus \mathsf{keystream})$,

$\mathcal{R}$ then returns c' to $\mathcal{A}$ and forwards $\mathcal{A}$'s guess $\hat{b}$ to its own challenger. At any moment, $\mathcal{R}$ responds to $\mathcal{A}$'s queries to G using fresh random strings for each new query or to extend a past query to a larger size. Obviously, all the statistical distributions comply with their specifications. Consequently c' is an encryption of $m^\star$ if the input instance comes from $D_{t,1}$ and is an encryption of some perfectly uniform plaintext if the instance follows $D_{t,0}$. The reduction is tight as long as the abortion probability $q2^{-\ell_{IV}}$ remains negligible, q being the number of oracle queries made by $\mathcal{A}$. □

Interestingly, we note the following fact about our family of decision problems.

Theorem 2. *For any $t \geq 2$, DP_t is equivalent to DP_2.*

Proof. Obviously, a problem instance $(\mathsf{pk}, \mathsf{HE}_{\mathsf{pk}}(k), g_1, \ldots, g_t, \tilde{g}_1, \ldots, \tilde{g}_t)$ sampled from $D_{t,b}$ can be converted into an instance of $D_{2,b}$ for the same b, by just removing $g_3, \ldots, g_t$ and $\tilde{g}_3, \ldots, \tilde{g}_t$. This operation preserves the distributions of all inner variables. Therefore DP_t can be reduced to DP_2. Now, we describe a reduction $\mathcal{R}$ which, given an instance $(\mathsf{pk}, \mathsf{HE}_{\mathsf{pk}}(k), g_1, g_2, \tilde{g}_1, \tilde{g}_2)$ sampled from $D_{2,b}$, makes use of an adversary $\mathcal{A}$ against DP_t to successfully guess b. $\mathcal{R}$ converts its instance of $D_{2,b}$ into an instance of $D_{t,b}$ as follows. For $i = 3, \ldots, t$, $\mathcal{R}$ randomly selects $\alpha_i \leftarrow \mathbb{Z}/(2^n-1)\mathbb{Z}$ and sets

$$g_i = g_1^{\alpha_i} g_2^{1-\alpha_i} \;, \quad \tilde{g}_i = \tilde{g}_1^{\alpha_i} \tilde{g}_2^{1-\alpha_i} \;.$$

It is easily seen that, if $\tilde{g}_1 = g_1^k$ and $\tilde{g}_2 = g_2^k$ then $\tilde{g}_i = g_i^k$ for every i. If however $\tilde{g}_1, \tilde{g}_2$ are uniformly and independently distributed over $\mathbb{F}_{2^n}$ then so are $\tilde{g}_3, \ldots, \tilde{g}_t$. Our reduction runs $\mathcal{A}$ over that instance and outputs the guess $\hat{b}$ returned by $\mathcal{A}$. Obviously $\mathcal{R}$ is tight. □

Overall, the security of our compressed encryption scheme relies on breaking DP_1 for messages of bit-size at most n and on breaking DP_2 for larger messages. Beyond the fact that DP_2 reduces to DP_1, we note that these two problems are unlikely to be equivalent since DP_2 is easily broken using a DDH oracle over $\mathbb{F}_{2^n}$ while DP_1 seems to remain unaffected by it.

D.4 Performance Issues

Concrete security parameters. Note that our decisional security assumptions $\mathsf{DP}_t^{\exp}$ for all $t \geq 1$ reduce to the discrete logarithm computation in the finite field $\mathbb{F}_{2^n}$. Solving discrete logarithm in finite fields of small characteristics is currently a very active research area, marked notably by the quasi-polynomial algorithm of Barbulescu, Gaudry, Joux and Thomé [BGJT14]. In particular, the expected security one can hope for has been recently completely redefined [GKZ14,AMOR14]. In our setting, we will select a prime n so that computing discrete logarithms in $\mathbb{F}_{2^n}$ has complexity 2^κ for κ-bit security. The first step of Barbulescu *et al.* algorithm runs in polynomial time. This step has been extensively studied and its complexity has been brought down to $\mathcal{O}((2^{\log_2 n})^6)$ using a very complex and tight analysis by Joux and Pierrot [JP14]. As for the quasi-polynomial step of the algorithm, its complexity can be upper-bounded, but in practice numerous trade-off can be used and it is difficult to give to lower bound it [BGJT14,AMOR14]. To remains conservative in our choice of parameters, we will base our security on the first step. To ensure a 80-bit (resp. 128-bit) security level, one should therefore choose a prime n of $\log_2 n \approx 14$ bits (resp. 23 bits), *i.e.* work in a finite field of about $16,000$ elements (resp. 4 million elements).

How impractical is this approach? We now briefly see why our discrete-log based construction on binary fields is impractical. We focus more specifically on the exponentiation circuit $\mathcal{C}_{\text{exp}}$ whose most critical subroutine is a general-purpose field multiplication in the encrypted domain. Taking homomorphic bit multiplication as the complexity unit and neglecting everything else, how fast can we expect to multiply encrypted field elements in $\mathbb{F}_{2^n}$?

When working in the cleartext domain, several families of techniques exist with attractive asymptotic complexities for large n, such as algorithms derived from Toom-Cook [Bod07] or Schönhage-Strassen [Pin89]. It is unclear how these different strategies can be adapted to our case and with what complexities[12]. However, let us optimistically assume that they *could be adapted somehow* and that one of these adaptations would just take n homomorphic bit multiplications.

A straightforward implementation of $\mathcal{C}_{\text{exp}}$ consists in viewing all circuit inputs $\mathsf{L}_{h_i}(\mathsf{HE}(k_i))$ as generic encrypted field elements and in performing generic field multiplications along the binary tree, which would require $\ell_k \cdot n$ homomorphic bit multiplications. Taking $\ell_k = 160$, $n = 16000$ and 0.5 seconds for each bit multiplication (as a rough estimate of the timings of Section 4), this accounts for more than 14 days of computation.

This can be improved because the circuit inputs are precisely not generic encrypted field elements; each one of the n ciphertexts in $\mathsf{L}_{h_i}(\mathsf{HE}(k_i))$ is known to equal either $\mathsf{HE}(k_i)$, $\mathsf{HE}(k_i \oplus 1)$, $\mathsf{HE}(0)$ or $\mathsf{HE}(1)$. Similarly, a circuit variable of depth 1 *i.e.*

$$\mathsf{L}_{h_i}(\mathsf{HE}(k_i)) \otimes_p^{\mathsf{HE}} \mathsf{L}_{h_{i+1}}(\mathsf{HE}(k_{i+1})) \ ,$$

contains n ciphertexts that are all an encryption of one of the 16 quadratic polynomials $ak_ik_{i+1} + bk_i + ck_{i+1} + d$ for $a, b, c, d \in \{0, 1\}$. This leads us to a strategy where one *simulates* the τ first levels of field multiplications at once, by computing the $2^{\lceil \log \ell_k \rceil - \tau}$ dictionaries of the form

$$\left\{ \mathsf{HE}\left(k_i^{b_0} k_{i+1}^{b_1} \cdots k_{i+2^\tau-1}^{b_{2^\tau-1}} \right) \right\}_{b_0,\ldots,b_{2^\tau-1} \in \{0,1\}}$$

and computing the binary coefficients (in clear) to be used to reconstruct each bit of the $2^{\lceil \log \ell_k \rceil - \tau}$ intermediate variables of depth τ from the dictionaries through linear (homomorphic) combinations. By assumption, this accounts for nothing in the total computation time. The rest of the binary tree is then performed using generic encrypted field multiplications as before, until the circuit output is fully aggregated. This approach is always more efficient than the straightforward implementation and optimal when the total number

$$\left(2^{2^\tau} - 2^\tau - 1\right) \cdot 2^{\lceil \log \ell_k \rceil - \tau} + \left(2^{\lceil \log \ell_k \rceil - \tau - 1} - 1\right) \cdot n$$

of required homomorphic bit multiplications is minimal. With $\ell_k = 160$ and $n = 16000$ again, the best choice is for $\tau = 4$. Assuming 0.5 seconds for each bit multiplication, this still gives a prohibitive 6.71 days of computation for a single evaluation of $\mathcal{C}_{\text{exp}}$.

[12] One could expect these techniques to become the most efficient ones here since their prohibitive overhead would disappear in the context of homomorphic circuits.

Block Ciphers that are Easier to Mask: How Far Can we Go?

Benoît Gérard[1,2], Vincent Grosso[1], María Naya-Plasencia[3], François-Xavier Standaert[1]

[1] ICTEAM/ELEN/Crypto Group, Université catholique de Louvain, Belgium.
[2] Direction Générale de l'Armement–Maîtrise de l'information, France. [3] INRIA Paris-Rocquencourt, France.

Abstract. The design and analysis of lightweight block ciphers has been a very active research area over the last couple of years, with many innovative proposals trying to optimize different performance figures. However, since these block ciphers are dedicated to low-cost embedded devices, their implementation is also a typical target for side-channel adversaries. As preventing such attacks with countermeasures usually implies significant performance overheads, a natural open problem is to propose new algorithms for which physical security is considered as an optimization criteria, hence allowing better performances again. We tackle this problem by studying how much we can tweak standard block ciphers such as the AES Rijndael in order to allow efficient masking (that is one of the most frequently considered solutions to improve security against side-channel attacks). For this purpose, we first investigate alternative S-boxes and round structures. We show that both approaches can be used separately in order to limit the total number of non-linear operations in the block cipher, hence allowing more efficient masking. We then combine these ideas into a concrete instance of block cipher called `Zorro`. We further provide a detailed security analysis of this new cipher taking its design specificities into account, leading us to exploit innovative techniques borrowed from hash function cryptanalysis (that are sometimes of independent interest). Eventually, we conclude the paper by evaluating the efficiency of masked `Zorro` implementations in an 8-bit microcontroller, and exhibit their interesting performance figures.

1 Introduction

Masking (aka secret sharing) is a widespread countermeasure against side-channel attacks (SCA) [28]. It essentially consists in randomizing the internal state of a device in such a way that the observation of few (say d) intermediate values during a cryptographic computation will not provide any information about any of the secret (aka sensitive) variables. This property is known as the "d-th order SCA security" and was formalized by Coron et al. as follows [16]: *A masked implementation is d-th order secure if every d-tuple of the intermediate values it computes is independent of any sensitive variable.* Reaching higher-order security is a theoretically sound approach for preventing SCAs, as it ensures that any adversary targeting the masked implementation will have to "combine" the information from at least $d+1$ intermediate computations. More precisely, if one can guarantee that the leakage samples corresponding to the manipulation of the different shares of a masking scheme are independent, then a higher-order security implies that an adversary will have to estimate the $d+1$-th moment of the leakage distribution (conditioned on a sensitive variable), leading to an exponential increase of the SCA data complexity [15][1]. In practice though, this exponential security increase only becomes meaningful if combined with a sufficient amount of noise in the side-channel leakage samples [58]. Also, the condition of independent leakage for the shares may turn out to be difficult to fulfill because of physical artifacts, e.g. glitches occurring in integrated circuits [39]. Yet, and despite these constraints, masking has proven to be one of the most satisfying solutions to improve security against SCAs, especially in the context of protected software implementations in smart cards [45, 53, 54, 56].

In general, the most difficult computations to mask are the ones that are non-linear over the group operation used to share the sensitive variables (e.g. the S-boxes in a block cipher). Asymptotically, the time complexity of masking such non-linear operations grows at least quadratically with the order d. As a result, a variety of research works have focused on specializing masking to certain algorithms (most frequently the AES Rijndael, see e.g. [14, 44]), in order to reduce its implementation overheads. More recently, the opposite approach has been undertaken by Piret et al. [47]. In a paper presented at ACNS 2012, the authors suggested that improved SCA security could be achieved at a lower implementation cost by specializing a block cipher for efficient masking. For this purpose, they started from the provably secure scheme proposed by Rivain and Prouff at CHES 2010 (see Appendix A.1), and specified a design allowing better performances than

[1] In certain scenarios, e.g. in a software implementation where all the shares are manipulated at different time instants, masking may also increase the time complexity of the attacks, as an adversary will have to test all the pairs, triples, ... of samples to extract information from a 2nd, 3rd, ... secure implementation.

the AES Rijndael as the order of the masking increases. More precisely, the authors first observed that bijective S-boxes that are at the same time easy to mask and have good properties for resisting standard cryptanalysis (e.g. linear [40], differential [5], algebraic [17]) are remarkably close to the AES S-box. As a result, they investigated the gains obtained with non-bijective S-boxes and described a Feistel network with a Substitution-Permutation Network (SPN) based round function taking advantage of this S-box. One interesting feature of this approach is that its impact on the performances of block cipher implementations will grow with the the physical security level (informally measured with the order d). That is, it enables performance gains that become more significant as we move towards physically secure implementations.

In this paper, we complement this first piece of work and further investigate design principles that could be exploited to improve the security of block ciphers implementations against SCAs thanks to the masking countermeasure. In particular, we investigate two important directions left open by Piret et al. First, we observe that non-bijective S-boxes usually lead to simple non-profiled attacks (as their output directly gives rise to "meaningful leakage models" [59]). As recently shown by Whitnall et al., we even have a proof that generic (non-profiled) SCAs against bijective S-boxes cannot exist [61]. This naturally gives a strong incentive to consider bijective S-boxes in block ciphers that are purposed for masked implementations. Hence, we analyze the possibility to trade a bit of the classical S-box properties (linearity, differential profile, algebraic degree) for bijectivity and more efficient masking. Second, we observe that the previous work from ACNS 2012 focused on the S-box design in order to allow efficient masking. This is a natural first step as it constitutes the only non-linear element of most block ciphers. Yet, it is also appealing to investigate whether the algorithm structure could not be modified in order to limit the total number of S-boxes executed during an encryption. We investigate this possibility and suggest that irregular designs in which only a part of the state goes through an S-box in each round can be used for this purpose, if the diffusion layer is adapted to this setting.

Our results show that each of the principles we propose (i.e. the modified S-box and structure) can be used to reduce the total number of non-linear operations in an AES-like block cipher - yet with a stronger impact of the second one. We then describe a new block cipher for efficient masking, that combines these two ideas in order to further reduce the total complexity corresponding to non-linear operations in the cipher. We call this cipher `Zorro` in reference to the masked fictional character. We further provide a detailed security evaluation of our proposal, considering state-of-the-art and dedicated cryptanalysis, in order to determine the number of rounds needed to obtain a secure cipher. Because of the irregular structure of `Zorro`, this analysis borrows recent tools from hash function cryptanalysis and describes new techniques for providing security bounds (e.g. against linear and differential cryptanalysis). We conclude with performance evaluations exhibiting that `Zorro` already leads to interesting performance gains for small security orders $d = 1, 2, 3$.

2 Bijective S-boxes that are easier to mask

In this section we aim at finding an 8-bit S-box having both a small masking cost and good cryptographic properties regarding the criteria presented in Appendix A.2. For this purpose, we will use the number of field multiplications and amount of randomness needed to execute a shared S-box as performance metrics. As discussed in Appendix A.3, reducing this number directly leads to more efficient Boolean masking using the state-of-the-art scheme of Rivain and Prouff [54]. Interestingly, it is also beneficial for more advanced (polynomial) masking schemes inspired from the multiparty computation literature, such as proposed by Prouff and Roche [50]. So our proposal is generally suitable for two important categories of masking schemes that (provably) generalize to high security orders. For reference, we first recall that the AES S-box consists in the composition of an inversion of the element in the field $GF(2^8)$ and an affine transformation A: $S_{AES} : x \mapsto A(x^{-1})$. Starting from this standard example, a natural objective would be to find an S-box that can be masked with a lower cost than the AES one (i.e. an S-box that can be computed using less than 4 multiplications [54]), and with similar security properties (i.e. a maximum of the differential spectrum close to 4, a maximum of the Walsh spectrum close to 32, and a high algebraic degree). Since there are $2^8!$ permutations over $GF(2^8)$, an exhaustive analysis of all these S-boxes is computationally unfeasible. Hence, we propose two different approaches to cover various S-boxes in our analysis. First, we exhaustively consider the S-boxes having a sparse polynomial representation (essentially one or two non-zero coefficients). Next, we investigate some proposals for constructing 8-bit S-boxes from a combination of smaller ones. In particular, we consider a number of solutions of low-cost S-boxes that have been previously proposed in the literature.

2.1 Exhaustive search among sparse polynomials

Monomials in $GF(2^8)$. First notice that in $GF(2^8)$ the square function is linear. Hence, we can define an equivalence relation between exponents: $e_1 \sim e_2 \Leftrightarrow \exists\, k \in \mathrm{N}$ st. $e_1 = e_2\, 2^k \mod 255$. This relation groups exponents in 34 different equivalence classes. Only 16 classes out of the 34 lead to bijective functions. A list of the different security criterias corresponding to these monomials can be found in Appendix B, Table 3. It shows that the AES exponent (class of exponent 127) has the best security parameters and the largest number of multiplications. Our goal is to find an S-box with a lower number of multiplications, maintaining good (although not optimal) security features. In this respect, exponents $7, 29$ and 37 are of interest.

Binomials in $GF(2^8)$. We also performed an exhaustive search over all the S-boxes defined by a binomial. Note that in this case, an additional (refreshing) mask is required for the addition because of the dependency issue mentioned in Section A.3. Again, we were only interested in S-boxes that can be computed in less than 4 multiplications. The number of such binomials was too large for a table representation. Hence, we provide a few examples of the best improvements found, with binomials requiring 2 and 3 multiplications.

- **2 multiplications.** We found binomials having properties similar to monomials X^7 and X^{37}, with better non-linearity (a maximum of the Walsh spectrum between 64 and 48). Binomial $8X^{97}+X^{12}$ is an example.
- **3 multiplications.** In this case, we additionally found several binomials reducing both the maximum value of the Walsh spectrum (from 64 to 48) and the maximum value of the differential spectrum (from 10 to 6) compared to the monomial X^{29}. Binomial $155X^7 + X^{92}$ is an example.

2.2 Constructing 8-bit S-boxes from smaller ones

As the exhaustive analysis of more complex polynomial representations becomes computationally intractable, we now focus on a number of alternatives based on the combination of smaller S-boxes. In particular, we focus on constructions based on 4-bit S-boxes that were previously proposed in the literature, and on 7-bit S-boxes (in order to benefit from the properties of S-boxes with an odd number of bits).

Building on $GF(2^4)$ S-boxes. This is the approach chosen by the designers of PICARO. Namely, they selected an S-box that can be computed using only 4 secure multiplications over $GF(2^4)$. This S-box has good security properties, excepted that its algebraic degree is 4 and that it is non-bijective.

In general, constructing 8-bit S-boxes from the combination of 4-bit S-boxes allows decreasing the memory requirements (e.g. when S-box computations are implemented as look-up tables), possibly at the cost of an increased execution time (as we generally need to iterate these smaller S-boxes). That is, just putting two 4-bit S-boxes side-by-side allows no interaction between the two nibbles of the byte. Hence the maximum of the Walsh spectrum and the maximum of the differential spectrum of the resulting 8-bit S-box are 2^4 times larger than the one of its 4-bit building block. This weakness can be mitigated by using at least two layers of 4-bit S-boxes interleaved with nibble-mixing linear operations. For instance, the KHAZAD [3] and ICEBERG [57] ciphers are using 8-bit S-boxes obtained from three applications of 4-bit S-box layers, interleaved with a bit permutation mixing two bits of each nibble (as illustrated in Figure 5(a)). The resulting S-boxes show relatively good security properties and have maximal algebraic degree. Unfortunately, these proposals are not good candidates to improve the performances of a masked implementations, since six 4-bit S-boxes have to be computed to obtain one 8-bit S-box. As any non-linear permutation in $GF(2^4)$ requires at least 2 multiplications, even using only two layers would cost more secure multiplications than the AES S-box.

Another natural alternative to double the size of an S-box is to build on a small Feistel network, as illustrated in Figure 5(b). Note that in this case, we need to perform at least 3 rounds to ensure that security properties against statistical cryptanalyses will be improved compared to the ones of the underlying 4-bit S-box. Indeed, let us choose a differential (or linear) mask with all active bits in the left part of the input; then after 1 round we obtain the same difference in the right part; hence the differential (or linear) approximation probability after two rounds will be the one of the small S-box again. In fact, an exhaustive analysis revealed that 4-round networks are generally required to obtain good cryptanalytic properties. However, it also turned out that adding a linear layer could lead to improved results for S-boxes that are efficiently masked. That

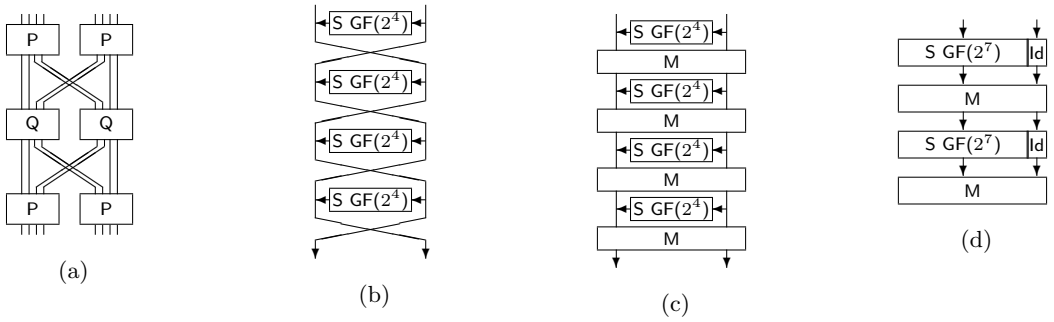


Fig. 1: (a): ICEBERG S-box. (b) 4-round Feistel network without linear mixing layer. (c) 4-round Feistel network with linear mixing layer. (d) Combination of 7-bit S-boxes with linear mixing layer.

is, as illustrated in Figure 5(c), we can add an invertible 8×8 binary matrix to mix the bits of the two Feistel branches between each round. Such a layer allows improving the differential and linear properties of the S-box, with limited impact on the cost of its masked implementations (since the transform is linear).

Example 1. We instantiate the 4-round Feistel network of Figure 5(c) with a 4-bit S-box corresponding to the monomial X^3, and add the 8-bit linear transformation M_1 (given in Appendix C) at the end of each round. The corresponding 8-bit S-box has a maximum differential spectrum of 10, a maximum of the Walsh spectrum equal to 64 and an algebraic degree of 7. It can be computed using 4 secure multiplications in $GF(2^4)$.

Example 2. We instantiate the 4-round Feistel network of Figure 5(c) with a 4-bit S-box using the polynomial $8X + 7X^2 + 7X^3 + 14X^4 + 3X^6 + 6X^8 + 9X^9 + 5X^{12}$ (which can be computed with 1 multiplication), and add the 8-bit linear transformation M_2 (given in Appendix C) at the end of each round. The corresponding 8-bit S-box has a maximum differential spectrum of 8, a maximum of the Walsh spectrum equal to 64 and an algebraic degree of 6. It can also be computed using 4 secure multiplications in $GF(2^4)$.

Summarizing the previous investigations, Table 4 in Appendix D compares the security properties and number of secure multiplications of the proposed S-boxes to the other 8-bit S-boxes build from $GF(2^4)$ ones mentioned at the beginning of the section. The new S-boxes proposed (i.e. Example 1 and Example 2) have the same number of multiplications as the PICARO S-box. They have the additional advantage of being invertible and have better linear and algebraic properties, at the cost of a worse differential spectrum.

Exploiting $GF(2^7)$ and linear layers. We finally investigated the use of a smaller S-box in $GF(2^7)$. This choice was motivated by the fact that S-boxes in $GF(2^n)$ with n odd provide better security properties against differential cryptanalysis than S-boxes acting on an even number of bits. For instance, the existence of Almost Perfect Non-linear permutations (aka APN permutations) is still an open problem for even values of n while many have been constructed for odd values of n. Hence, we expect that low-cost S-boxes acting on 7 bits will exhibit relatively good security properties. As in the previous paragraph, moving from a 7-bit to an 8-bit S-box can be done by combining the 7-bit S-box with an 8-bit linear transform. That is, we used the S-box in Figure 5(d), where the 7-bit S-box is applied twice, separated by a linear transformation to mix bits inbetween. This implies that good masking properties could only be obtained if the 7-bit S-box uses only a single multiplication. We found several 8-bit S-boxes using 2-multiplications based on this design, having 64 as maximum of the Walsh spectrum, 10 as maximum of the differential spectrum and 4 as algebraic degree.

Example 3. We use the monomial X^3 as 7-bit S-box and the linear transform M_3 given in Appendix C.

2.3 Comparing proposed S-boxes to AES one

To conclude this section, we compiled the results we obtained in Table 1, in which most of our performance and security metrics are reported. As explicit with the column "additional operations", such a table is admittedly limited in providing precise estimates of the exact implementation costs, as these costs are always technology-dependent. Yet, it provides general indications about S-box candidates for efficient masking, and also complements the work of Piret et al. in providing some interesting bijective proposals.

Table 1: Comparison of the proposals.

	required randomness (bit)			# sec. mult.	additional operations	security properties		
	$d=1$	$d=2$	d			$deg(S)$	$\max \Delta_S$	$\max \Omega_S$
AES [33]	48	128	$16d^2+32d$	4 (GF(2^8))	7 squ. + 1 Diff. matrix	7	4	32
AES [19]	32	84	$10d^2+22d$	5 (GF(2^4))	3 squ. + 5 Diff. matrix	7	4	32
PICARO	16	48	$8d^2+8d$	4 (GF(2^4))	2 squ.	4	4	68
X^7	24	64	$8d^2+16d$	2 (GF(2^8))	2 squ. + 1 Diff. matrix	3	6	64
X^{29}	32	88	$12d^2+20d$	3 (GF(2^8))	4 squ. + 1 Diff. matrix	4	10	64
X^{37}	24	64	$8d^2+16d$	2 (GF(2^8))	5 squ. + 1 Diff. matrix	3	6	64
$8X^{97}+X^{12}$	32	80	$8d^2+24d$	2 (GF(2^8))	6 squ. + 1 Diff. matrix	3	6	48
$155X^7+X^{92}$	40	104	$12d^2+28d$	3 (GF(2^8))	8 squ. + 1 Diff. matrix	4	6	48
Ex. 1	32	80	$8d^2+24d$	4 (GF(2^4))	4 squ. + 4 Diff. matrix	7	10	64
Ex. 2	48	112	$8d^2+40d$	4 (GF(2^4))	28 squ. + 4 Diff. matrix	6	8	64
Ex. 3	28	70	$7d^2+21d$	2 (GF(2^7))	2 squ. + 2 Diff. matrix	4	10	64

3 Reducing the number of S-box executions

The previous section discussed how to reduce the number of multiplications per S-box execution in a block cipher, by trading cryptanalytic properties for more efficient masking. A complementary approach in order to design a block cipher that is easy to mask is to additionally reduce the total number of S-box executions. For this purpose, a natural solution is to consider rounds where not all the state goes through the S-boxes. To some extent, this proposal can be viewed as similar to an NLFSR-based cipher (e.g. Grain [30], Katan [12], Trivium [13]), where the application of a non-linear component to the state is not homogeneous. For example, say we consider two n-bit block ciphers with s-bit S-boxes: the first (parallel) one applies n/s S-boxes in parallel in each of its R rounds, while the second (serial) one applies only a single S-box per round, at the cost of a larger number of rounds R'. If we can reach a situation such that $R' < R \cdot \frac{n}{s}$, then the second cipher will indeed require less S-boxes in total, hence being easier to protect against side-channel attacks. Of course, the number of S-box executions in the serial version does not have to be stuck at one, and different trade-offs are possible. In general, the relevance of such a proposal highly depends on the diffusion layer. For example, we have been able to conclude that wire crossing permutations (like the one of PRESENT [8]) cannot lead to any improvement of this type (see Appendix E). By contrast, an AES-like structure is better suited to our goal. The rationale behind this intuition essentially relates to the fact that the AES Rijndael has strong security margins against statistical attacks, and the most serious concerns motivating its number of rounds are structural (e.g. [38]). Hence, iterating simplified rounds seems a natural way to prevent such structural attacks while maintaining security against linear/differential cryptanalysis. Furthermore, the impact of linear hulls and differentials in ciphers with strong diffusion could ideally lead to reductions in the total number of S-box executions required to reach a cipher that is secure against statistical attacks. In the following, we show that a modified AES cipher with 4 S-boxes per round (rather than 16) is indeed a good candidate for this purpose. We then put our results together in order to specify our new block cipher `Zorro`.

3.1 The AES Rijndael

The AES Rijndael was designed by Daemen and Rijmen [19]. It operates on message blocks of 128 bits, that can be seen as a matrix of 4×4 bytes. One round is composed of four transformations. In SubBytes (`SB`), a single 8-bit S-box is applied 16 times in parallel to each byte of the state matrix. In ShiftRows (`SR`), the the 4 bytes in the ith row of the state matrix are rotated by i positions to the left. In MixColumns (`MC`), a linear transformation defined by an MDS matrix is applied independently to each column of the state matrix. Finally, in AddKey (`AK`), a 128-bit subkey provided by the key scheduling is added to the internal state by an exclusive or. Depending on the size of the key, the number of rounds varies from 10 to 14. We will compare our design with the 128-bit key version, which simply iterates 10 rounds, with a key whitening in the first one, and no `MC` operation in the last one. We do not describe the key scheduling as we will not reuse it.

3.2 Preliminary investigations: how many S-boxes per round?

As in the previous section (about S-boxes that are easier to mask), an exhaustive analysis of all the round structures that could give rise to less S-box executions in total is out of reach. Yet, and as this number of S-box executions mainly depends on the SB operations, we considered several variants of it, while keeping SR, MC and AK unchanged. For this purpose, we have first analyzed how some elementary diffusion properties depend on the number and positions of the S-boxes within the state. Namely, we considered (1) the number of rounds so that all the input bytes have passed at least once through an S-box (NrSbox); (2) the number of rounds so that all the output bytes have at least one non-linear term (NrNlin); and (3) the maximal number of rounds so that an input difference has a non-linear effect in all the output bytes (NrDiff). In all three cases, these number of rounds should ideally be low. They are given in Appendix F, Table 5 for different S-box configurations. While such an analysis is of course heuristic, it indicates that considering four S-boxes per round, located in a single row of the state matrix seems an appealing solution. In the following, we will carefully analyze the security of this setting in front of various cryptanalysis techniques. Our goal will be to show that an AES-like block cipher where each round only applies four "easy-to-mask" S-boxes as found in the previous section can be secure. In particular, we will select the number of rounds as $R' = 24$, so that we have (roughly) twice less S-boxes executed than the original AES Rijndael (i.e. 24×4 vs. 10×16).

3.3 The block cipher Zorro: specifications

We will use a block size and key size of $n = 128$ bits, iterate 24 rounds and call the combination of 4 rounds a step. Each round is a composition of four transforms: $\mathtt{SB}^*$, AC, SR, and MC, where the two last ones are exactly the same operations as in the AES Rijndael, $\mathtt{SB}^*$ is a variant of SB where only 4 S-boxes are applied to the 4 bytes of the first row in the state matrix, and AC is a round-constant addition described in Appendix G. We additionally perform a key addition AK before the first and after each step. As for the selection of the S-box (given in Appendix H), we will use Example 1 from the previous section, and just add the constant 0xB2 to remove a fixed point. The latter choice is motivated by best trading efficiency (e.g. operations in $GF(2^4)$ can be tabulated) and security (regarding statistical and algebraic attacks). Eventually, and order to maintain high implementation efficiency, we did not design any complex key scheduling and simply add the master key each time AK is called - as in the block cipher LED [29]. Using less key additions than in LED is justified by the exclusion of related-key attacks from our security claims (see the next section for the details). As for other lightweight block ciphers such as NOEKEON [18] or PRINCE [10], we believe that related-key attacks are not relevant for the intended use case (e.g. challenge-response authentication in smart cards), and mainly focused on the generation of a good permutation in the single key setting. A schematic view of the full cipher is given in Appendix I, Figure 6. Reduced-round versions (used in the following) maintain at least three steps, with number of rounds following the pattern: 4-4-4-4-4-4, 4-4-4-4-4-3,4-4-4-4-4-2, 4-4-4-4-4-1, 4-4-4-4-4, ...

4 Security analysis

Despite its AES-like flavor, the irregular structure of the block cipher Zorro makes it quite different than most recently proposed SPNs. As a result, its security evaluation also requires more dedicated cryptanalysis than usually considered when designing such regular ciphers. In this section, we provide a preliminary investigation of a number of standard and less standard attacks against Zorro, paying a particular attention to different solutions to exploit the modified non-linear layer $\mathtt{SB}^*$. While further studies by external cryptanalysts would certainly be welcome, we hope that the following analysis provides reasonable confidence that the proposed structure can lead to a secure block cipher - and will trigger more research in this direction.

4.1 Linear/differential cryptanalysis.

In general, security against linear [40] and differential [5] cryptanalysis can be estimated by counting the number of active S-boxes [20]. Based on the specifications in the previous section, we would need to pass through 28 (resp. 32) S-boxes in order to reach a security level of 2^{128} against differential (resp. linear) cryptanalysis. Nevertheless, since less than 16 S-boxes are applied per round, simple bounds based on the MDS property of the diffusion layer cannot be obtained such as for the AES. An easy shortcoming is that

trails that do not start in the first state row will be propagated through the second round with probability one. Besides, since the S-boxes only apply to one out of the 4 input bytes of MC in each round, the number of active S-boxes also progresses slower. As a result, the main question for bounding security against these statistical attacks is to determine the extent to which actual characteristics can take advantage of this feature, by keeping a maximum number of inactive S-boxes. For this purpose, we propose a technique inspired by hash functions cryptanalysis, that finds the best balance between this number of inactive S-boxes and the number of freedom degrees for the differential (or linear) paths. Taking the example of differential cryptanalysis, we first consider a fully active input state (we discuss next how to adapt our reasoning to other input differences) and a fixed (unknown) key. In this case, we have 16+16 degrees of freedom at the beginning of the differential path (in bytes, i.e. we have 2^{32*8} possible trials to test if the differential path is verified). A first observation is that, in order to have x inactive S-boxes in the next round, we need to verify at least x byte conditions through the MC operation, which will spend x bytes of the freedom degrees available. Conversely, we have that verifying x byte conditions through MC can desactivate at most x S-boxes in the following rounds[2]. Our bounds then follow from the fact that desactivating an S-box is only possible as long as degrees of freedom are available (otherwise there will be no solutions for the differential path). That is, we can consider that for each round i we can ask x_i conditions to be verified through the MC transform, and that at most x_i S-boxes will not be activated in the following rounds because of these conditions. Hence, the following inequalities have to be verified for finding a valid path. They represent the degrees of freedom still available after r rounds, and the cumulated number of active S-boxes (that must be smaller than 28 as previously pointed out):

$$\sum_{i=1}^{r} x_i < 32, \qquad \text{and} \qquad 4 \times r - \sum_{i=1}^{r} x_i < 28.$$

For the sake of simplicity, we can just consider the average number of conditions $\bar{x}$ that we can impose at each round. We then observe that the highest number of rounds is achieved for $r = 14$ and $\bar{x} = 32/14 = 2.285$, where we have 24 active S-boxes and no more freedom degrees available (for 15 rounds, the number of active S-boxes exceeds 28). Eventually, we note that when the initial state is not completely active, e.g. taking only Y possible differences, we have that with $c_{in} = \log_2(2^{16*8}/Y)/8$ byte conditions we will be able to desactivate at most c_{in} S-boxes. Hence, the inequalities taking all possible input differences into account become:

$$\sum_{i=1}^{r} x_i < 32 - c_{in}, \qquad \text{and} \qquad 4 \times r - \sum_{i=1}^{r} x_i - c_{in} < 28.$$

They provide the same result as before: 14 rounds is the upper bound for building a classical differential path[3]. A similar reasoning for linear cryptanalysis leads to an upper bound of 16 rounds (out of 24).

4.2 Truncated differential attacks

In view of the non-linear transformation in Zorro, a natural extension of differential cryptanalysis to investigate is the use of dedicated truncated differentials [36]. In particular, the most damaging truncated differential patterns are those that would exclude active bytes affected by non-linear operations. For this reason, we analyzed the possible existence of cycles of differences that verify transitions from three active rows of the state to another three active rows with probability one for any number of rounds (i.e. excluding non-linear operations). Such patterns are represented in Figure 2, where big squares represent states, small squares represent bytes, highlighted ones are affected by non-linear transformations and gray bytes are the ones with a non-zero difference. Truncated differentials only following the pattern of the figure would never go through the S-boxes. Quite naturally, staying in this pattern for several rounds implies more conditions, but if an input difference exists so that it follows the pattern for some rounds before regenerating this first

[2] For example, consider the case where the first output byte of MC is inactive, meaning that we have one less active S-box in the next round. For more S-boxes to be inactive, we would have to pay more conditions on MC. Alternatively, say MC has only one active output difference per column (hence implying $x = 12$ byte conditions). Then, we will have at most 6 inactive S-boxes in the two next rounds, before coming back to the whole active state with $6 < x$.

[3] Note that despite these bounds to being possibly loose for small number of rounds, they also guarantee security against boomerang attacks. Namely, we have at least 9 active S-boxes after 10 rounds, which would correspond to best differentials with probabilities $p, q \approx 2^{42}$ in a boomerang attack (leading to $p^2q^2 \approx 2^{-168}$).

input difference again, this would imply that the pattern can be followed for an infinite number of rounds as a cycle would have been created. If no cycle exists, we have essentially 4 byte constraints per round for 12 unknowns, and we run out of degrees of freedom for verifying the pattern after 3 rounds. As a result, we essentially have to ensure that no cycle has been created, that would prevent differences to affect the first state row for an infinite number of rounds. The probability that such a cycle exists is small (about $2^{64-96} + 2^{32-96} + 2^{-96} \approx 2^{-32}$). Yet, in order to be sure they do not exist, we performed an exhaustive search over all the 3-row input differences, and checked whether they generate a cycle or end by spreading the difference. The naive cost of such a search is $2^{12*8} = 2^{96}$. We describe a time and memory efficient alternative in Appendix J. It allowed us to verify that the pattern of Figure 2 can be verified for at most two rounds.

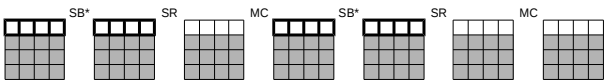


Fig. 2: Two rounds of truncated differential pattern.

4.3 Meet-in-the-middle and bicliques

Biclique cryptanalysis has been introduced in [33] and recently attracted a lot of attention because of its application to the full AES in [7]. It can be viewed as an improvement of classical meet-in-the-middle attacks, where the starting point does not correspond to a single state but to several rounds, that are covered with a structure called biclique. In the case of the full-AES, this principle can be applied so that the complexity of verifying each key candidate is reduced, hence leading to an accelerated exhaustive search. The direct extension of such an attack to our new algorithm does not strongly differ from attacks against the AES. Yet, because of our particular key addition, the number of rounds covered by bicliques as described in [7] would be bigger. We have evaluated that the constant exhaustive key search complexity reduction for 24 cipher rounds is larger than 0.5 (which improves the security over the 0.27 constant found for the AES).

Quite naturally and as in the previous section, the most interesting attacks against `Zorro` are the ones taking advantage of its particular structure. In the following, we describe a dedicated meet-in-the-middle attack for this purpose. Its main specificity is that, while classical meet-in-the-middle attacks work with pairs of plaintexts and ciphertexts to recover the key, our specialized attack will consider quadruplets of the type (plaintext_1, plaintext_2, ciphertext_1, ciphertext_2). This will allow us to extend meet-in-the-middle cryptanalysis by two more rounds, by choosing input differences that do not go through the S-boxes after the first round, and only go through one S-box after the second round. That is, since other round transformations are linear, we can compute differences after two rounds with only 2^8 guesses. As a result, we will match differences in the middle of the cipher (rather than values as traditionally done). The principle of the attack is represented in Figure 3 in which (*i*) the gray bytes are the bytes with differences that we know or guess; (*ii*) the bytes with '?' have an unknown difference; and (*iii*) the bytes with 'a', 'b', 'c' or 'd' are such that if their corresponding byte at the beginning of the state 4 is known, then they are also known. As in Figure 2, the highlighted bytes are the ones affected by S-boxes. The middle is placed in round 5 (through the `MC` transformation). On the sides of the figure, we added the cost for predicting gray bytes in both directions, which comes for the guessing of state bytes each time a difference goes through an S-box.

For the sake of simplicity, we will consider that any bit of internal state recovered can be translated into a key bit (since actual partial key recoveries can only be more complex, this also provides us with confident security margins). Under this assumption, the attack essentially proceeds as follows. Given one pair plaintext/ciphertext, we choose the second plaintext so that it has a one-byte difference with the first one that is not located on the first state row. As previously said, it allows us to postpone the guessing of bits compared to classical meet-in-the-middle attacks. Next, we perform 2^8 guesses each time we pass through an S-box, both forward and backward. In Figure 3, independent groups of bytes involved in the middle match are represented with different letters. In the right middle state, we can see three gray rows that have been guessed in the backward direction with a cost of $2^{32*3} = 2^{96}$. In the left middle state, we can see three colored rows, that have been determined in the forward direction with a cost of $2^{32+32+8} = 2^{72}$. As the match in the middle is done through the (linear) `MC` operation, and we completely know three rows before and three rows after it, we have 64 bits of conditions in total. This means that we will keep $2^{96+72-64} = 2^{104}$

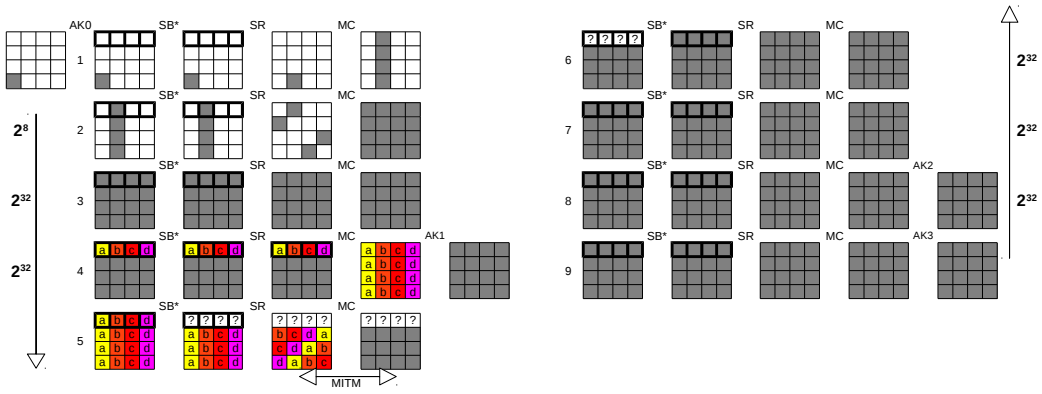


Fig. 3: Representation of the 9 rounds meet-in-the-middle scenario

possibilities. Considering the (pessimistic) case where these guesses directly translate into key bits, we only have 2^{104} possibilities for 128 bits. In other words, the proposed attack reduces the cost of an exhaustive key search by a factor 2^{24}. Note that we could consider better ways of making the merge in the middle point, by exploiting the independence between the colored groups of bytes. But even in this case, attacking more than 12 rounds (as illustrated in Appendix K, Figure 7) is unlikely. Namely, we need an additional 2^{32} key guesses per round, and even supposing that the colored bytes can be merged with a reduced cost, the time complexity of the 12-round attack would be at least $2^{96+8} + 2^{96}$ (so adding one more round with 2^{32} guesses would increase this complexity beyond 2^{128}). Eventually, we note that this attack might be combined with bicliques for increasing the number of targeted rounds (with the size of the bicliques equal to the number of rounds added). The straightforward application of the AES techniques would suggest an improvement of two rounds, still leaving a comfortable security margin for the 24 rounds we suggest for `Zorro`.

4.4 Impossible differential attacks

Impossible differential attacks exploit differential paths over some cipher rounds that cannot occur in order to discard key candidates leading to these differences to happen (hence reducing the complexity of an exhaustive key search) [4]. In this section, we describe such an attack against 10 rounds of `Zorro`. It is based on two main ingredients. First, we re-use the property (observed in Section 4.2) that we can choose up to $2^{96-32*2} = 2^{32}$ differences on the last three state rows so that the difference in the first row remains inactive after two `MC` operations with probability one[4]. We will use this property twice, namely for rounds 2, 3 and for rounds 8, 9. Second, we will take advantage of the best differential characteristic of our S-box (with probability 10/256). The attack principle is pictured in Figure 4, where the impossible differential path stands between rounds 2 and 10. Bytes denoted with a c (resp. k) are such that their difference is chosen (resp. known). The 0's correspond to bytes with no difference and the '?'s represent the bytes whose differences have gone through an S-box and are consequently unknown. The remaining bytes (i.e. with A, B or nothing written on them) are unknown bytes that still verify certain known relations. Eventually, the output bytes are represented with s' meaning that although all of them are active, they have been generated by a concrete subspace of size 2^{32} when the conditions of the impossible path are verified. Given these notations, we first choose one out of the 2^{32} differences in three rows that keep the first row inactive through two `MC` operations, and fix it to the first state of the second round. The attack will then essentially exploit a chosen difference Δ_{in} at the beginning of this second round, and look for impossible differences Δ_{out} in round 10. As previously mentioned, we will choose Δ_{in} so that the difference in the output of `SB`* in the first round corresponds to the best S-box characteristic. Next, we observed that for a chosen Δ_{in}, we can precompute if there exist a Δ_{out} such that the middle transition (also represented in the figure) is impossible. That is, for a fixed Δ_{in} and on average (over the state values and keys), there is only a probability 2^{-4} of finding a Δ_{out} such that this middle transition is possible[5]. As a result, we can easily filter the Δ_{in}'s leading to impossibilities and use them in our attack.

[4] As previously detailed, it does not extend to more rounds which prevents attack improvements in this direction.

[5] Namely, we have 2^{32} possible output differences $\times$ 2^{32*2} bits of state values that affect the path $\times$ 2^{-96} conditions in the middle $\times$ 2^{-4} conditions for the difference transition to exist through the S-boxes $= 2^{-4}$. This can be tested

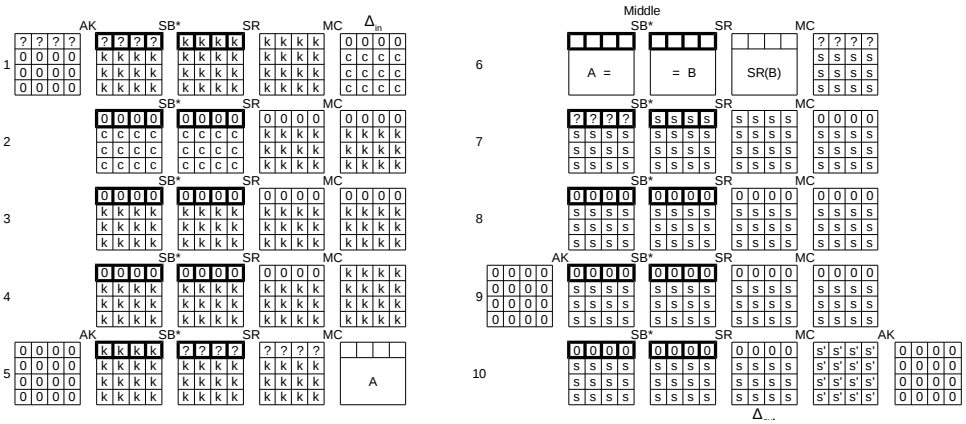


Fig. 4: Representation of the 10 rounds impossible differential attack

Once a correct Δ_{in} is chosen, we can compute the differences in the input plaintext bytes denoted with k, and choose a difference corresponding to the input difference of the best S-box characteristic (so that 10 values per S-box can make the transition to Δ_{in} for the '?' bytes in the first state). We then generate 2^{115} different pairs of plaintexts by modifying the values in the last 3 rows and 19 bits of the first row. Our goal will be to discard key candidates in order to identify the correct 32 bits of the first key row. For this purpose, and for each of the 2^{96} values in the three last rows that we try for a fixed value in the first row, we expect to find the output corresponding to the impossible differential path once. When this occurs, all the keys that verify the transition to Δ_{in} for that plaintext value must be discarded. This means $10^4 \approx 2^{13.3}$ discarded values for the 32 bits of the key per first row value tried. As we typically want to discard all the wrong candidates but the correct one, we need to repeat this procedure $2^{32}/2^{13.3} = 2^{18.7} \approx 2^{19}$ times. We point out here that we chose the 2^{19} values for the first row in a manner that all the groups of size $2^{13.3}$ are represented (i.e. so that the discarded keys are different for each of the 2^{19} values). To sum up, by trying 2^{115} different values for the plaintext, we obtain $2^{115-96} = 2^{19}$ output pairs corresponding to the impossible path. For each of these 2^{19} pairs, we discard the keys that make the first $\mathtt{SB}^*$ transition to Δ_{in} possible (i.e. $2^{13.3}$ per plaintext pair). This procedure allows us to recover the 32 first keybits with a complexity of 2^{115}, and next the whole key (as the remaining 96 key bits can be found by exhaustive search).

4.5 Derivative and Algebraic analysis

A standard requirement for iterated block cipher constructions is that a few of their rounds allow reaching the maximum algebraic degree (here 127). Nevertheless, as in the previous sections standard techniques for estimating this degree (e.g. [11]) do not directly apply. In the following, we approximate that the state-bit equations expressed as function of their input-bit variables reach their maximum degree after 6 rounds. For this purpose we first observed that while being of degree 7, the chosen S-box has four of its coordinates of degree 6 (and the four components of degree 7 share the same degree-7 term). Taking into account the particular structure of the $\mathtt{SB}^*$ layer, we have deduced the following relation for estimating the degree of the bits of the state. Assuming that at round r, the bits from the first row have degree d_0^r and the others $d_{1,2,3}^r$, then the degrees obtained after the next S-box application are $d_0^{r+1} \leq d_0^r + 6d_{1,2,3}^r$ and $d_{1,2,3}^{r+1} \leq d_0^r$. Since the initial values are $d_0^0 = d_{1,2,3}^0 = 1$, we directly obtain that after 5 rounds, the bound is larger than 128 and thus the bit degrees should be close (or equal to) 127. Following, and in order to verify the validity of these equations, we have additionally checked in detail what happens during the third round. Starting with the S-box output, we found that their degree is 53, which is quite close to the 55 obtained with our previous estimation. We further noticed that the monomials of degree 53 of these 4 bytes have 28 variables in common (which correspond to the terms that reached degree 7 after the first $\mathtt{SB}^*$ layer). Amongst the 25 remaining variables, 20 are exclusive of each monomial, and the remaining five can take various values.

with a cost of approximately 2^{32}. Furthermore, if one transition to a fixed Δ_{out} was possible, it would not change the complexity, as we would just have to discard one out of the 2^{32} pairs that we tried.

Several monomials of degree 53 can also be generated after the S-boxes, and because of the symmetry of the construction, we can ensure that after each S-box, the 5 remaining variables take at least 10 different values. This means that in the (unlikely) worst-case scenario where round 4 would not increase the degree, two rounds later the sixth round will multiply for sure the four terms of degree 53 (because of the `MC` of round 4 and the `SR` of round 5). Hence, we can guarantee that the degree will reach at least $28 + 20 * 4 + 10 = 118$ at this stage. As from round 4 on, all the variables appear in all the bytes, each S-box will at least add one new variable to the highest-degree term. This means that the maximum degree is surely reached in round 6.

Cube testers. As a complement to the previous approximations, we also launched a heuristic analysis of higher-order derivatives within `Zorro`. For this purpose, we used the cube testers introduced in [2] and next improved in [23, 35] by imposing conditions that allow detecting non-random properties for more rounds and allowing to recover some key bits. Cube testers embrace other analysis tools (e.g. [24, 25]) and essentially aim at (statistically) detecting some non-random properties of some bits in the derivatives of some cipher state equations. As previously discussed, the reduced number of S-boxes in $\texttt{SB}^*$ leads the degree of the internal state bits to grow slower and less homogeneously than for the AES. Hence, we have performed several tests to check the number of rounds for which we could distinguish our construction from a random one. In particular, we have looked for linear dependencies, neutrality of variables and balancedness in the super-poly terms associated to the cube tested. Experiments have been performed for several trade-offs between the number of samples (up to 2^{24}) and the size of the cubes (up to 2^{16}). We also tested different cubes, but we obtained similar results with most of them. The most adequate ones turned out to be either corresponding to any couple of bytes in the 4×4 matrix, or corresponding to a set of bits located at the same position in the state bytes. The minimum number of rounds such that no particular weakness was detected are reported in Table 2, for different S-box choices and number of S-boxes per round. The highest number of rounds that we could distinguish was 7, which could be done using 2^8 samples and a cube of size 2^{16}. Considering more samples or cubes did not allow us to extend the distinguisher to any more rounds. This is to compare with 4 rounds that could be distinguished for the AES Rijndael. Hence, this experiment suggests that 24 rounds of `Zorro` should provide a similar security level as the AES with respect to this type of properties.

Table 2: Minimum number of rounds for which the cube tester did not find weaknesses.

	2-byte cubes		16-bit cubes	
	`SB`	$\texttt{SB}^*$	`SB`	$\texttt{SB}^*$
AES S-box	4	6	4	6
`Zorro S-box`	4	6	5	7

4.6 Rebound attacks

Rebound attacks have been introduced in [41] and widely applied in the context of the SHA-3 competition. Their first application was to provide distinguishers for the compression functions of AES-like hash functions. Besides, they have also been used for deducing non-random properties for the underlying permutation of some block ciphers [29, 43]. In view of the new round structure proposed for `Zorro`, they consequently are a tool of choice for better understanding the permutations generated. Hence, we have adapted rebound attacks in order to be able to apply them to our structure. For this purpose, we propose an original way to compute bounds on the maximal number of rounds for which we could distinguish such fixed-key permutations from a random one. The details of this analysis are reported in Appendix L. Summarizing, we could distinguish up to 12 rounds, which (as expected) is more than the best rebound distinguisher for the AES (8 rounds).

4.7 Related-key attacks

Security against related-key attacks is not claimed for `Zorro`. Nevertheless, we believe that a few observations regarding them is important to further justify our design choices for the key scheduling and number of key additions (i.e. the number of rounds per step). In particular, we first would like to point out that two extreme solutions in this respect lead to extremely strong related-key issues. First, say we would add the key after every round and we have a pair of related keys with $\Delta_k = a \oplus \texttt{MC(SR(a))}$, where a has no difference in the first row. Then, it is easy to see that a plaintext difference $\Delta = \texttt{MC(SR(a))}$ will propagate through all the

rounds with probability one. There are 2^{96} such related-keys. A similar probability-one distinguisher exists with 2^{32} related keys if the key is added every 2 rounds (using the results from Section 4.2). By contrast, if the key is added every three rounds, no probability-one related-key distinguisher exists anymore. Now say that we would add the key only three times, e.g. with 2 steps of 10 rounds in between. Then, we can build a related key boomerang distinguisher with probability one as follows. First encrypt a pair of plaintexts p_1, p_2 such that $\Delta = p_1 \oplus p_2$ under related keys k_1, k_2 such that $\Delta = k_1 \oplus k_2$, with $c_1 = \mathtt{E}_{k_1}(p_1)$ and $c_2 = \mathtt{E}_{k_2}(p_2)$. Next build $c_3 = c_1 \oplus \Delta$ and $c_4 = c_2 \oplus \Delta$. Eventually decrypt c_3 with k_2 and c_4 with k_1. Since the differential probabilities through half the cipher equal to one, we also have $p_3 \oplus p_4 = \Delta$ with probability one.

These two extreme situations motivated us to select an intermediate number of key additions for `Zorro`, where related-key issues could only be observed with smaller probabilities. In this respect, we first refer to the results of Mendel et al. [42], where it is shown that the "generic" related-key attack against multiple Even-Mansour given in [9] extends from 2 to 3 (resp. 4) rounds if good differentials (resp. iterative differentials) can be found for the inner permutations (aka steps). We also refer to the recent announcements of Dinur et al. regarding key recovery attacks against 3-round Even-Mansour constructions [22]. From these state-of-the-art results, we expect that possible related-key attacks against `Zorro` will require sufficiently high data complexities for not being a concern in the fixed-key setting for which we claim security.

5 Concluding remarks

The previous cryptanalysis investigations are admittedly far from exhaustive. Yet, we believe that the attacks evaluated are among the most relevant regarding the structure and components of `Zorro`. A number of other standard cryptanalysis techniques would naturally apply just like for any other cipher. One can mention the slide attacks introduced in [6] and exploiting the similarity of the round functions (that are prevented by the use of round constants). Another example are integral attacks exploiting properties of the `MC` transform [38]. Since our modified $\mathtt{SB}^*$ does not affect these diffusion properties, they would target 7 rounds, just as for the AES [37]. We leave the investigation of these alternative attack paths as a scope for further research.

To conclude this work, we report on masked implementations of `Zorro` in an Atmel AtMega644p 8-bit microcontroller. In order to justify the interest of this new cipher, we compared its performance figures with two natural competitors, namely the AES and PICARO. We considered the schemes of Rivain and Prouff [54] for this purpose. In the AES case, we also considered the optimization from Kim et al. [34]. The results of Figure 5 suggest that the AES remains most efficient cipher in the unprotected case, while PICARO and `Zorro` gradually lead to improved cycle counts with larger masking orders. The fact that `Zorro` exploits both an improved S-box and a modified structure explains its asymptotic gain over PICARO. Besides, we recall that using bijective S-boxes is important in order to avoid easy attack paths for non-profiled side-channel analysis. Note that considering the polynomial masking scheme of Prouff and Roche in [50] could only lead to more significant gains (since the cost of masking is cubic in the security order in this case).

Finally, we stress that the design of `Zorro` leads to interesting open problems regarding further optimizations for algorithms that are "easy to mask". Keeping the (generic) criteria of minimizing the number of field multiplications in the algorithm, a natural direction would be to consider cipher designs with stronger diffusion layers such as Khazad [51]. Alternatively, one could also give up a bit of our generality and focus exclusively on Boolean masking (e.g. the Rivain and Prouff 2010 scheme) while giving up polynomial types of masking schemes (e.g. the Prouff and Roche 2011 one). For example, the S-boxes of block ciphers such as PRESENT [8] or NOEKEON [18] require three multiplications in $GF(2^{16})$, which makes them less suitable than `Zorro` regarding our current optimization criteria (as these ciphers require 16×32 and 31×16 of these S-boxes, respectively). But they have efficient bitslice representations minimizing the number of AND gates, which could lead to further improvements of Boolean masked implementations. In general, taking advantage of bitslicing in this specialized context, while maintaining a "regular" design (e.g. excluding bit manipulations that would leak more on certain bits than others) appears as an interesting open problem.

Acknoledgements. We would like to thank Dmitry Khovratovich for having pointing out the strong related key issue occurring when adding the key after each round of `Zorro`. We also would like to thank Orr Dunkelman for having shared his recent results about the cryptanalysis of Even-Mansour constructions, which motivated us to increase the number of steps in the final version of `Zorro`. This work has been funded in

parts by the European Commission through the ERC project 280141 (acronym CRASH) and the European ISEC action grant HOME/2010/ISEC/AG/INT-011 B-CCENTRE project. François-Xavier Standaert is an associate researcher of the Belgian Fund for Scientific Research (FNRS-F.R.S.).

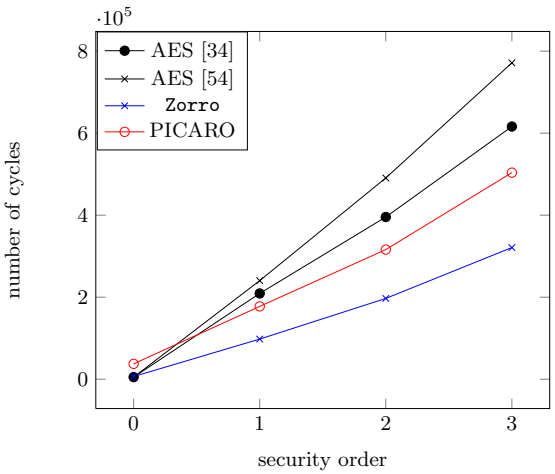


Fig. 5: Performance evaluation.

References

1. Masayuki Abe, editor. *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*. Springer, 2010.
2. Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. Cube testers and key recovery attacks on reduced-round MD6 and Trivium. In *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2009.
3. P. Barreto and V. Rijmen. The KHAZAD legacy-level block cipher. *Primitive submitted to NESSIE*, page 4, 2000.
4. Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In Jacques Stern, editor, *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 12–23. Springer, 1999.
5. Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *CRYPTO*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.
6. Alex Biryukov and David Wagner. Slide attacks. In Lars R. Knudsen, editor, *FSE*, volume 1636 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 1999.
7. Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371. Springer, 2011.
8. Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In Paillier and Verbauwhede [46], pages 450–466.
9. Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, François-Xavier Standaert, John P. Steinberger, and Elmar Tischhauser. Key-alternating ciphers in a provable setting: Encryption using a small number of public permutations - (extended abstract). In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 45–62. Springer, 2012.
10. Julia Borghoff, Anne Canteaut, Tim Gneysu, Elif Bilge Kavun, Miroslav Kneevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Sren S. Thomsen, and Tolga Yalin. PRINCE - a low-latency block cipher for pervasive computing applications. In Wang and Sako [60], pages 208–225.
11. Christina Boura, Anne Canteaut, and Christophe De Cannière. Higher-order differential properties of KECCAK and *luffa*. In Joux [32], pages 252–269.

12. Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - a family of small and efficient hardware-oriented block ciphers. In Christophe Clavier and Kris Gaj, editors, *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2009.
13. Christophe De Cannière and Bart Preneel. Trivium. In Matthew J. B. Robshaw and Olivier Billet, editors, *The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 244–266. Springer, 2008.
14. D. Canright and Lejla Batina. A very compact "perfectly masked" S-Box for AES. In Steven M. Bellovin, Rosario Gennaro, Angelos D. Keromytis, and Moti Yung, editors, *ACNS*, volume 5037 of *Lecture Notes in Computer Science*, pages 446–459, 2008.
15. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
16. Jean-Sébastien Coron, Emmanuel Prouff, and Matthieu Rivain. Side channel cryptanalysis of a higher order masking scheme. In Paillier and Verbauwhede [46], pages 28–44.
17. Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In Yuliang Zheng, editor, *ASIACRYPT*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer, 2002.
18. Joan Daemen, Michaël Peeters, Gilles Van Assche, and Vincent Rijmen. Nessie proposal: NOEKEON, 2000. Available online at `http://gro.noekeon.org/Noekeon-spec.pdf`.
19. Joan Daemen and Vincent Rijmen. Rijndael candidate for AES. In AES *Candidate Conference*, pages 343–348, 2000.
20. Joan Daemen and Vincent Rijmen. The wide trail design strategy. In Bahram Honary, editor, *IMA Int. Conf.*, volume 2260 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 2001.
21. Donald W. Davies and Sean Murphy. Pairs and triplets of DES S-Boxes. *J. Cryptology*, pages 1–25, 1995.
22. Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Key Recovery Attacks on 3-round Even-Mansour (with Applications!), Eurocrypt rump session, May 2013.
23. Itai Dinur and Adi Shamir. Breaking Grain-128 with dynamic cube attacks. In Joux [32], pages 167–187.
24. Håkan Englund, Thomas Johansson, and Meltem Sönmez Turan. A framework for chosen IV statistical analysis of stream ciphers. In K. Srinathan, C. Pandu Rangan, and Moti Yung, editors, *INDOCRYPT*, volume 4859 of *Lecture Notes in Computer Science*, pages 268–281. Springer, 2007.
25. Simon Fischer, Shahram Khazaei, and Willi Meier. Chosen IV statistical analysis for key recovery attacks on stream ciphers. In Serge Vaudenay, editor, *AFRICACRYPT*, volume 5023 of *Lecture Notes in Computer Science*, pages 236–245. Springer, 2008.
26. Henri Gilbert and Helena Handschuh, editors. *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005*, volume 3557 of *Lecture Notes in Computer Science*. Springer, 2005.
27. Henri Gilbert and Thomas Peyrin. Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations. In Seokhie Hong and Tetsu Iwata, editors, *FSE*, volume 6147 of *Lecture Notes in Computer Science*, pages 365–383. Springer, 2010.
28. Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya Koç and Christof Paar, editors, *CHES*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.
29. Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In Preneel and Takagi [48], pages 326–341.
30. Martin Hell, Thomas Johansson, and Willi Meier. Grain: a stream cipher for constrained environments. *IJWMC*, 2(1):86–93, 2007.
31. Jérémy Jean, María Naya-Plasencia, and Thomas Peyrin. Improved rebound attack on the finalist Grøstl. In *FSE*, Lecture Notes in Computer Science. Springer, 2012. to appear.
32. Antoine Joux, editor. *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011*, volume 6733 of *Lecture Notes in Computer Science*. Springer, 2011.
33. Dmitry Khovratovich, Christian Rechberger, and Alexandra Savelieva. Bicliques for preimages: Attacks on Skein-512 and the SHA-2 family. In Anne Canteaut, editor, *FSE*, volume 7549 of *Lecture Notes in Computer Science*, pages 244–263. Springer, 2012.
34. HeeSeok Kim, Seokhie Hong, and Jongin Lim. A fast and provably secure higher-order masking of AES s-box. In Preneel and Takagi [48], pages 95–107.
35. Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional differential cryptanalysis of NLFSR-based cryptosystems. In Abe [1], pages 130–145.
36. Lars R. Knudsen. Truncated and higher order differentials. In Bart Preneel, editor, *FSE*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1994.
37. Lars R. Knudsen and Vincent Rijmen. Known-key distinguishers for some block ciphers. In *ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 315–324. Springer, 2007.

38. Lars R. Knudsen and David Wagner. Integral cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *FSE*, volume 2365 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2002.
39. Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In Alfred Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.
40. Mitsuru Matsui. Linear cryptoanalysis method for DES cipher. In Tor Helleseth, editor, *EUROCRYPT*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993.
41. F. Mendel, C. Rechberger, M. Schläffer, and S. S. Thomsen. The Rebound Attack: Cryptanalysis of Reduced WHIRLPOOL and Grøstl. In *Fast Software Encryption - FSE 2009*, volume 1008 of *Lecture Notes in Computer Science*. Springer, 5665.
42. Florian Mendel, Vincent Rijmen, Deniz Toz, and Kerem Varici. Differential analysis of the LED block cipher. In Wang and Sako [60], pages 190–207.
43. Ivica Nikolic, Josef Pieprzyk, Przemyslaw Sokolowski, and Ron Steinfeld. Known and chosen key differential distinguishers for block ciphers. In Kyung Hyune Rhee and DaeHun Nyang, editors, *ICISC*, volume 6829 of *Lecture Notes in Computer Science*, pages 29–48. Springer, 2010.
44. Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. A side-channel analysis resistant description of the AES S-Box. In Gilbert and Handschuh [26], pages 413–423.
45. Elisabeth Oswald and Kai Schramm. An efficient masking scheme for AES software implementations. In JooSeok Song, Taekyoung Kwon, and Moti Yung, editors, *WISA*, volume 3786 of *Lecture Notes in Computer Science*, pages 292–305. Springer, 2005.
46. Pascal Paillier and Ingrid Verbauwhede, editors. *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*. Springer, 2007.
47. Gilles Piret, Thomas Roche, and Claude Carlet. PICARO - a block cipher allowing efficient higher-order side-channel resistance. In Feng Bao, Pierangela Samarati, and Jianying Zhou, editors, *ACNS*, volume 7341 of *Lecture Notes in Computer Science*, pages 311–328. Springer, 2012.
48. Bart Preneel and Tsuyoshi Takagi, editors. *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011*, volume 6917 of *Lecture Notes in Computer Science*. Springer, 2011.
49. Emmanuel Prouff. DPA attacks and S-Boxes. In Gilbert and Handschuh [26], pages 424–441.
50. Emmanuel Prouff and Thomas Roche. Higher-order glitches free implementation of the AES using secure multi-party computation protocols. In Preneel and Takagi [48], pages 63–78.
51. Vincent Rijmen and Paulo Barreto. Nessie proposal: KHAZAD, 2000. Available online at `http://www.larc.usp.br/~pbarreto/KhazadPage.html`.
52. Vincent Rijmen, Bart Preneel, and Erik De Win. On weaknesses of non-surjective round functions. *Des. Codes Cryptography*, pages 253–266, 1997.
53. Matthieu Rivain, Emmanuelle Dottax, and Emmanuel Prouff. Block ciphers implementations provably secure against second order side channel analysis. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *Lecture Notes in Computer Science*, pages 127–143. Springer, 2008.
54. Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010.
55. Yu Sasaki, Yang Li, Lei Wang, Kazuo Sakiyama, and Kazuo Ohta. Non-full-active Super-Sbox Analysis: Applications to ECHO and Grøstl. In Abe [1], pages 38–55.
56. Kai Schramm and Christof Paar. Higher order masking of the AES. In David Pointcheval, editor, *CT-RSA*, volume 3860 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2006.
57. François-Xavier Standaert, Gilles Piret, Gaël Rouvroy, Jean-Jacques Quisquater, and Jean-Didier Legat. ICEBERG : An involutional cipher efficient for block encryption in reconfigurable hardware. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 279–299. Springer, 2004.
58. François-Xavier Standaert, Nicolas Veyrat-Charvillon, Elisabeth Oswald, Benedikt Gierlichs, Marcel Medwed, Markus Kasper, and Stefan Mangard. The world is not enough: Another look on second-order DPA. In Abe [1], pages 112–129.
59. Nicolas Veyrat-Charvillon and François-Xavier Standaert. Generic side-channel distinguishers: Improvements and limitations. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 354–372. Springer, 2011.
60. Xiaoyun Wang and Kazue Sako, editors. *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012*, volume 7658 of *Lecture Notes in Computer Science*. Springer, 2012.
61. Carolyn Whitnall, Elisabeth Oswald, and François-Xavier Standaert. The myth of generic DPA...and the magic of learning. Cryptology ePrint Archive, Report 2012/256, 2012. `http://eprint.iacr.org/`.

A Background

A.1 The Rivain-Prouff 2010 masking scheme

The CHES 2010 scheme described in [54] is based on Boolean masking. That is, its initial secret sharing consists in randomly picking d elements $\{x_i\}_{i=1}^{d}$, and computing $x_0 = s \oplus x_1 \oplus \cdots \oplus x_d$ where the $d+1$ variables x_i are called the shares. As the observation of d shares does not provide information about the secret value s, we have that order-d Boolean masking ideally provides d-th order SCA security[6]. In this context, all the block cipher operations that are linear over $GF(2)$ can be applied independently to each share (e.g. bit permutations, bitwise XORs). By contrast, non-linear operations (i.e. S-boxes, typically) require the joint manipulation of multiple shares. In the following, we will consider n-bit bijective S-boxes, that can be represented as a polynomial $S : \mathbb{F}_{2^n} \rightarrow \mathbb{F}_{2^n}$. Using this representation, the only non-linear operation is the field multiplication. The efficient solution to perform a d-th order SCA-secure field multiplication proposed by Prouff and Rivain is given in Algorithm 1, where $r \in_R \mathbb{F}_{2^n}$ means that r is uniformly randomly chosen in $\mathbb{F}_{2^n}$. It requires the generation of $\frac{d^2+d}{2}$ random n-bit values, $d^2 + 2d + 1$ field multiplications and $2d^2 + 2d$ XORs.

Algorithm 1 Multiplication of two masked secrets $\in \mathbb{F}_{2^n}$.

Require: Shares x_i and y_i such that $x = x_d \oplus \cdots \oplus x_0$ and $y = y_d \oplus \cdots \oplus y_0$
Ensure: Shares w_i such that $xy = w = w_d \oplus \cdots \oplus w_0$
  for i from 0 to d **do**
    for j from $i+1$ to d **do**
      $r_{i,j} \in_R \mathbb{F}_{2^n}$
      $r_{j,i} \leftarrow r_{i,j} \oplus x_i y_j \oplus x_j y_i$
    end for
  end for
  for i from 0 to d **do**
    $w_i \leftarrow x_i y_i$
    for j from 0 to $d, j \neq i$ **do**
      $w_i \leftarrow w_i \oplus r_{i,j}$
    end for
  end for
  return $(w_d, ..., w_0)$

A.2 Cryptanalytic properties for S-boxes

S-boxes exhibiting good properties against SCAs are usually weaker against mathematical cryptanalysis [49]. As one goal of this paper is to find an adequate trade-off between these conflicting goals, this section briefly summarizes the main cryptographic properties we will consider. As mentioned in introduction, we will focus in bijective S-boxes since (a) non-bijective S-boxes have already been investigated in [47] and (b) non-bijective S-boxes are more exposed to structural attacks [21, 52] and also more sensitive to so-called generic (non-profiled) SCAs [61]. We now recall some tools used for evaluating the resistance of S-boxes against linear, differential and algebraic attacks. Such tools are based on Boolean functions theory. For this purpose, we consider an S-box as a vector of Boolean functions $S = (f_0, \ldots, f_{n-1})$, $f_i : \mathbb{F}_{2^n} \rightarrow \mathbb{F}_2$. For $x \in \mathbb{F}_{2^n}$ and $u \in \mathbb{F}_2^n$, the notation x^u stands for the product $\prod_{i=0}^{n-1} x_i^{u_i}$, with the convention $0^0 = 1$. We will denote by $\#A$ the cardinality of a set A and by $\langle a, b \rangle$ the dot product between two elements $a, b \in \mathbb{F}_{2^n}$: $< a, b >= \sum_{i=0}^{n-1} a_i b_i$.

Non-linearity. Linear cryptanalysis is one of the most investigated attacks against block ciphers [40]. To prevent it, the target algorithm must present a high non-linearity (usually coming from the S-box characteristics). The Walsh transform can be used to evaluate the correlation of a linear approximation $(a, b) \neq (0, 0)$.

[6] Again, the conditions of high enough noise and independent leakages described in introduction have to be fulfilled.

Definition 1. *Walsh transform of a Boolean vector S:*

$$W_S(a,b) := \sum_{x \in \mathbb{F}_{2^n}} (-1)^{<a,x>+<b,S(x)>}.$$

Definition 2. *Walsh spectrum of a Boolean vector S:*

$$\Omega_S = \{W_S(a,b) | a,b \in \mathbb{F}_{2^n}, (a,b) \neq (0,0)\}.$$

The smaller is $\max(\Omega_S)$, the stronger is the S-box regarding linear cryptanalysis.

Differential profile. The second well-known family of statistical attacks is differential cryptanalysis [5]. As for linear cryptanalysis, we consider all non-zero differentials and their probabilities (up to a factor 2^{-n}).

Definition 3. *Differential spectrum of a Boolean vector S:*

$$\Delta_S = \{\#\{X | S(X+a) = S(X)+b\} | a,b \in \mathbb{F}_{2^n}, (a,b) \neq (0,0)\}.$$

The smaller is $\max(\Delta_S)$, the strongest is the S-box regarding differential cryptanalysis.
If $\max(\Delta_S) = d$, the S-box is said to be differentially d-uniform.

Algebraic degree. Although the tools for analyzing algebraic attacks are not as advanced as for linear and differential attacks, the algebraic degree is generally considered as a good indicator of security. Moreover, having a non-maximal algebraic degree allows distinguishing a function from a random one. For any Boolean function, the algebraic degree can be defined as follows.

Definition 4. *Algebraic degree of a boolean function f. A Boolean function f can be uniquely represented using its Algebraic Normal Form (ANF):*

$$f(x) = \sum_{u \in \mathbb{F}_2^n} a_u x^u.$$

The algebraic degree of f is defined as:

$$\deg(f) = \max_{u \in \mathbb{F}_2^n} \{\mathrm{Hw}(u), a_u \neq 0\}.$$

Where Hw *is denotes the Hamming weight function.*

Definition 5. *Algebraic degree of a Boolean vector S. The algebraic degree of a vector is defined as the maximum degree of its coordinates:*

$$\deg(S) = \max_{0 \leq i < n} \deg(f_i).$$

A.3 Performance evaluation metrics

Masking an implementation implies performance overheads, both in terms of number of operations to perform and randomness to generate. As previously mentioned, linear operations in an d-th order secure block cipher execution simply have to be performed $d+1$ times (i.e. for each share independently). Hence, it is generally the cost of the non-linear operations that dominates in the performance evaluation of masking. In particular, there are two main criteria that can be used to evaluate how friendly is an S-box regarding Boolean masking. First, *the number of multiplications* directly matters, as described in Algorithm 1. Second, one also has to pay attention to any operation (even XORs) performed on pairs of dependent variables. In order to maintain the d-th order security, it is required that the masks of these dependent variables are kept independent, which can be achieved by refreshing the shares (i.e. XORing them with new random variables). As the generation of many random bytes can become expensive in low-cost devices, the *number of additional random masks* required to execute the S-box securely also has to be counted as a performance metric.

Example 4. In [54], Rivain and Prouff compute the inverse in $GF(2^8)$ using 4 multiplications and need to refresh the mask 2 times. As a result, they require $2d^2 + 4d$ random bytes, $4d^2 + 8d + 4$ field multiplications and some linear transformations to compute the AES S-box in a d-th order secure manner.

B Monomials in $GF(2^8)$

Table 3: Masking cost and security properties of S-boxes $S(X) = X^e$.

e	# mul.	$deg(S)$	max Δ_S	max Ω_S	e	# mul.	$deg(S)$	max Δ_S	max Ω_S
1	0	1	256	256	**37**	**2**	**3**	**6**	**64**
7	**2**	**3**	**6**	**64**	43	3	4	30	96
11	2	3	10	64	47	3	5	16	48
13	2	3	12	64	53	3	4	16	64
19	2	3	16	48	59	3	5	12	64
23	3	4	16	64	61	3	5	16	64
29	**3**	**4**	**10**	**64**	91	3	5	16	32
31	3	5	16	32	**127**	**4**	**7**	**4**	**32**

C Linear transforms specifications

$$M_1 = \begin{pmatrix} 0&1&1&0&1&1&1&0\\ 0&0&1&1&0&0&1&1\\ 1&1&0&1&0&1&1&0\\ 0&0&0&0&1&0&0&0\\ 1&0&1&1&0&1&1&0\\ 0&0&0&1&0&0&0&1\\ 1&0&0&0&1&0&1&0\\ 1&0&0&0&1&1&1&0 \end{pmatrix}, \quad M_2 = \begin{pmatrix} 1&0&0&0&0&1&0&1\\ 1&0&0&1&0&0&1&1\\ 0&0&1&1&0&0&1&0\\ 0&0&1&1&1&0&0&0\\ 1&0&1&0&0&0&1&1\\ 1&1&1&1&1&0&1&1\\ 0&1&0&1&1&0&0&1\\ 0&1&0&0&0&0&0&0 \end{pmatrix}, \quad M_3 = \begin{pmatrix} 1&1&1&1&1&1&0&0\\ 1&1&0&0&1&1&1&1\\ 1&0&1&1&1&1&1&1\\ 1&1&0&1&1&1&0&1\\ 1&0&1&0&1&0&1&0\\ 1&0&0&1&1&0&1&0\\ 1&0&0&0&0&0&1&1\\ 1&1&1&1&0&1&1&0 \end{pmatrix}.$$

D 8-bit S-boxes from 4-bit ones

Table 4: Comparison of $GF(2^8)$ S-box built from $GF(2^4)$ S-box.

	Ex. 1	Ex. 2	PICARO	KHAZAD	ICEBERG
Permutation	yes	yes	no	yes	yes
# mul.	4	4	4	18	18
$deg(S)$	7	6	4	7	7
max Δ_S	10	8	4	8	8
max Ω_S	64	64	68	64	64

E Wire crossing permutations

In this appendix we argue why the approach investigated in Section 3 cannot be successful when considering wire-crossing permutations. For this purpose, let us consider such a permutation acting on 128 bits and 8-bit S-boxes (the following reasoning identically applies to any other choice of parameters). The parallel approach consists in applying 16 S-boxes in one round while the serial approach boils down to apply one S-box per round for a larger number of rounds. As a result, we directly have that at least 16 serial rounds are required to obtain a security similar to the one of a single parallel round (if less than 16 rounds are performed, then at least one output bit will be equal to an input bit due to the wire-crossing permutation). Worse, if the permutation is chosen such that each bit has passed through an S-box after 16 serial rounds, then the 16 groups of 8 bits can be computed independently. In other words, the whole cipher would be the concatenation of sixteen 8-bit ciphers in this case, and we would at least need 17 rounds to obtain a security level similar to the parallel approach. The same kind of observation holds when applying more than one S-box per round.

F How many S-boxes per round?

	NrSbox	NrNlin	NrDiff
1 S-box	3	2	4
4 S-boxes, 1 line	2	1	3
8 S-boxes, 2 lines	2	1	3
4 S-boxes, 1 column	3	1	3
4 S-boxes, 1 diagonal	2	2	3
4 S-boxes, 1 per column	2	2	3
4 S-boxes, Square	3	2	4

Table 5: Diffusion properties for different $\texttt{SB}^*$ configurations. Symmetric configurations provide the same results.

G Round constants

The round constants addition is limited to the first state row. Constants can be generated "on-the-fly" according to $\{i, i, i, i << 3\}$, where i is the round index and $<<$ the left shift operator.

H S-box

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	B2	E5	5E	FD	5F	C5	50	BC	DC	4A	FA	88	28	D8	E0	D1
10	B5	D0	3C	B0	99	C1	E8	E2	13	59	A7	FB	71	34	31	F1
20	9F	3A	CE	6E	A8	A4	B4	7E	1F	B7	51	1D	38	9D	46	69
30	53	E	42	1B	F	11	68	CA	AA	6	F0	BD	26	6F	0	D9
40	62	F3	15	60	F2	3D	7F	35	63	2D	67	93	1C	91	F9	9C
50	66	2A	81	20	95	F8	E3	4D	5A	6D	24	7B	B9	EF	DF	DA
60	58	A9	92	76	2E	B3	39	C	29	CD	43	FE	AB	F5	94	23
70	16	80	C0	12	4C	E9	48	19	8	AE	41	70	84	14	A2	D5
80	B8	33	65	BA	ED	17	CF	96	1E	3B	B	C2	C8	B6	BB	8B
90	A1	54	75	C4	10	5D	D6	25	97	E6	FC	49	F7	52	18	86
A0	8D	CB	E1	BF	D7	8E	37	BE	82	CC	64	90	7C	32	8F	4B
B0	AC	1A	EA	D3	F4	6B	2C	FF	55	A	45	9	89	1	30	2B
C0	D2	77	87	72	EB	36	DE	9E	8C	DB	6C	9B	5	2	4E	AF
D0	4	AD	74	C3	EE	A6	F6	C7	7D	40	D4	D	3E	5B	EC	78
E0	A0	B1	44	73	47	5C	98	21	22	61	3F	C6	7A	56	DD	E7
F0	85	C9	8A	57	27	7	9A	3	A3	83	E4	6A	A5	2F	79	4F

I Block cipher Zorro: schematic view

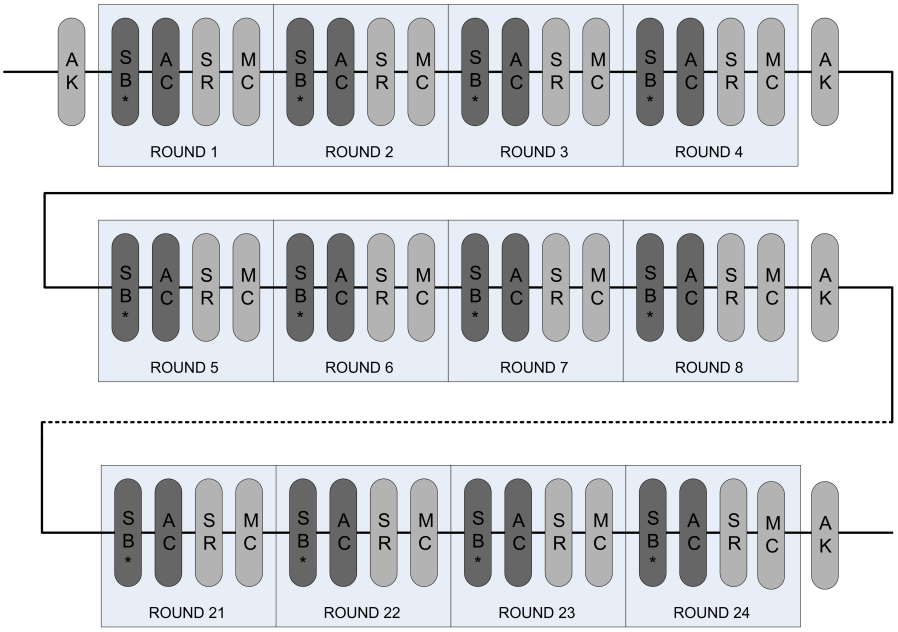


Fig. 6: Block cipher Zorro: light gray operations are AES-like, dark gray ones are new.

J Exhaustive Search for the Truncated Differential

In this appendix we investigate the number of rounds for which a truncated pattern with no difference in the first row (see Figure 2) can be found. Instead of testing all the 2^{96} possible input differences at once, we will process column by column. The main idea is that when considering a column with only three input active bytes, given two of these bytes differences, there exists one and only one byte difference for the third active byte such that the output of MC applied to this column will have a 0 difference on the first row. We easily derive from this fact that there exists 2^{16} column differential patterns having no difference in the first byte before and after the application of MC. At this point we could form the $(2^{16})^4$ differentials and test them to see what is the maximum number of rounds for which the truncated pattern is preserved. This would cost 2^{64} which is still too large and thus we will try to reduce the complexity by determining the differentials for which the truncated pattern is preserved after 2 rounds before launching any exhaustive search. We denote by $c_{0...3}$ the differences in the columns at the beginning of the second round and by $c_i^{0...3}$ the four bytes of differences from column c_i. To compute the differential obtained in column i after the round 2 we have to know the 4 bytes of the form c_{i+j}^j (due to SR). Since we know that for any i, $c_i^0 = 0$, then computing the differential only requires the knowledge of 3 bytes. Moreover, since we are looking for differentials preserving the truncated pattern, 2 bytes determine the value of the third one. The idea is then to use a hash table to match couples of columns. More precisely: (i) for any of the 2^{32} differential values for (c_0, c_1) determine the values of c_2^3 and c_3^1 such that the columns 2 and 3 will have zeros in the first coordinate after MC (ii) store in a hash table the 2^{32} corresponding values $(c_0^3, c_1^1, c_2^3, c_3^1)$ (iii) for any of the 2^{32} differential values for (c_2, c_3) determine the values of c_0^3 and c_1^1 such that the columns 0 and 1 will have zeros in the first coordinate after MC (iv) match these values with the ones in the hash table and keep the matching tuples. The tuples obtained correspond to differentials such that the truncated pattern remain valid after 2 rounds. The expected number of such

tuples is 2^{32} times 2^{32} for the number of configurations times 2^{-32} for the probability that two configurations match that is 2^{32}, that can be obtained with a time complexity of 2^{32}. We now have a small enough number of differentials to test. Eventually, we attached an additional third round to all these differentials and we could not find any one corresponding to the truncated pattern after these 3 rounds.

K Meet-in-the-middle path for 12 rounds

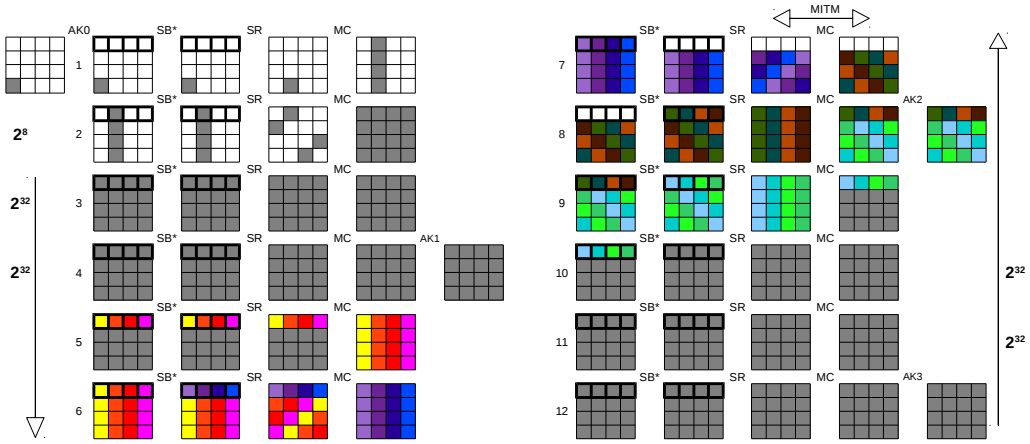


Fig. 7: Representation of the 12 rounds scenario

L Rebound-attack-like analysis

Rebound attacks were introduced in [41] mainly for distinguishing the compression function of AES-like hash functions from random ones and have found many applications and targets since. Besides being applied to hash functions, they have also been used for deducing some non-desired properties from block ciphers. They have been one of the most used attacks since the SHA-3 competitions and we have used them to study the security of our construction in the known-key setting. As expected, the number of rounds that we were able to analyze is larger than for the classical AES setting (where the best rebound like distinguisher works for 8 rounds). In a first time we detail the rebound distinguishers applying to `Zorro` then, we provide complexities of generic distinguishers (in other words, bounds on the rebound distinguisher complexities).

Rebound distinguishers. Because of the structure of `Zorro` (and more precisely because a full active state only involves 4 S-boxes), the rebound attack will allow us to solve more rounds of the differential path with rebound-like techniques. The inbound phase that will be detailed here for 4 full-active state rounds can indeed be performed with up to 7 full-active rounds as shown in Table 6 (while only 3 such rounds are handled for AES [31]). We did not consider less than 4 rounds in the inbound since the lack of freedom degrees implies that no solutions can be found, and did not consider more than 7 rounds since the total complexity would exceed the generic complexity as it will be shown later (and is implicit in Table 7). Note that the 2^{14} difference between time and total time complexities in Table 7 comes from the outbound cost.

We now consider the rebound analysis for the particular case of 4 full-active rounds in the middle of the differential path, and we detail how to solve its inbound phase. For more rounds in the middle, the analysis is very similar. An example with four fully active rounds in the middle and 5+5 rounds of outbound is illustrated in Figure 8, where gray squares are active squares. The inbound phase covers, in this example, rounds from the end of round 5 to the state before `MC` in round 11. First we choose a difference for the beginning of round

Inbound rounds	Time complexity	Memory complexity	Total time complexity
4 (+2)	2^{29}	2^{28}	2^{43}
5 (+2)	2^{43}	2^{42}	2^{57}
6 (+2)	2^{57}	2^{56}	2^{71}
7 (+2)	2^{93}	2^{70}	2^{107}

Table 6: Average cost of finding one solution for an inbound with 4 to 7 full-active states.

5, as well as the difference transition in the S-box of round 6 which will completely determine the difference in the end of round 6. Once we have found a solution for the inbound part, these conditions will allow us to have a probability of obtaining a difference on just one byte with a probability of roughly 2^{-7} for the first rounds (we just have to satisfy the difference transition that we have imposed). The same holds for the end of the inbound: being able to determine the difference in state 10 before the MC application. This means that in total, we will need to obtain 2^{14} solutions for the inbound part so as to obtain one for the total number of rounds. For the $\texttt{SB}^*$ transition in round seven (respectively round 10), we guess the 2^{28} possible differences in the output (respectively input) of the S-boxes. That means that before the MC of the 8th round, we know the differences of the last 3 rows. But we also fixed the differences in the 3 last rows after this MC. The probability that a pair of such configurations (one after and one before) are compatible with the MC transformation is 2^{-64}. Since we have 2^{28+28} such pairs, we have a probability of 2^{-8} of obtaining a valid one. This means that we will have to repeat the procedure from the beginning around 2^8 times, but once the MC transition from the 8th round is verified, we will obtain 2^8 solutions for the values of the $\texttt{SB}^*$ transitions in rounds 7 and 10 while the remaining parts will be completely determined. So the average cost of one solution is 2^{28+1} in time and 2^{28} in memory. Concerning the outbound part, its probability is 2^{-7*2}, since we can already force that three out of the four differences allow the $4 \to 1$ or the $1 \to 4$ transitions during the inbound part.

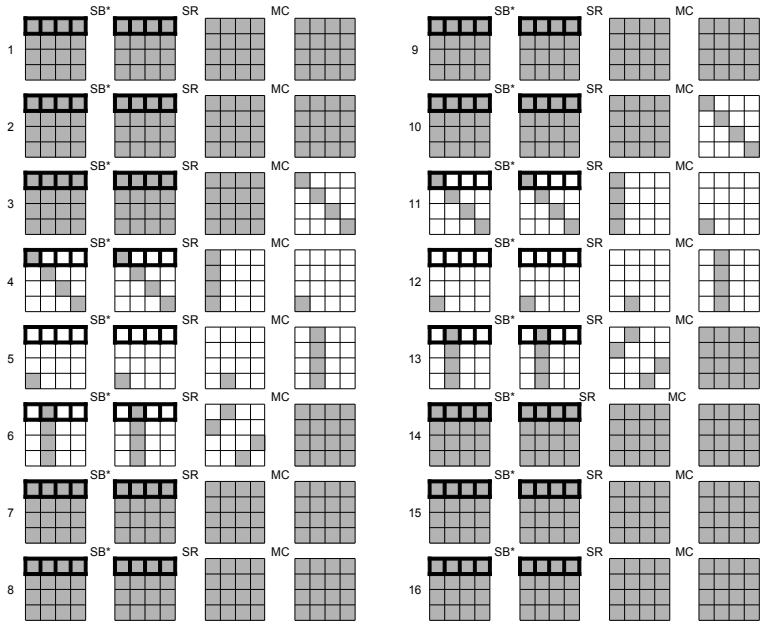


Fig. 8: Illustration of the rebound analysis (AddKey and AddConstant operations have not been represented on the figure, as the key is supposed to be known and consequently, do not modify the differential path).

Generic distinguishers. Table 7 shows the costs for the generic distinguishers to obtain the same differences in the input and the output as the ones of rebound distinguishers, depending on how many outbound rounds are considered in each direction. The aim of the rebound distinguishers, depending on its number of outbound rounds then, is the find solutions for the path with better complexity than the associated generic distinguisher. Such costs can be obtained with the limited birthday technique as described in [27], where n is the size of the state in bits (128 in our case), IN is the size of the subset of the input difference, and OUT the size of the subset of the output one:

$$\max\left\{\min\left\{\sqrt{2^n/IN}, \sqrt{2^n/OUT}\right\}, 2^n/(IN \cdot OUT)\right\}.$$

Table 7: Generic distinguisher complexities for different number of backward+forward rounds outside the inbound.

Rounds	IN	OUT	Generic Complexity
6+6	2^{127}	2^{127}	$2^{0.5}$
5+5	2^{99}	2^{99}	$2^{14.5}$
4+4	2^{71}	2^{71}	$2^{28.5}$
4+3	2^{71}	2^{43}	$2^{28.5}$
3+3	2^{43}	2^{43}	$2^{42.5}$
3+2	2^{43}	2^{15}	2^{70}
2+2	2^{15}	2^{15}	2^{88}
2+1	2^{15}	2^{8}	2^{105}
1+1	2^{8}	2^{8}	2^{112}

Conclusion and discussion. From the previous tables, we deduce that the best distinguisher that we can build for `Zorro` works on 2+6+2+2=12 rounds or on 3+5+2+2=12 rounds, so for 6 rounds in the inbound and 2+2 outbound rounds, or for 5 inbound rounds plus 3+2 outbound rounds (as the complexities are 2^{71} compared to 2^{88} and 2^{57} compared to 2^{70} respectively). The number of rounds included in the inbound part might be improved but we still have a large security margin against possible improvements for this type of attacks. Considering sparse differential paths as in [55] does not seem promising as the main interest of this approach is to reduce the complexity of the outbound part which is already minimal in our case.

How to Improve Rebound Attacks*

María Naya-Plasencia†

FHNW, Windisch, Switzerland

Abstract. Rebound attacks are a state-of-the-art analysis method for hash functions. These cryptanalysis methods are based on a well chosen differential path and have been applied to several hash functions from the SHA-3 competition, providing the best known analysis in these cases. In this paper we study rebound attacks in detail and find for a large number of cases that the complexities of existing attacks can be improved.
This is done by identifying problems that optimally adapt to the cryptanalytic situation, and by using better algorithms to find solutions for the differential path. Our improvements affect one particular operation that appears in most rebound attacks and which is often the bottleneck of the attacks. This operation, which varies depending on the attack, can be roughly described as *merging* large lists. As a result, we introduce new general purpose algorithms for enabling further rebound analysis to be as performant as possible. We illustrate our new algorithms on real hash functions. More precisely, we demonstrate how to reduce the complexities of the best known analysis on four SHA-3 candidates: JH, Grøstl, ECHO and LANE and on the best known rebound analysis on the SHA-3 candidate Luffa.

Keywords: hash functions, SHA-3 competition, rebound attacks, algorithms

1 Introduction

The rebound attack is a recent technique introduced in [13] by Mendel *et al.* It was conceived to analyze AES-like hash functions (like Grøstl [7] in [14, 8, 15], Echo [2] in [14, 8, 17], Whirlpool [1] in [11]). A rebound attack is composed of two parts: the inbound phase and the outbound phase. The aim of the inbound phase is to find, at a low cost, a large number of pairs of values that satisfy a part of a differential path that would be very expensive to satisfy in a probabilistic way. The outbound phase then uses these values to perform an attack.

This technique has been applied to other algorithms with inner permutations which are not AES-like; for instance it has been applied to JH [20] (reduced to 22 rounds) in [16] and Luffa [4] (reduced to 7 rounds) in [10]; both of those hash functions use Sboxes of size 4×4 and have a linear part in which the mixing is done in a very different way than in the AES. The hash function LANE [9], which includes several AES states, each treated by the AES round transformation, and a different transformation for mixing these states has also been analysed in [12, 21] using rebound attacks.

*This is the extended version of the article published at CRYPTO 2011.

†Supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center of the Swiss National Science Foundation under grant number 5005-67322.

In these cryptanalysis results, the rebound attack technique needs to be refined and adapted to each case, but all of them follow the same scheme: first find a differential path, then find solutions verifying this differential path. This paper focuses on optimizing the latter part. In all the previously mentioned cryptanalysis, that part involves enumerating, from a very large set of possible candidates represented as a cross product of lists, all those that verify a given relation. We call this operation *"merging" the lists*. The merging problem can be described more formally as follows.

Merging problem with respect to t: Let t be a Boolean function taking N k-bit words as input, *i.e.* $t : (\{0,1\}^k)^N \rightarrow \{0,1\}$. Let $L_1, \ldots, L_N$ be N given lists of k-bit words drawn uniformly and independently at random from $\{0,1\}^k$. We assume that the probability over all N-tuples X in $L_1 \times \ldots \times L_N$ that $t(X) = 1$ is P_t. For any given function t and any given N-tuple of lists $(L_1, \ldots, L_N)$ the *merging problem* consists in finding the list $\mathcal{L}_{sol}$ of all $X \in L_1 \times \ldots \times L_N$ satisfying $t(X) = 1$. We call this operation *merging* the lists $L_1, \ldots, L_N$ to obtain $\mathcal{L}_{sol}$.

It is assumed that the image of a given input under t can be easily computed. In the following, the size of a list L is denoted by $|L|$. A brute force method for solving this problem therefore consists in enumerating all the $|L_1| \times \ldots \times |L_N|$ inputs, in computing t on all of them and in keeping the ones verifying $t = 1$. Note that, in the lack of any additional information on t, it is theoretically impossible to do better. However, in practice, the function t often has a set of properties which can be exploited to optimize this approach. We aim at reducing the number of candidates which have to be examined, in some cases by a preliminary sieving similar to the one used in [5]. This paper presents such optimization techniques, that, when applied to most of the rebound attacks published on the SHA-3 candidates, yield significant improvements in the overall time and/or memory complexities of the attack, as shown on Table 1. In this table we can see that we have considered the best existing attacks against four hash functions and the best rebound attack on a fifth (two of them are finalists and two are second-round candidates of the SHA-3 competition), where by best attack we denote the one on the highest number of rounds. We have been able to improve their complexities by scrutinizing the original attack and finding a more efficient algorithm for obtaining the solutions for the differential path. Most of the time the improvement relies on a better *merging* of the lists, and sometimes it is due to the use of more adequate conditions in the general algorithm. Let us recall here that the aim is to find *all* the N-tuples that verify $t = 1$ for a complex function t, which is significantly different from finding just *one* (or few) of them for a linear t such as in [19, 18, 6, 3]. As in the previous rebound analysis, we will work throughout the paper with average values in the probabilistic cases.

In Section 2, we define *Problem 1* that corresponds to functions t with a particular form, and we propose three generic algorithms to solve it. These 3 algorithms have different optimal scenarios. Some examples of applications are given. In Section 3 we define *Problem 2* and propose the *stop-in-the-middle algorithms* for solving it. We also present two concrete algorithms in this family applied to the scenarios of ECHO and Lane. In Section 4 we show

how applying these algorithms combined with an appropriate definition and decomposition of the problem in each case, allows us to improve the complexities of the best known rebound attacks on 5 SHA-3 candidates.

Table 1. Improvements on best known attacks. The highlighted values are the improved complexities. For Luffa we consider the best known rebound attack where the complexities presented in the second row have already been obtained in [10] by a dedicated algorithm similar to our general approach.

Hash function	SHA3 Round	Best Known Analysis	Rounds / total	Previous Time	Previous Memory	Ref.	This paper Time	This paper Memory
JH	Final	semi-free-start coll.	16 / 42	2^{190}	2^{104}	[16]	$\mathbf{2^{97}}$	$\mathbf{2^{97}}$
JH		semi-free-start near coll.	22 / 42	2^{168}	$2^{143.70}$	[16]	$\mathbf{2^{96}}$	$\mathbf{2^{96}}$
Grøstl-256	Final*	(compr. function property)	10 / 10	2^{192}	2^{64}	[15]	$\mathbf{2^{182}}$	2^{64}
Grøstl-256		(internal permutation dist.)	10 / 10	2^{192}	2^{64}	[15]	$\mathbf{2^{175}}$	2^{64}
Grøstl-512		(compr. function property)	11 / 14	2^{640}	2^{64}	[15]	$\mathbf{2^{630}}$	2^{64}
ECHO-256	2^{nd}	internal permutation dist.	8 / 8	2^{182}	2^{37}	[17]	$\mathbf{2^{151}}$	2^{67}
Luffa	2^{nd}	semi-free-start coll.	7 / 8	2^{132}	$2^{68.8}$	[10]	$\mathbf{2^{112.9}}$ $\mathbf{(2^{104})}$	$2^{68.8}$ (2^{102})
LANE-256	1^{st}	semi-free-start coll.	6+3 / 6+3	2^{96}	2^{88}	[12]	$\mathbf{2^{80}}$	$\mathbf{2^{66}}$
LANE-512		semi-free-start coll.	8+4 / 8+4	2^{224}	2^{128}	[12]	2^{224}	$\mathbf{2^{66}}$

* The Grøstl analysis does not apply after the final round tweak.

Besides the results in Table 1, the main interest of this paper is to present a general framework for improving rebound attacks. We introduce several new algorithms that considerably improve the overall effectiveness when the attack needs to *merge* large lists. We provide a formal definition of the field of application of those algorithms, and describe them as a set of constraints on t, in hope that designers of rebound attacks will be able to easily *identify* scenarios where one of these algorithms, or variants, may be applied. This was motivated by our own research path, when we realized that a generalization of the techniques leveraged in specific cases allowed us to find similar improvements in almost all of the rebound attacks that we have studied so far.

2 When t is Group-Wise

In some cases we can considerably reduce the complexity of the *merging* problem by redefining it into a more concrete one. We consider here a very common case that will appear in many rebound scenarios, as we will later show with the examples. This case corresponds to a function t that can be decomposed in smaller functions. After introducing the general problem, we will illustrate it with an example. Though we preferred to state the problem

in full generality for any possible N, in the concrete rebound examples that we studied, the number of lists N was either 2, 4 or 6. Also, the elements of each list can be decomposed in sets of small size s, where s is typically the size of the involved Sbox; and z is the number of such sets involved [1] in the function t.

Problem 1: Let $L_1, \ldots, L_N$ be N lists of size $2^{l_1}, \ldots, 2^{l_N}$ respectively, where the elements are drawn uniformly and independently at random from $\{0,1\}^k$.
Let t be a Boolean function, $t : \left(\{0,1\}^k\right)^N \rightarrow \{0,1\}$ for which there exists $N' < N$, an integer z and some triples of functions $t_j : \{0,1\}^{2s} \rightarrow \{0,1\}$, $f_j : (\{0,1\}^k)^{N'} \rightarrow \{0,1\}^s$ and $f'_j : (\{0,1\}^k)^{(N-N')} \rightarrow \{0,1\}^s$ for $j = 1, \ldots, z$ such that, $\forall\, (\boldsymbol{x_1}, \ldots, \boldsymbol{x_N}) \in L_1 \times \ldots \times L_N$:

$$t(\boldsymbol{x_1}, \ldots, \boldsymbol{x_N}) = 1 \quad \Leftrightarrow \quad \begin{array}{l} \forall j = 1, \ldots, z, \\ \begin{cases} t_j(v_j, v'_j) = 1 \\ \text{with } v_j = f_j(\boldsymbol{x_1}, \ldots, \boldsymbol{x_{N'}}) \\ \text{and } \ v'_j = f'_j(\boldsymbol{x_{N'+1}}, \ldots, \boldsymbol{x_N}) \end{cases} \end{array}$$

Let P_t be the probability that $t = 1$ for a random input.
Problem 1 consists in *merging* these N lists to obtain the set $\mathcal{L}_{sol}$, of size $P_t 2^{\sum_{i=1}^{N} l_i}$, of all N-tuples of $(L_1 \times \ldots \times L_N)$ verifying $t = 1$.

Reduction from N to 2: For any $N \geq 2$ *Problem 1* can be reduced to an equivalent and simplified problem with $N = 2$, *i.e.* merging two lists L_A and L_B, which consist of elements in $(\{0,1\}^s)^z$ corresponding to $\boldsymbol{x_A} = \boldsymbol{v} = (v_1, \ldots, v_z)$ and $\boldsymbol{x_B} = \boldsymbol{v'} = (v'_1, \ldots, v'_z)$, with respect to the function $\boldsymbol{x_A}, \boldsymbol{x_B} \mapsto \Pi_{j=1}^{z} t_j(v_j, v'_j)$. The reduction is performed as follows:

1. Build a table T_A^* of size $2^{\sum_{i=1}^{N'} l_i}$ storing each element $\boldsymbol{e_A} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_{N'}})$ of $L_1 \times \ldots \times L_{N'}$, indexed[2] by the value of $(f_1(\boldsymbol{e_A}), \ldots, f_z(\boldsymbol{e_A}))$, *i.e.* $(v_1, \ldots, v_z)$. Store the corresponding $(v_1, \ldots, v_z)$ in a list L_A. Note that several $\boldsymbol{e_A}$ may lead to the same value of $(v_1, \ldots, v_z)$.
2. Build a similar table T_B^* of size $2^{\sum_{i=N'+1}^{N} l_i}$ storing each element $\boldsymbol{e_B} = (\boldsymbol{x_{N'+1}}, \ldots, \boldsymbol{x_N})$ of $L_{N'+1} \times \ldots \times L_N$, indexed by $(f_1(\boldsymbol{e_B}), \ldots, f_z(\boldsymbol{e_B}))$, *i.e.* $(v'_1, \ldots, v'_z)$. Store $(v'_1, \ldots, v'_z)$ in a list L_B.
3. *Merge* L_A and L_B with respect to $\Pi_{j=1}^{z} t_j$ and obtain $\mathcal{L}_{sol}$.
4. Build $\mathcal{L}_{sol}^*$ by iterating over each pair $((v_1, \ldots, v_z), (v'_1, \ldots, v'_z))$ of $\mathcal{L}_{sol}$, and adding the set of all $(\boldsymbol{x_1}, \ldots, \boldsymbol{x_{N'}}, \boldsymbol{x_{N+1}}, \ldots, \boldsymbol{x_N}) \in T_A^*\left[(v_1, \ldots, v_z)\right] \times T_B^*\left[(v'_1, \ldots, v'_z)\right]$. $\mathcal{L}_{sol}^*$ is the solution to the original problem.

[1] Sometimes, elements are only partially involved in t.

[2] Here and in the following sections we can use standard hash tables for storage and lookup in constant time, since the keys are integers.

Let $2^{T_{\text{merge}}}, 2^{M_{\text{merge}}}$ be the time and memory complexities of step 3. The total time complexity of solving *Problem 1* is $\mathcal{O}(sz2^{\sum_{i=1}^{N'} l_i} + sz2^{\sum_{i=N'+1}^{N} l_i} + 2^{T_{\text{merge}}} + P_t 2^{\sum_{i=1}^{N} l_i})$ where the last term comes from the fact that only the N-tuples satisfying $t = 1$ are examined at step 4 because of the sieve applied at step 3. The proportion of such tuples is then P_t. The memory complexity [3] is $\mathcal{O}((zs + N'k)2^{\sum_{i=1}^{N'} l_i} + (zs + (N - N')k)2^{\sum_{i=N'+1}^{N} l_i} + 2^{M_{\text{merge}}} + P_t 2^{\sum_{i=1}^{N} l_i})$, where the last term appears only when the solutions must be stored.

Using the brute force approach, $2^{T_{\text{merge}}}$ would be $2^{l_A+l_B}$ where 2^{l_A} (respectively 2^{l_B}) denotes the size of L_A (L_B), and $2^{M_{\text{merge}}}$ would be negligible. We present in the following sections some algorithms for solving *Problem 1* considering $N = 2$ with L_A and L_B, that provide better complexities than the brute force approach. Note that the roles of L_A and L_B are assigned by choice to obtain the best overall complexity. Those algorithms can be applied for obtaining a smaller $2^{T_{\text{merge}}}$ when $N > 2$.

2.1 Basic Algorithm for Solving *Problem 1*: Instant Matching

As s is typically very small we can enumerate the solutions (v_j, v'_j) of $t_j(v_j, v'_j) = 1$ and store them in tables T_j of size $\leq 2^{2s}$, indexed by v'_j. This costs $\mathcal{O}(z \cdot 2^{2s})$ in time and memory. We propose in Fig. 1 a first algorithm for solving *Problem 1*, which has lower complexity than the brute-force approach. Although being the simplest algorithm presented in this paper, it has not been applied in critical steps of some of the previously mentioned attacks, though it could yield significant improvements.

Fig. 1 Instant matching algorithm.

Require: Two lists L_A, L_B and a Boolean function t as described in *Problem 1*.
Ensure: The returned list $\mathcal{L}_{sol}$ will contain all elements of $L_A \times L_B$ verifying t.

```
1: for j from 1 to z do
2:    for all (v_j, v'_j) in {0,1}^s × {0,1}^s do
3:       if t_j(v_j, v'_j) = 1, then add v_j to T_j[v'_j].
4: for each (v'_1, ..., v'_z) ∈ L_B do
5:    Empty L_aux.
6:    for j from 1 to z do
7:       if T_j[v'_j] is empty, then go to 4.
8:    Add all tuples (v_1, ..., v_z) verifying ∀j v_j ∈ T_j[v'_j] to L_aux.
9:    for each (v_1, ..., v_z) in L_aux do
10:      if (v_1, ..., v_z) ∈ L_A then
11:         Add (v_1, ..., v_z, v'_1, ..., v'_z) to L_sol.
12: Return L_sol.
```

Let 2^{-p_j} be the probability over all pairs (v_j, v'_j) that $t_j(v_j, v'_j) = 1$. The relationship between t and the $(t_j)_{1 \leq j \leq z}$ implies that $\sum_{j=1}^{z} p_j = -\log_2(P_t)$ where P_t is the probability

[3] The first two terms, corresponding to the storage of T_A^* and T_B^* could be avoided if they were the bottleneck by slightly increasing the time complexity by a factor of 2.

that $t = 1$.
Let us determine the average size of L_{aux}. The average size of $T_j[v'_j]$ over all v'_j is 2^{s-p_j}. Then the average size of L_{aux} is $2^{zs-\sum_{j=1}^{z} p_j} = P_t 2^{zs}$. It follows that the time complexity of the algorithm is $\mathcal{O}(z2^s + zP_t 2^{l_B+zs})$ and is proportional to the product of the size of L_B by the average size of [4] L_{aux}. The memory complexity is $\mathcal{O}(z2^s + 2^{l_A} + 2^{l_B} + P_t 2^{l_A+l_B})$. In some cases, the last term can disappear, namely if we do not need to store the list $\mathcal{L}_{sol}$, but just use each solution as soon as it is obtained. The same way, the list L_B does not need to be stored, if it can be given on the fly.

We now describe a concrete example of application of the *instant-matching algorithm* in a case included in a particular rebound attack, improving its complexity. In Appendix A we provide two more examples where it clearly appears that identifying and isolating the most appropriate problem (or problems) to solve is of major importance. These two last examples might help also to understand the role of f_j and f'_j.

Example 1: Application of the Instant Matching Algorithm We use here a case presented in the analysis of JH [16] which is the attack on 8 rounds using one inbound when the dimension of a block of bits denoted by d is 4. Here we improve step 3 of the attack, which is also the bottleneck in time complexity. Two lists are given, L_A and L_B of size $2^{24.18}$ elements each. The aim of step 3 is to *merge* those lists, *i.e.* find all pairs $(\boldsymbol{v}, \boldsymbol{v'}) \in L_A \times L_B$ verifying 10 conditions on groups of $s = 4$ bits of $(\boldsymbol{v}, \boldsymbol{v'})$.
In [16] this is solved by exhaustive search, *i.e.* all possible pairs are examined and only the ones that verify the 10 conditions are kept, which has cost $2^{48.36}$. We can improve this complexity by applying the instant-matching algorithm: first, we notice that 6 out of these 10 conditions can be written as

$$t_j(v_j, v'_j) = 1, \forall j \in \{1, \ldots, 6\},$$

where variables v_j and v'_j represent groups of differences of 4 bits. The functions t_j return 1 when the linear function of JH, L, applied to v_j and v'_j produces 4 bits out of 8 without difference in the wanted positions. Those functions t_j can be computed directly by using a precomputed table of size 2^8.
This is an instance of *Problem 1* with the parameters: $z = 6$ (corresponding to the number of relations $t_1, \ldots, t_6$), and $p_j = 3.91\ \forall j$. Hence $P_t 2^{zs} = 2^{0.09 \cdot 6} = 2^{0.54} \simeq 1.45$. The instant-matching algorithm allows us to find all pairs satisfying these 6 conditions with a complexity of $2^{27.8}$ in time and no additional memory. We then obtain $2^{24.9}$ pairs of elements that pass the first 6 conditions. To complete step 3 of the attack, we evaluate the 4 remaining conditions for each pair, for a global complexity of $2^{24.9}$.
To summarize, we were able to resolve step 3 of the attack with a time complexity of about $2^{27.8}$, improving significantly the complexity of $2^{48.36}$ given in [16].

[4] The cost of building and storing the lists $T_j[v'_j]$ is negligible.

2.2 Solving Problem 1 when $P_t 2^{zs} > 2^{l_A}$: Gradual Matching

In Fig. 2 we present an algorithm for solving *Problem 1* that is useful in cases where the average size of L_{aux} exceeds the size of L_A, *i.e.*[5] $P_t 2^{zs} > 2^{l_A}$. In this case the instant-matching algorithm has a higher complexity than the exhaustive search. This is why here, instead of directly matching the z groups that appear in relation t, we will first match the $z' < z$ ones, and next, the $z - z'$ remaining ones. We present here how to use one step of the *gradual-matching algorithm* for solving Problem 1. This algorithm reminds the method used in Example 1 where the problem is first solved with only 6 relations. But the difference is that the remaining $z - z'$ relations can also be written in the form needed for *Problem 1* and $P_t 2^{zs} > 2^{l_A}$. Let us suppose that we choose z' so that $z's < l_A$ (the best value for z' depends on the situation).

Fig. 2 Gradual matching algorithm.

Require: Two lists L_A and L_B and a function t as described in *Problem 1*.
Ensure: List $\mathcal{L}_{sol} \subset L_A \times L_B$ of all elements verifying t.
1: **for** j from 1 to z **do**
2: **for all** (v_j, v'_j) in $\{0,1\}^s \times \{0,1\}^s$ **do**
3: **if** $t_j(v_j, v'_j) = 1$, **then** add v_j to $T_j[v'_j]$.
4: **for** each $\alpha = (\alpha_1, \ldots, \alpha_{z'})$ in $(\{0,1\}^s)^{z'}$ **do**
5: Empty L_{aux}.
6: Consider the sublist $L_B(\alpha)$ of all elements in L_B with $(v'_1, \ldots, v'_{z'}) = \alpha$.
7: **for** each $(v_1, \ldots, v_{z'})$ in $T_1[\alpha_1] \times \ldots \times T_{z'}[\alpha_{z'}]$ **do**
8: add $(v_1, \ldots, v_{z'})$ to L_{aux}.
9: **for** each $\gamma = (\gamma_1, \ldots, \gamma_{z'})$ in L_{aux} **do**
10: Consider the sublist $L_A(\gamma)$ of all elements of L_A with $(v_1, \ldots, v_{z'}) = \gamma$.
11: Merge $L_A(\gamma)$ with $L_B(\alpha)$ with respect to $t' = \Pi_{j=z'+1}^{z} t_j$.
12: Add the solutions to $\mathcal{L}_{sol}$.
13: Return $\mathcal{L}_{sol}$, containing about $P_t 2^{l_A + l_B}$ elements.

Let 2^{merge} be the time complexity of *merging* once lists $L_B(\alpha)$ and $L_A(\gamma)$ as defined in Fig. 2. Since their respective average sizes are $2^{l_A - z's}$ and $2^{l_B - z's}$ the complexity of the brute force is $2^{l_A + l_B - 2z's}$. It can be improved by using one of the proposed algorithms from this section but it cannot be smaller than the size of the resulting merged list, *i.e.* $2^{l_A + l_B - 2z's - \sum_{j=z'+1}^{z} p_j}$. Now the average size of L_{aux} [6] is $\mathcal{S} = 2^{z's - \sum_{j=1}^{z'} p_j}$. Then, the time complexity of this algorithm is $\mathcal{O}(z2^s + 2^{z's}(z' + \mathcal{S}2^{\text{merge}}))$. It is worth noticing that this complexity corresponds to $z'2^{z's} + 2^{l_A + l_B - \sum_{j=1}^{z'} p_j}$ when the intermediate lists are merged by the brute force algorithm and to $z'2^{z's} + P_t 2^{l_A + l_B}$ if they are merged by an optimal algorithm. The memory complexity is $\mathcal{O}(z2^s + 2^{l_A} + 2^{l_B} + \mathcal{S} + P_t 2^{l_A + l_B})$. Again, in some

[5] When $P_t 2^{zs}$ is close to 2^{l_A} this algorithm might also outperform the instant-matching technique.

[6] Here and in the previous section, there is no need for storing L_{aux}, as each element can be treated as soon as it is obtained, but these auxiliary lists are very useful for describing the complexities.

cases, the last term can disappear, if we do not need to store the list $\mathcal{L}_{sol}$, but just use the solutions on the fly.

2.3 Time-Memory Trade-Offs when $P_t 2^{zs} > 2^{l_A}$: Parallel Matching

The *parallel-matching algorithm* improves the time complexity of the gradual-matching by a time-memory trade-off and can be applied in the same situations. It is a generalization of an algorithm proposed in [10]. As the gradual-matching algorithm this algorithm first finds elements that verify $t_j = 1$ for $j \in \{1, \ldots, z'\}$ and then, for each of them, it checks if the remaining $(z - z')$ relations are also verified. However, in this algorithm, the matching of the z' relations is done in parallel for n and m relations, so that $z' = m+n$. The motivation of choosing different variables for n and m is showing that there is no need for them to be the same when applying the algorithm.

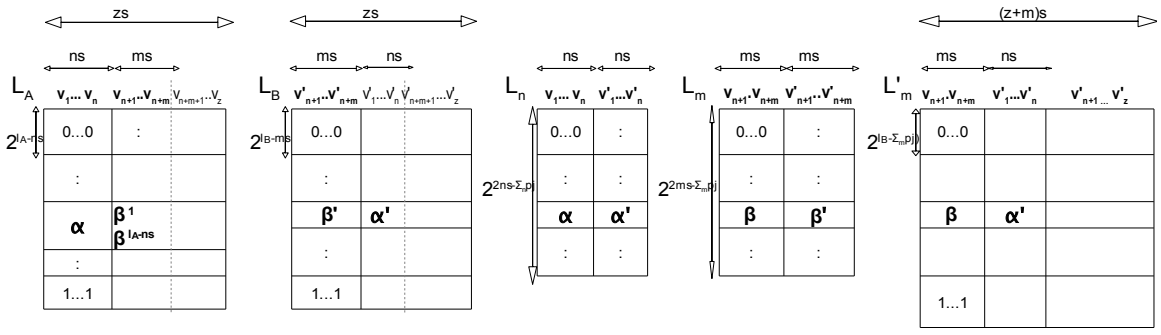


Fig. 3: Representation of the parallel-matching algorithm.

We choose n so that $n < z$, $ns < l_A$ and $ns < l_B$, and in the same way, we choose m $(n + m = z' \leq z)$. This algorithm will be explained with ordered lists, as it is more graphical and helps the understanding. However, since we can perform it with hash tables indexed by the values we want to have ordered, we do not need to take into account the logarithmic terms for ordering and searching in the final complexity. First we build the lists that we will use and that are represented in Fig. 3:

- We order the list L_A by the first n groups $(v_1, \ldots, v_n)$. L_A has $2^{l_A - sn}$ elements in average corresponding to a given value of these n groups.
- We order the list L_B by the next m groups $(v'_{n+1}, \ldots, v'_{n+m})$. L_B has $2^{l_B - sm}$ elements in average corresponding to a given value of these m groups.

– We build the list L_n of size $2^{2ns-\sum_{j=1}^{n} p_j}$ formed by all $(v_1, \ldots, v_n, v'_1, \ldots, v'_n)$ with $v_j \in T_j[v'_j]$ for all $1 \leq j \leq n$. All the elements from this list satisfy $t_j(v_j, v'_j) = 1$ for $j \in [1, \ldots, n]$.
– We build the list L_m of size $2^{2ms-\sum_{j=n+1}^{n+m} p_j}$ formed by all $(v_{n+1}, \ldots, v_{n+m}, v'_{n+1}, \ldots, v'_{n+m})$ with $v_j \in T_j[v'_j]$ for all $(n+1) \leq j \leq (n+m)$. All the elements from this list satisfy $t_j(v_j, v'_j) = 1$ for $j \in [n+1, \ldots, n+m]$.
– From L_m and L_B we build L'_m as follows: for each (β, β') in L_m, we add to L'_m all elements $(\beta, v'_1, \ldots, v'_z)$ of L_B such that $(v'_{n+1}, \ldots, v'_{n+m}) = \beta'$ and we store them ordered by the values of $(\beta, v'_1, \ldots, v'_n)$. The average size of L'_m is $2^{l_B+sm-\sum_{j=n+1}^{n+m} p_j}$. Then we perform the algorithm given in Fig. 4.

Fig. 4 Parallel matching algorithm.

```
1: for each (α, α') in L_n do
2:    for each (v_1, ..., v_z) in L_A with (v_1, ..., v_n) = α do
3:       if L'_m contains any element (v_{n+1}, ..., v_{n+m}, v'_1, ..., v'_z) starting by (v_{n+1}, ..., v_{n+m}, α') then
4:          if (v_1, ..., v_z, v'_1, ..., v'_z) satisfies the remaning (z − n − m) conditions then
5:             Add (v_1, ..., v_z, v'_1, ..., v'_z) to L_sol.
6: Return L_sol containing about P_t 2^{l_A+l_B} elements.
```

As already mentioned, the respective average sizes of L_n and L_m are $2^{l_n} = 2^{2ns-\sum_{j=1}^{n} p_j}$ and $2^{l_m} = 2^{2ms-\sum_{j=n+1}^{n+m} p_j}$, and the average size of L'_m is $2^{l_B+ms-\sum_{j=n+1}^{n+m} p_j}$. In total we will find the $2^{l_A+l_B-\sum_{j=1}^{z} p_j}$ matches that exist, with a complexity in time $\mathcal{O}(2^{l_n} + 2^{l_m} + 2^{l_A+l_B-\sum_{j=1}^{n+m} p_j} + 2^{l_A+ns-\sum_{j=1}^{n} p_j} + 2^{l_B+ms-\sum_{j=n+1}^{m} p_j})$ and $\mathcal{O}(2^{l_n} + 2^{l_m} + 2^{l_B} + 2^{l_B+ms-\sum_{j=n+1}^{m} p_j} + 2^{l_A+l_B-\sum_{j=1}^{z} p_j})$ in memory, where the last term corresponds to the storage of all solutions, not always needed. In this case, the storage of L_A is not necessary.

2.4 Example 2: Gradual Matching vs Parallel Matching

We are going to apply both previous algorithms to the analysis of Luffa presented in [10]. We are given two lists L_A and L_B of size 2^{67} and $2^{65.6}$. These lists contain elements formed by $z = 52$ groups of differences of $s = 4$ bits. List L_A contains the possible differences for the input of 52 Sboxes. List L_B contains the possible differences for the output of the same 52 Sboxes. For the j-th Sbox, the probability that one input difference can be associated to one output difference is $2^{-p_j} = 2^{-1.23}$. The average size of L_{aux} if we apply the instant-matching algorithm is then $P_t 2^{zs} = 2^{144.04}$. In this case t can be decomposed in 52 t_j, one per Sbox. So $t_j(v_j, v'_j) = 1$ if there exists $x \in \{0,1\}^s$ such that

$$\text{Sbox}(x) \oplus \text{Sbox}(x \oplus v_j) = v'_j.$$

The brute force algorithm for solving this problem has a time complexity $2^{65.6+67} = 2^{132.6}$ and a memory complexity of $2^{68.8}$. If we apply the gradual-matching algorithm with $z' = 16$

we have $\mathcal{S} = 2^{44.32}$, and we obtain the $2^{68.8}$ solutions with a time complexity of $2^{112.9}$ and the same memory as before as no additional memory is needed. If instead we apply the parallel-matching algorithm with $m = n = 13$, we can obtain the solutions with a time complexity of 2^{104} and a memory complexity of 2^{102}. Different choices of parameters allow many other time-memory trade-offs, but we just show here the one that provides the lowest time complexity, and so the highest memory needs, for contrast with the gradual matching algorithm.

3 Stop-in-the-Middle Algorithms

In this section we present another case that allows to reduce the complexity of solving the basic problem. It is described in *Problem 2*. Then, we describe the main lines of the *stop-in-the-middle algorithms*, that we use for solving *Problem 2*. Next, we present such an algorithm that solves *Problem 2* in the scenario of LANE-256. Then a more complex variant of this algorithm is applied to a ECHO-256 scenario. But we believe that, in particular, this kind of algorithms can be adapted and applied to functions that use several AES (like) states in parallel which are then merged at the end of each round. In the following, we consider a permutation F from $\{0,1\}^{sk}$ to $\{0,1\}^{sk}$ and we assume that there exist a decomposition function ϕ (respectively ψ) of the input of F (respectively the output) in k elements of $\{0,1\}^s$. These two decompositions may be different. Then, instead of the original function F we will now focus on the function $f = \psi \circ F \circ \phi^{-1}$ which is a function over $(\{0,1\}^s)^k$ (see Fig. 5). In the following (u, w) denotes the word corresponding to the concatenation of the vectors u and w.

Problem 2: Let z_A and z_B be two integers less than or equal to k . Let L_A be a list of elements in $(\{0,1\}^s)^{z_A}$ and L_B be a list of elements on $(\{0,1\}^s)^{z_B}$. The *Problem 2* consists on finding all triples (a, b, c) with $a \in L_A$, $b \in L_B$ and $c \in L_C = (\{0,1\}^s)^k$ such that

$$f(c) \oplus f(c \oplus (a, 0^{s(k-z_A)})) = (b, 0^{s(k-z_B)}),$$

where there exists the function $F_1 : (\{0,1\}^s)^k \to (\{0,1\}^s)^k$ and some permutations of $\{0,1\}^s$, $g_1, \ldots, g_k$ and $h_1, \ldots, h_k$ over $\{0,1\}^s$ such that

$$f = H \circ F_1 \circ G$$

where

$$\begin{aligned} G : \quad & (\{0,1\}^s)^k \to (\{0,1\}^s)^k \\ & (x_1, \ldots, x_k) \to (g_1(x_1), \ldots, g_k(x_k)) \end{aligned}$$

and

$$\begin{aligned} H : \quad & (\{0,1\}^s)^k \to (\{0,1\}^s)^k \\ & (x_1, \ldots, x_k) \to (h_1(x_1), \ldots, h_k(x_k)) \end{aligned}$$

It is worth noting that we assume that both decompossitions ϕ and ψ have been chosen in an appropriate way such that the z_A words of a (respectively the z_B words of b) correspond to the first words of the input state (respectively of the output state). We call

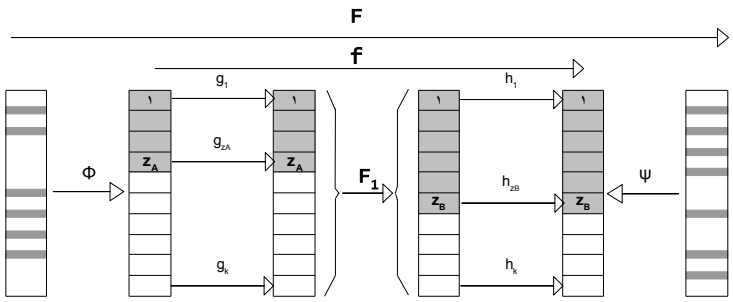


Fig. 5: Representation of F from *Problem 2*.

stop-in-the-middle algorithms those that solve *Problem 2* following the main general scheme described in Fig. 6. The complexities associated depend on the particular form of F_1, as we show in the next sections.

Fig. 6 General scheme of stop-in-the-middle algorithms.

for each b in L_B **do**
  for each $j \in [1, \ldots, z_B]$ **do**
    for each $y_j \in \{0,1\}^s$ **do**
      add $(h_j^{-1}(y_j), h_j^{-1}(y_j) \oplus h_j^{-1}(y_j \oplus b_j))$ to $L_{j,b}$.
for each a in L_A **do**
  for each $i \in [1, \ldots, z_A]$ **do**
    for each x_i in $\{0,1\}^s$ **do**
      add $(g_i(x_i), g_i(x_i) \oplus g_i(x_i \oplus a_i))$ to L_i.
  Using the previous lists L_i and $L_{j,b}$, match in the middle using F_1, *i.e.* construct the list
  $L_{aux} = \{(x, b_1, \ldots, b_{z_B}), x \in (\{0,1\}^s)^k\}$ such that
  $((F_1[g_1(x_1), \ldots, g_{z_A}(x_{z_A}), x^*)], F_1[g_1(x_1 \oplus a_1), \ldots, g_{z_A}(x_{z_A} \oplus a_{z_A}), x^*)]) =$
  $\big((h_1^{-1}(y_1), \ldots, h_{z_B}^{-1}(y_{z_B}), y^*), (h_1^{-1}(y_1 \oplus b_1), \ldots, h_{z_B}^{-1}(y_{z_B} \oplus b_{z_B}), y^*)\big)$
  for some $x^* \in (\{0,1\}^s)^{k-z_A}$ and $y^* \in (\{0,1\}^s)^{k-z_B}$.
  for all $(x, b_1, \ldots, b_{z_B})$ in L_{aux} **do**
    if $b = (b_1, \ldots, b_{z_B}) \in L_B$ **then**
      add (a, b, x) to $\mathcal{L}_{sol}$.
Return $\mathcal{L}_{sol}$.

In the cases we have studied and that we detail below, the function f is formed by several AES transformations in parallel. We then expect $2^{l_A+l_B}$ solutions, as for each $a \in L_A$ and

each $b \in L_B$ there exists one $c \in L_C$ so that the condition of *Problem 2* holds. The match-in-the-middle step is assumed to be simple due to the simple form of F_1 (typical functions F_1 are linear diffusion layers). For the same reason, L_{aux} can typically be written in a compact way, for example, in several independent lists.

3.1 Algorithm for LANE-256

Each lane of the internal state of LANE-256 is composed of two AES states. An AES state is a state of size 128 bits that can be seen as a 4×4 matrix of bytes. The AES transformations are noted: SB for SubBytes, SR for ShiftRows and MC for MixColumn. The transformation SC mixes the two AES states at the end of each round by interchanging their columns. We consider Fig. 7 that represents a part of the differential path used in [12]. In that attack it was treated as the merging of two inbounds and 2^{64} solutions were found with a complexity of 2^{96} in time and 2^{88} in memory. We consider the scheme represented in Fig. 7 where we have swapped lines and columns for a more easy intuitive understanding (so SR is applied to the columns and MC is applied to the lines).

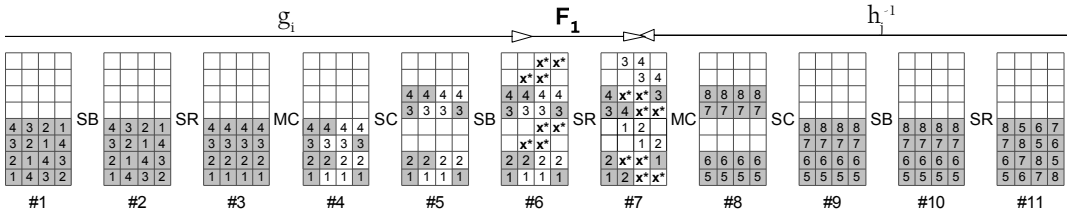


Fig. 7: Differential path associated to the first improvement on the LANE analysis.

Using the example from [12], $l_A = 32$ and $l_B = 32$ and L_C is the list of all possible input values and needs to be neither stored nor computed. We consider that the input state (respectively the output state) of the function f presented in Fig. 7 is decomposed into eight 32-bit words (*i.e.* $s = 32$ and $k = 8$). The input differences and output differences that we consider in L_A and L_B correspond to the first $z_A = z_B = 4$ 32-bit words of the state. In Fig. 7 each one of the $4 + 4 = 8$ 32-bits active word corresponds to the four active bytes with the same number written on them (1 to 4 for the four active input words and 5 to 8 for the 4 active output words).

With the algorithm described in Fig. 8 we find the 2^{64} solutions with a complexity of 2^{66} in time and 2^{65} in memory. The time complexity associated to the studied path is $z_B 2^{l_B+32} + 2^{l_A+32}$. This comes from the fact that each L_i has average size 2^{16}. Then, $L_{5,6}$ and $L_{7,8}$ have size 2^{l_B+32}. Then the size of both L^0_{aux} and L^1_{aux} is 2^{l_B} since in each we keep

Fig. 8 Algorithm for solving two inbounds of LANE-256.

Require: Function f and lists L_A and L_B of differences in #1 and #11 respectively.
Ensure: List $\mathcal{L}_{sol} = \{(a, b, c)$ such that $f(c \oplus (a, 0^{s(k-z_A)})) \oplus f(c) = (b, 0^{s(k-z_B)})\}$.

for each b in L_B **do**
  for i from 5 to 8 **do**
    for each $y \in \{0,1\}^{32}$ **do**
      if $h_i^{-1}(y) \oplus h_i^{-1}(y \oplus b_i)$ has only the two wanted bytes active (*see #7 of Fig. 7*) **then**
        Store $(y, b_i, h_i^{-1}(y), h_i^{-1}(y \oplus b_i))$ in L_i, where the last two terms are truncated to the 2 active bytes.
  for each (y_5, b_5, u_5, w_5) from L_5 and (y_6, b_6, u_6, w_6) from L_6 **do**
    Add $(u_5, w_5, u_6, w_6, y_5, y_6, b_5, b_6)$ in $L_{5,6}$ indexed by the values of the u_5, w_5, u_6, w_6 operations.
  for each (y_7, b_7, u_7, w_7) from L_7 and (y_8, b_8, u_8, w_8) from L_8 **do**
    Add $(u_7, w_7, u_8, w_8, y_7, y_8, b_7, b_8)$ in $L_{7,8}$ indexed by the values of the u_7, w_7, u_8, w_8 operations.
  Empty L_5, L_6, L_7 and L_8.
for each a in L_A **do**
  for i from 1 to 4 **do**
    for each $x_i \in \{0,1\}^{32}$ **do**
      if $g_i(x_i) \oplus g_i(x_i \oplus a_i)$ has only the two wanted bytes active (*see #4 of Fig. 7*) **then**
        Store $(x_i, g_i(x_i), g_i(x_i \oplus a_i))$ in L_i, where the two last terms are truncated to the 2 active bytes.
  for i from 0 to 1 **do**
    for each $(x_{2i+1}, u_{2i+1}, w_{2i+1})$ in L_{2i+1} and $(x_{2i+2}, u_{2i+2}, w_{2i+2})$ in L_{2i+2} **do**
      if there exists an element in $L_{5+2i,6+2i}$ indexed by $(u_{2i+1}, w_{2i+1}, u_{2i+2}, w_{2i+2})$ **then**
        Add $(x_{2i+1}, x_{2i+2}, b_{5+2i}, b_{6+2i})$ to L^i_{aux} indexed by (b_{5+2i}, b_{6+2i}).
  for each (x_1, x_2, b_5, b_6) in L^0_{aux} **do**
    for each (b_7, b_8) such that $(b_5, b_6, b_7, b_8) \in L_B$ **do**
      if there exists an element in L^1_{aux} indexed by (b_7, b_8) **then**
        add $(a, (b_5, b_6, b_7, b_8), (x_1, x_2, x_3, x_4))$ to $\mathcal{L}_{sol}$.
Return $\mathcal{L}_{sol}$.

the pairs of elements that match on 4 active bytes, and this happens with a probability of 2^{-64} (32 values and 32 differences); and the number of possible pairs is $2^{16+16+l_B+32}$. The memory complexity is $2^{l_B+32+1} + 2^{32+1} + 2^{l_A+l_B}$ for obtaining $2^{l_A+l_B}$ solutions. We explain in Section 4.5 how this algorithm allows to considerably reduce the complexity of the LANE-256 semi-free-start collision presented in [12] when applied jointly with other improvements concerning other steps of the attack.

3.2 Algorithm for ECHO-256

An ECHO-256 state is a state of size 2048 bits that can be seen as a 4×4 matrix of AES states. The ECHO operations BigSR, BigMC and BigSB are similar to the AES ones, but they operate on AES states instead of bytes. A SuperSbox is an Sbox defined by SR ∘ SB ∘ MC ∘ SR ∘ SB. Applied on an AES state, it can be seen as a 32×32 Sbox. We define a *SuperSbox set* as each one of the 4 (in the AES state) sets of bits that act as input and output of the SuperSbox. We define a *BigSuperSbox* as an Sbox defined by

BigSR ∘ BigSB ∘ BigMC ∘ BigSR ∘ BigSB. Applied to ECHO it defines 4 sets of size 4 AES-states.

We consider Fig. 9, where each column represents the four AES states that form a BigSuperSbox at a certain state $\#i$, for i from 1 to 13. Each possible differences in #1 in L_A consist of $z_A = 12$ 32-bit words and the possible differences in #13 consist of $z_B = 8$ 32-bit words, where L_B can be written as $L_B = L_{B_1} \times L_{B_2}$ with both L_{B_1} (associated to AES state B_1 in Fig. 9) and L_{B_2} (associated to AES state B_4) are subsets of $(\{0,1\}^{32})^4$ each of size 2^{32} (this is a particular case which has to be adapted in other cases). Finding solutions for this differential path with the previously mentioned conditions is a problem proposed in [17] and was solved in such a way that 2^{32} solutions could be found with a complexity of 2^{128} in time and 2^{37} in memory. We propose here a new algorithm that can solve it for obtaining 2^{64} solutions with the same time complexity and a memory of 2^{67}. Variants of our algorithm can be applied in several cases, like when the transition in #7 to #8 is from 2 active states to 3, or from 1 to 4 or from 4 to 1. Additionally we believe that it can improve the complexity of other future attacks on ECHO-256.

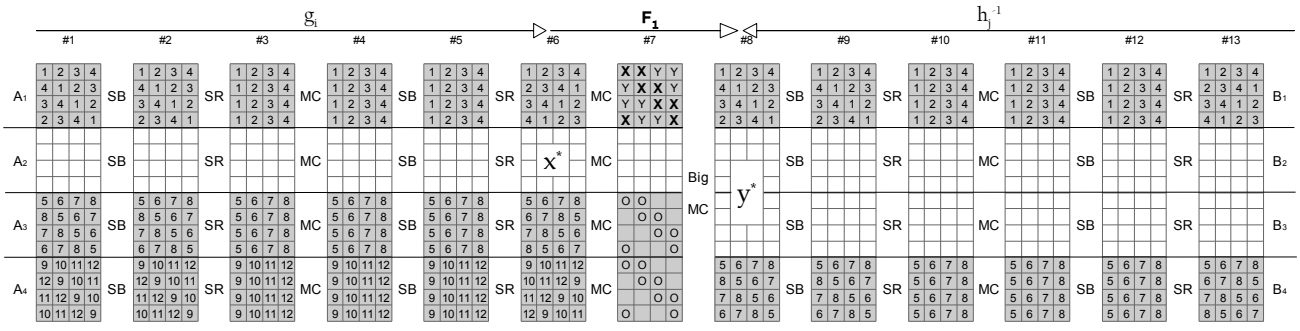


Fig. 9: Differential path on a BigSuperSbox of ECHO-256.

The list L_C contains all the possible values for the input state. This list needs to be neither computed nor stored. Here the aim is to find for each possible (a, b^1, b^2) in $L_A \times L_{B_1} \times L_{B_2}$ the associated c so that $f(c) \oplus f(c \oplus (a, 0^{s(k-z_A)}) = (b^1, b^2, 0^{s(k-z_B)})$. In Fig. 9 we can see how the function f can be written in the way requested by *Problem 2*. We omit the operation BigSR as it does not affect the states, as well as the round keys that are taken into account in the different g_i and h_j. For the sake of simplicity we consider in Fig. 9 the list L_B of possible differences before the last MC of the BigSuperSbox. This can be done by a simple transformation MC^{-1} of the differences in #B' (see Fig. 12). The grey bytes represent the bytes with differences. We can observe that, from #1 to #6 there are $z_A = 12$ independent active SuperSbox sets ($s = 32$), denoted in Fig. 9 by a number from 1 to 12. To each of these groups we can associate a difference from L_A and a value

from L_C at state #1 and we can apply independently g_i, $i \in [1, \ldots, 12]$ to obtain the value and the difference of the group in #6. The same way, from #8 to #13 there are $z_B = 8$ independent active SuperSbox sets and the corresponding functions $h_i^{-1}, i \in [1, \ldots 8]$ that link state #13 with state #8. The function F_1 =MC$\circ$BigMC takes a complete internal state in #6 and computes the corresponding state in #8. Let $f(x) = y$, and let $d_i^{\#7}$ the ith active diagonal in state #7. Without knowing the values of x^* nor of y^* represented in Fig. 9 we can still write the following equations that have to be verified, that are obtained from BigMC, and that are used in the algorithm:

$$2 \times d_i^{\#7} \oplus d_{i+4}^{\#7} \oplus d_{i+8}^{\#7} \oplus 9 \times d_i^{\#7} \oplus 3 \times d_{i+4}^{\#7} \oplus 6 \times d_{i+8}^{\#7} = h_i^{-1}(y_i) \oplus 3 \times h_{i+4}^{-1}(y_{i+4}) \quad (1)$$

for $i \in 1, \ldots, 4$ where the multiplication corresponds to the one in the definition of MC applied independently to each byte of the diagonal.

We consider that the input state (respectively the output state) of the function presented in Fig. 9 is decomposed into sixteen 32-bit words (*i.e.* $s = 32$ and $k = 16$). The input differences (respectively output differences) that we consider in L_A (L_B) correspond to the first $z_A = 12$ ($z_B = 8$) 32-bit words of the state. In Fig. 9 each one of the 12 (respectively 8) 32-bits active word from the input (respectively the output) corresponds to the four active bytes with the same number written on them (1 to 12 for the twelve active input words and 1 to 8 for the eight active output words).

Let V_X (V_Y, V_O respectively) be the values at the positions in #7 marked with an X (Y, O) and Δ_X (Δ_Y, Δ_O) their differences. Let $\Delta_{j'}^{\#r}$ be an auxiliary variable denoting the difference for the SuperSbox set j' in state #r. The algorithm is described in Fig. 10.

So the time complexity is $\mathcal{O}(z_B 2^{l_{B_1}+s} + z_B 2^{l_{B_2}+s} + z_A 2^s + 2^{l_A+64}(2^{l_{B_1}} + 2^{l_{B_2}} + 2^{l_{B_1}} 2^{l_{B_2}} + z_A 2^{64}))$. The memory complexity is $\mathcal{O}(z_B 2^{l_{B_1}+s} + z_B 2^{l_{B_2}+s} + 2^{l_{B_1}+l_{B_2}} + |\mathcal{L}_{sol}|)$. In the case of $l_A = 0$, we will obtain a complexity of 2^{129} in time and 2^{66} in memory for obtaining 2^{64} solutions. This algorithm proposes several trade-offs when changing the values of $|\Delta_X|$, and can be adapted for other forms of L_B.

4 How to improve the best known attacks on five SHA-3 candidates

In this section we first enumerate briefly the main algorithms or ideas that we use to improve the best known attacks on each of the hash functions JH, Grøstl, ECHO, Luffa and Lane as shown on Table 1. Then, we provide more detailed descriptions.

- **JH:** To improve the complexities over the ones in [16] we use the instant-matching (as in Section 2.1) and gradual-matching algorithms as well as the fact that we do not merge the lists until we really have to (to keep lists of smaller sizes, with a smaller complexity).
- **Grøstl:** Instead of the initial lists used in [15], we can define them so that we erase the elements that for sure won't verify the outbound part. Having lists of smaller size translates to a smaller complexity.

Fig. 10 Algorithm for finding solutions for one ECHO BigSuperSbox.

Require: Function f, list L_A of differences in #1 and lists L_{B_1} and L_{B_2} of differences in #13.
Ensure: List $\mathcal{L}_{sol} = \{(a, b^1, b^2, c),$ such that $f(c) \oplus f(c \oplus (a, 0^{s(k-z_A)})) = (b^1, b^2, 0^{s(k-z_B)})\}$.

for j from 1 to 4 **do**
  for each $y_j \in \{0,1\}^{32}$ and **for** each b^1 from L_{B_1} **do**
    Store $(h_j^{-1}(y_j), h_j^{-1}(y_j) \oplus h_j^{-1}(y_j \oplus b_j^1))$ in $L^j_{\#8,b^1}$ (one of 4×2^{32} *lists of size* 2^{32}).
for j from 5 to 8 **do**
  for each $y_j \in \{0,1\}^{32}$ and **for** each b^2 from L_{B_2} **do**
    Store $(h_j^{-1}(y_j), h_j^{-1}(y_j) \oplus h_j^{-1}(y_j \oplus b_j^2))$ in $L^j_{\#8,b^2}$ (one of 4×2^{32} *lists of size* 2^{32}).
for each a in L_A **do**
  for i from 1 to 12 **do**
    for each $x_i \in \{0,1\}^{32}$ **do**
      Store $(g_i(x_i), g_i(x_i) \oplus g_i(x_i, a_i))$ in $L^i_{\#6}$.
  for Δ_X from 0 to $2^{64} - 1$ (*and not the 128 bits as done in [17]*) **do**
    Compute Δ_O (*with linear conditions of inactive states in #8*) and $\Delta^{\#8}_{j'}$ for $j' \in \{1,2,5,6\}$ (*with BigMC*).
    for each b^1 in L_{B_1} and for $j = [1,2]$ **do**
      Find an element in $L^j_{\#8,b^1}$ such that $h_j^{-1}(y_j) \oplus h_j^{-1}(y_j \oplus b_j^1) = \Delta_j^{\#8}$ and store $(h_1^{-1}(y_1), \Delta_1^{\#8}, h_2^{-1}(y_2), \Delta_2^{\#8}, b^1)$ in L_{aux_1}.
    for each b^2 in L_{B_2} and for $j = [5,6]$ **do**
      Find an element in $L^j_{\#8,b^2}$ such that $h_j^{-1}(y_j) \oplus h_j^{-1}(y_j \oplus b_j^2) = \Delta_j^{\#8}$ and store $(h_5^{-1}(y_5), \Delta_5^{\#8}, h_6^{-1}(y_6), \Delta_6^{\#8}, b^2)$ in L_{aux_2}.
    for each $(h_1^{-1}(y_1), \Delta_1^{\#8}, h_2^{-1}(y_2), \Delta_2^{\#8}, b^1)$ in L_{aux_1} and **for** each $(h_5^{-1}(y_5), \Delta_5^{\#8}, h_6^{-1}(y_6), \Delta_6^{\#8}, b^2)$ in L_{aux_2} **do**
      Compute $V_1' = h_1^{-1}(y_1) \oplus 3 \times h_5^{-1}(y_5)$ and $V_2' = h_2^{-1}(y_2) \oplus 3 \times h_6^{-1}(y_6)$, and store $((h_1^{-1}(y_1), \Delta_1^{\#8}, h_2^{-1}(y_2), \Delta_2^{\#8}, b^1), (h_5^{-1}(y_5), \Delta_5^{\#8}, h_6^{-1}(y_6), \Delta_6^{\#8}, b^2))$ in a hash table T indexed by these (V_1', V_2').
    for Δ_Y from 0 to $2^{64} - 1$ **do**
      Determine by BigMC $\Delta^{\#8}_{j'}$ for $j' = 3,4,7,8$; and $\Delta_j^{\#6}$ for $j \in [1, \ldots, 12]$.
      for i from 1 to 12 **do**
        Find the element from $L^i_{\#6}$ such that $g_i(x_i) \oplus g_i(x_i, a_i) = \Delta_i^{\#6}$.
      Compute with them by MC the values $d_i^{\#7}$ of the active diagonals in #7 and then $V_j = 2 \times d_j^{\#7} \oplus d_{j+4}^{\#7} \oplus d_{j+8}^{\#7} \oplus 9 \times d_j^{\#7} \oplus 3 \times d_{j+4}^{\#7} \oplus 6 \times d_{j+8}^{\#7}$ for $j = 1, 2$.
      if there is an element such that $V_1' = V_1$ and $V_2' = V_2$ in T (*one on average, determines* b^1 *and* b^2) **then**
        Find from $L^{j'}_{\#8,b^1}$ the element $(h_{j'}^{-1}(y_{j'}), \Delta^{\#8}_{j'})$ for $j' = 3,4$. This determines y_3 and y_4.
        Find from $L^{j'}_{\#8,b^2}$ the element $(h_{j'}^{-1}(y_{j'}), \Delta^{\#8}_{j'})$ for $j' = 7,8$. This determines y_7 and y_8.
        if with these values of $(h_{j'}^{-1}(y_{j'})$, $j' = 3,4,7,8$ and the ones obtained in step 22 of $g_i(x_i)$ for $i = 3,4,7,8,11,12$ that we have not used yet, the equation (1) for $i = 3,4$ derived from F_1 can be verified (*happens with a probability of* 2^{-64}) **then**
          The value x_* is determined. Add the element $(x_1, \ldots, x_{z_A}, x^*, a, b^1, b^2)$ to $\mathcal{L}_{sol}$
Return $\mathcal{L}_{sol}$, containing about 2^{64+l_A} elements.

– **ECHO:** Using conviniently the algorithm from Section 3.2 we provide better trade-offs improving the time complexity from [17].
– **Luffa:** The parallel-matching algorithm is applied in [10], improving the time complexity over the brute force merging method by increasing the memory requirements. If we apply instead the gradual-matching algorithm (with three layers), the time complexity can still be better than the brute force one while the memory needs are not increased.
– LANE**:** In the cases of LANE-256 and LANE-512 several improvements are applied at different steps of the attacks from [12]. They use the instant-matching algorithm, as well as some more appropriate ways to formulate the problem, as shown in Appendix A, and the algorithms from Section 3.1 and from Appendix B.

For a detailed description of the hash functions, we refer to their SHA-3 submission documents. As those attacks are quite complex, we do not explain here all the details, but we give the information needed for identifying the problem, referring in each case to the corresponding attack. These improvements are based on the algorithms that we have described in this paper as well as on recognizing the situations where they can be applied. This way we are able to reduce the overall complexity of the attacks.

4.1 JH

For simplicity, we consider here the attack on JH with $d = 4$ for 8 rounds when using the three-inbound attack given in [16] with a complexity of $2^{32.09}$ in time and $2^{24.18}$ in memory. We shall see here how, when we apply one of the previously introduced algorithms, this complexity can be significantly improved. For $d = 8$ the improvement is performed the same way for the three-inbound attack on 19 and 22 rounds, and it is simpler in the case of one-inbound for 16 rounds. The three-inbound attack for $d = 4$ uses the differential path represented in Fig. 11, where #0 represents the initial internal state and #8 the final one. The colored parts are the parts with a difference. Each small square represents the 4×4 Sbox for rows from 0 to 15, and each rectangle represents the linear permutation on 8 bits. Each wire conrresponds to 4 bits.

To improve this attack, we use the algorithms from Sections 2.1 and 2.2. Besides, we consider the same three inbounds as in [16] but we sometimes keep two non-overlaping groups of solutions per inbound (instead of merging them into one). Without this, we could still improve the time complexity, but this would be limited by the size of the intermediate lists stored, $2^{24.18}$. Keeping two lists instead of one means that their size will be smaller. We start the attack as in [16] by finding the possible solutions for the first inbound (from round #0 to the beginning of round #2), storing a list L_A of $2^{11.36}$ solutions with a cost of 2^{16}.

We consider the third inbound, from round #5 to the beginning of round #7. In this part, we obtain two sets of 2^{16}, each one associated to a list: $L^5_{0,1,8,9}$ and $L^5_{2,3,10,11}$. This is done by first building the lists, $L_{0,1}$, $L_{8,9}$, $L_{2,3}$ and $L_{10,11}$ of size $2^{11.91}$ each verifying the

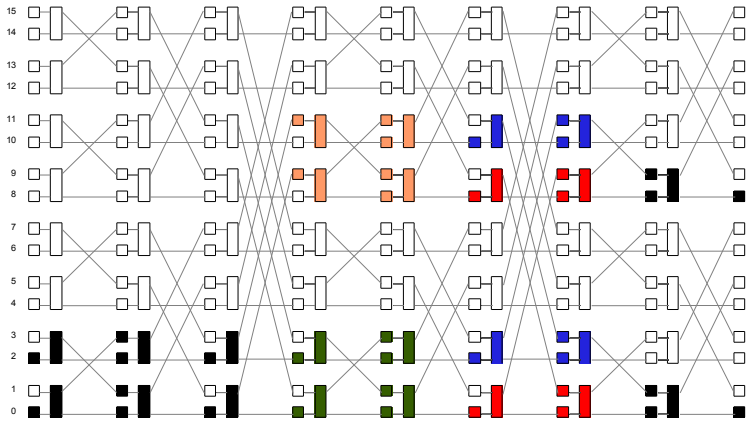


Fig. 11: Differential path for $d = 4$ of JH of the three-inbound attack.

conditions from #6. Next the two first ones are merged in the same way as the Example 1 of Section 2.1 using the instant-matching algorithm. The list $L^5_{0,1,8,9}$ is obtained. We do the same with lists $L_{2,3}$ and $L_{10,11}$ to obtain $L^5_{2,3,10,11}$, both of size 2^{16}. The cost of this phase is 2^{17}.

Next, for rounds #3 and #4 we will repeat the same procedure at the same cost for obtaining two sets of solutions for these two rounds: $L^3_{0,1,2,3}$ and $L^3_{8,9,10,11}$ of size also 2^{16}. We will now merge these two lists and the list L_A. Merging $L^3_{0,1,2,3}$ and $L^3_{8,9,10,11}$ determines 3.91×2 bit conditions and, for merging both of them with L_A, 16 bit conditions need to be verified (from the two active 4-bit words where they collide). Merging these three lists can be done by first applying a gradual-matching to $L^3_{0,1,2,3}$ and L_A using the groups of differences. Next, we can apply an instant-matching with the partial solutions and list $L^3_{8,9,10,11}$. As a result, a new list L_B of size $2^{19.54}$ is obtained with a cost of $2^{19.54}$. The elements of this list are solutions for the rounds #0 to #5. Next, in a similar way we will merge L_B with $L^5_{0,1,8,9}$ and $L^5_{2,3,10,11}$ (here there are 32 bit conditions to verify), and we will obtain $2^{19.54}$ solutions that verify the merge (and so rounds from #0 to #7) with a cost of $2^{19.54}$. For each solution, we check if it also verifies round #8 (3.91×2 bit conditions), obtaining $2^{11.72}$ solutions (as in [16], before taking the symmetries into account). The complexity of the attack using our algorithm is then $2^{19.54}$ in time and $2^{19.54}$ in memory, improving the previous complexity of $2^{32.09}$ in time and $2^{24.18}$ in memory. Similarly as we have shown for $d = 4$ and 8 rounds, we can identify the same problem and apply the algorithms of Sections 2.1 and 2.2 to the attack on 19 and 22 rounds of [16] that uses three-inbound attacks and has a complexity of $2^{168.02}$ in time and $2^{143.70}$ in memory, so

that it can also be improved using the same algorithm, and having a final complexity of $2^{95.63}$ in time and memory. The 16-round attack with one-inbound attack of [16], can also be improved to $2^{96.12}$ in time and memory, while its complexity was 2^{190} in time and 2^{104} in memory.

4.2 *Grøstl*

In this case we do not apply one of the algorithms but we state again the importance of identifying the best problem to solve. Here, we consider the results on Grøstl-256 presented in [15], where, in particular, distinguishers are given for the full compression function as well as for the internal permutation. We can improve by a factor of 2^{10} or 2^{17} (depending on the differential path considered) their time complexities. In this case, instead of finding a new algorithm (the corresponding part of the path can be directly solved with a SuperSbox precomputation) we have identified a better problem to solve: the lists L_A and L_B, representing differences in the input and in the output of the SuperSbox phase respectively, can have a smaller size than considered in [15]. They were built with all the possible differences, but we noticed that they can be smaller by just storing the differences that we know for sure might also satisfy the outbound phase. The factor that we are going to gain will depend on the number of active columns in the input (N_i) and the number of active columns in the output (N_o). So instead of merging two lists of size $2^{l_A} = 2^{64N_i}$ and $2^{l_B} = 2^{64N_o}$, we have to merge one list of size $2^{l_A} = 2^{63N_i}$ and one list of size $2^{l_B} = 2^{56N_o}$. The algorithm applied to merge these lists is the same one as in [15], obtaining a complexity in time of $2^{63N_i+56N_o}$ instead of $2^{64(N_i+N_o)}$. This is possible because in this attack the one byte differences introduced by the constants additions have a fixed value, implying that the number of possible differences at the input and output of the SuperSbox will be smaller. In the 10-round compression function analysis this improves from 2^{192} to 2^{182} and in the permutation distinguisher from 2^{192} to 2^{175}. In the case of Grøstl-512 we can improve time complexity of the analysis on 11 rounds of the compression function from 2^{640} to 2^{630}.

4.3 ECHO-256

In [17] an analysis of the whole ECHO-256 permutation is provided which has complexity 2^{182} in time and 2^{37} in memory. By studying in detail this analysis we have been able to provide some trade-offs that were previously unknown and that allow to improve the time complexity. For example, we can perform the same attack with a complexity 2^{151} in time and 2^{67} in memory. We consider the differential path given in [17]. In Fig. 12, the inbound part is represented. We need to find 2^{86} solutions of this part in order to satisfy also the outbound part. In Fig. 12, the BigSB are decomposed into the AES operations (2 rounds, where we omit operations that do not influence the differential path) and we can see how two BigSB can be seen as a BigSuperSbox (from #A to #B'), where the sets formed by it have the form of the highlighted sets of four AES states. For finding solutions for each

one of this 4 BigSuperSbox we can apply the algorithm from Section 3.2. As with one element from L_A we obtain 2^{64} solutions, we will have to iterate the algorithm over 2^{22} different a. For reducing the memory needs, we will find solutions for the whole inbound considering $l_A = 0$ and next we will repeat the process for 2^{22} different a. The elements in L_{B_1} (L_{B_2}) are generated by the 2^{32} possible differences in the AES state $(0, 0)$ $(1, 1$ respectively) in $\#\beta$. We then apply the algorithm from Section 3.2 to each BigSuperSbox

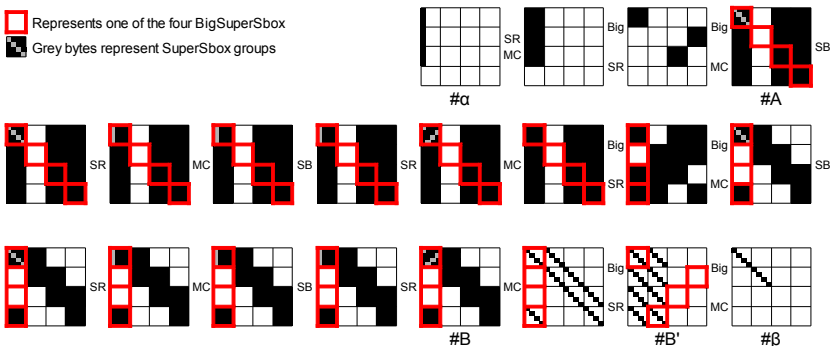


Fig. 12: Inbound part of the differential path on ECHO. A number of 2^{86} solutions needs to be found for satisfying the outbound part.

set, obtaining 4 associated sets of solutions of size $2^{64+l_A} = 2^{64}$. Each element from one set will be associated to an unique 3-tuple of elements from the other groups: the ones that were generated by the same difference in $\#\beta$ (that define the b_1, b_2 differences). This gives in total 2^{64} solutions for the whole inbound phase. As said previously, if we repeat this procedure for 2^{22} distinct differences in $\#\alpha$ (that define the a differences) we will obtain the 2^{86} needed solutions with a time complexity of 2^{151} and memory of 2^{67}.

4.4 *Luffa*

In [10], a way of finding a semi-free-start collision is provided for 7 rounds out of 8. This is done by using the differential path represented in Fig. 13, where each small square represents one bit, and the colored ones are the ones with differences. This path is solved by first, finding solutions for the possible differences of the path from #1 to #7 (in L_A). In parallel the possible differences are found for the part of the path from #8 to #14 (in L_B). State #7 is separated from #8 by 64 4×4 Sboxes. Among them, 52 are active. We want to keep the possible differences for the whole path from #1 to #14. In this case, the problem is very easy to identify: we have two lists of differences, one of differences of the inputs of 52 active Sboxes, the other one of the outputs of these 52 active Sboxes. We can

apply the gradual-matching or the parallel-matching algorithms, as we did in Example 2. In [10] the parallel-matching was applied, reducing the time complexity from $2^{132.6}$ to 2^{104}, while the memory complexity increases to 2^{102}. We can also apply the gradual-matching algorithm with $z' = 16$ and obtain an improved time complexity of $2^{112.9}$ while the memory complexity stays the same ($2^{68.8}$).

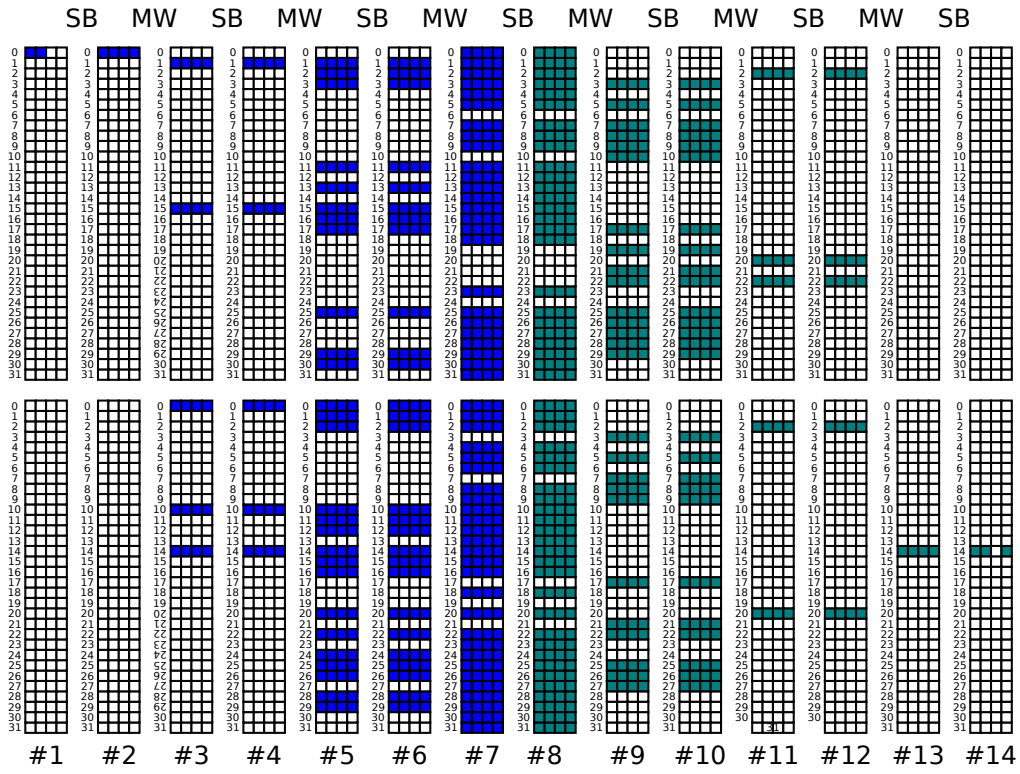


Fig. 13: Differential path used in the rebound attack from [10] on Luffa.

4.5 Lane-256

The analysis in [12] provides a way of finding a semi-free-start collision for the complete compression function of Lane-256 with a complexity of 2^{96} in time and 2^{88} in memory. In this section we are going to identify 2 concrete problems extracted from this attack, and by applying two of the previously described algorithms, we are able to reduce the total complexity of the attack to 2^{80} in time and 2^{66} in memory, or more precisely, to 2^{80} in

time and 2^{58} in memory + 2^{64} in time and 2^{66} in memory. We are not going to describe in detail here the analysis from [12], but we give the information needed for identifying and defining the problem to be treated by the corresponding algorithm.

First problem: In this attack, the first three steps aim at finding 2^{56} solutions for two inbounds in 4 independent lanes. Each one of the four lanes represents an independent and similar problem. Instead of looking at it as three steps, we are going to unify it in just one, and we will use the differential path from Fig. 7. We can now build the list of possible differences in the input: from five active bytes, we can obtain 2^{40} possible differences in the input, before the first SB considered. We store 2^{32} out of these 2^{40} and this forms the list L_A. We can do the same with the possible differences in the output: out of a totally full active AES state, we want to reach a position with only 4 active bytes. The list L_B will be formed by all the 2^{32} possible differences in the output after the last Sbox considered (in the two inbounds). We want to merge these two lists keeping the differences that can verify the whole path defined by the two inbounds and to recover the associated values. So we want to obtain a total of 2^{64} values and differences as solutions. There is an extra condition of one byte before the differential path from Fig. 7, so we finally obtain 2^{56} solutions. We will directly apply the algorithm from Section 3.1. The cost of this step was 2^{96} in time and 2^{88} in memory (it was the bottleneck for time and memory). Now, we can perform these two inbound phases with a complexity of 2^{66} in time complexity and 2^{65} in memory. As this step is not bottleneck anymore for the attack, we can now try to reduce the rest of the complexities.

Second problem: Once the previous step is finished we have obtained 2^{56} solutions for the first two inbounds, for each lane (four lists of values and differences). They need to be merged so that they verify the message expansion. In [12] this is done in steps 4 and 5 with a complexity of 2^{80} in time and memory. This memory complexity can be reduced to 2^{48} by directly applying the example from Section A.1, obtaining 2^{64} solutions for this step and giving the new bottleneck of the time complexity of the attack: 2^{80}. The last part of the attack is the same as in [12], and corresponds to the bottleneck in memory: $2^{64} \times 4$.

4.6 Lane-512

A semi-free-start collision attack is given in [12] for the whole compression function of Lane-512 with a complexity of 2^{224} in time and 2^{128} in memory. Applying three of the previously described algorithms we can reduce this memory complexity from 2^{128} to 2^{66}. At this aim, we have to identify 3 problems.

First problem: The original first step in the attack on Lane-512 leads to 4 lists of 2^{68} solutions for a first inbound. We realized that, as it is possible to change the number of active bytes at the beginning of each lane from 6 to 4, obtaining 2^{56} solutions is enough. Steps 2 and 3 merge these 4 lists for finding one solution that verifies also the message

expansion. We can apply, as we did before, the example from Section A.1. We obtain one solution with complexity 2^{56} in memory and 2^{80} in time, instead of the previous 2^{88} in time and memory.

Second problem: In the attack on LANE-512, in the Starting Points phase, four lists of values are built, of size 2^{64}. The Merge Lanes and Message Expansion phases need a complexity of 2^{128} in time and 2^{128} in memory. We can instead apply the example from Section A.2. With a complexity of 2^{192} in time and $2^{64} \times 4$ in memory we can obtain the 2^{128} starting points needed for repeating the rest of the attack enough times until we find one solution for the whole path (and so a semi-free-start collision). We do not need to store these 2^{128} starting points, because we can perform the rest of the attack as soon as we find one. This way, the memory complexity does not go beyond 2^{64}, and the time complexity, though higher, is not the bottleneck.

Third problem: In [12], the second merge of inbound phases (that finds a collision between two lanes) needs a memory of $2^{96} \times 4$. With the previous improvements, the memory needed is 2^{64}, so we want to reduce the memory needs of this last phase to 2^{64}. We have three lists of 2^{32} elements for each of the two lanes of the same branch (6 in total). Instead of merging the three lists into a new one of size 2^{96}, as done in [12], we can apply the algorithm from Appendix B. This way we will only store a list of 2^{64} elements.

5 Conclusion

The main contributions of this paper can be classified in three groups. First, we propose several algorithms for solving the problem which constitutes the bottleneck of most rebound attacks, leading to improvements of the previously known complexities.

Secondly, we highlight with some examples the importance of identifying the situations that could help improving the complexity of this type of attacks and we show how to find the problems in each particular case that will provide the best overall complexity. This is often a difficult task due to the high technicality of the attacks and algorithms.

Finally, the previous two contributions lead to improvements of most of the best known rebound attacks applied to the SHA-3 candidates JH, Grøstl, Luffa, ECHO-256 and LANE. It is important to point out that we just tried to improve the complexities of existing attacks. However, the work presented in this paper can be very useful for future rebound attacks, in particular we believe that the attacks on JH and on the compression function of ECHO can be improved (extending the number of rounds attacked) by exploiting the algorithms and ideas presented here. Finally, we believe that some of these algorithms, specially those of Section 2, will be applicable in other contexts besides rebound attacks.

Acknowledgements

The author would like to thank Willi Meier, Simon Knellwolf, Marine Minier, Thomas Peyrin, Martin Schläffer, Joana Treger and Fabien Viger for many helpful comments and discussions. A special mention is needed for Anne Canteaut and Andrea Röck for all the help and suggestions to improve this paper.

References

1. Barreto, P.S.L.M., Rijmen, V.: The Whirlpool Hashing Function, revised in 2003.
2. Benadjila, R., Billet, O., Gilbert, H., Macario-Rat, G., Peyrin, T., Robshaw, M., Seurin, Y.: Sha-3 proposal: ECHO. Submission to NIST (updated) (2009)
3. Camion, P., Patarin, J.: The knapsack hash function proposed at Crypto'89 can be broken. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 547, pp. 39–53. Springer (1991)
4. Canniere, C.D., Sato, H., Watanabe, D.: Hash Function Luffa: Specification. Submission to NIST (Round 2) (2009)
5. Canteaut, A., Naya-Plasencia, M.: Structural weaknesses of permutations with low differential uniformity and generalized crooked functions. In: Finite Fields: Theory and Applications - Selected papers from the 9th International Conference Finite Fields ans applications. Contemporary Mathematics, vol. 518, pp. 55–71. AMS (2010), http://www-rocq.inria.fr/secret/Maria.Naya_Plasencia/papers/canteaut-nayaplasencia.pdf
6. Chose, P., Joux, A., Mitton, M.: Fast correlation attacks: An algorithmic point of view. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 2332, pp. 209–221. Springer (2002)
7. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl – a SHA-3 candidate. Submitted to the SHA-3 competition, NIST (2008), http://www.groestl.info
8. Gilbert, H., Peyrin, T.: Super-Sbox Cryptanalysis: Improved Attacks for AES-like permutations. In: FSE. Lecture Notes in Computer Science (2010), to appear
9. Indesteege, S.: The Lane hash function. Submitted to the SHA-3 competition, NIST (2008), http://www.cosic.esat.kuleuven.be/publications/article-1181.pdf
10. Khovratovich, D., Naya-Plasencia, M., Røck, A., Schläffer, M.: Cryptanalysis of Luffa v2 components. In: SAC. Lecture Notes in Computer Science, vol. 6544, pp. 388–409 (2010)
11. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 5912, pp. 126–143. Springer (2009)
12. Matusiewicz, K., Naya-Plasencia, M., Nikolic, I., Sasaki, Y., Schläffer, M.: Rebound Attack on the Full Lane Compression Function. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 5912, pp. 106–125. Springer (2009)
13. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In: Fast Software Encryption - FSE 2009. Lecture Notes in Computer Science, vol. 1008. Springer (5665)
14. Mendel, F., Peyrin, T., Rechberger, C., Schläffer, M.: Improved cryptanalysis of the reduced Grøstl compression function, ECHO permutation and AES block cipher. In: Jacobson, Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 5867, pp. 16–35. Springer (2009)
15. Peyrin, T.: Improved Differential Attacks for ECHO and Grøstl. In: Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6223, pp. 370–392. Springer (2010)

16. Rijmen, V., Toz, D., Varici, K.: Rebound Attack on Reduced-Round Versions of JH. In: FSE. Lecture Notes in Computer Science, vol. 6147, pp. 286–303 (2010)
17. Sasaki, Y., Li, Y., Wang, L., Sakiyama, K., Ohta, K.: Non-Full-Active Super-Sbox Analysis Applications to ECHO and Grøstl. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 6477, pp. 38–55 (2010), to appear
18. Schroeppel, R., Shamir, A.: A T=O($2^{n/2}$), S=O($2^{n/4}$) algorithm for certain NP-complete problems. SIAM J. Comput. 10(3), 456–464 (1981)
19. Wagner, D.: A generalized birthday problem. In: CRYPTO. Lecture Notes in Computer Science, vol. 2442, pp. 288–303. Springer (2002)
20. Wu, H.: The hash function JH. Submission to NIST (updated) (2009), http://icsd.i2r.a-star.edu.sg/staff/hongjun/jh/jh_round2.pdf
21. Wu, S., Feng, D., Wu, W.: Cryptanalysis of the LANE hash fonction. In: SAC 2009 - Selected Areas in Cryptography. Lecture Notes in Computer Science, Springer (2009)

A Importance of Identifying the Appropriate Problems

Here we present some improvements of two different steps of the attacks against LANE presented in [12], which result from a formulation of the underlying problem which is more appropriate for applying the instant-matching algorithm.

A.1 Example 3: Using Equalities for Dividing the Problem

We improve the memory complexity needed for steps 4 and 5 of the attack in [12], which was the bottleneck of the attack. At the beginning of step 4, four lists have been obtained (L_1, L_2, L_3 and L_4), each one with 2^{56} elements. These elements can be represented in 20 groups of size $s = 8$ bits. Among these 20 bytes, 4 correspond to differences and 16 to values. Let $(v_j^i)_{1 \leq j \leq 20}$ be the 20 bytes of an element v in list L_i. We denote by ℓ_1 and ℓ_2 two linear permutations. We want to find all 4-tuples (v^1, v^2, v^3, v^4) from the four lists that satisfy the following relations:

$$v_i^1 = v_i^2 = v_i^3 = v_i^4 \text{ for } 1 \leq i \leq 4$$

$$\ell_1(v_i^1, v_i^2) = \ell_2(v_i^3, v_i^4) \text{ for } 5 \leq i \leq 20.$$

In [12], this problem was solved with a complexity of 2^{80} in time and 2^{80} in memory. Their approach was to first *merge* lists L_1 and L_2, as well as L_3 and L_4 by using the equations involving the differences. This lead to two new lists $L_{1,2}$ and $L_{3,4}$ of size $2^{56+56-8 \cdot 4} = 2^{80}$ each. Next, these two lists were *merged* using the remaining equations, obtaining one solution on average since $2^{l_{1,2}+l_{3,4}-p_t} = 2^{80+80-20 \cdot 8} = 1$. However this memory complexity can be improved. First we can separate this problem in smaller ones: by considering the first equation, we know that we won't find a 4-tuple that will be a solution unless all the elements have the same first four bytes. We can then separate each one of the four lists in $2^{4s} = 2^{32}$ sublists, L_i^γ for γ from 0 to $2^{32} - 1$, so that each sublist L_i^γ contains the elements from L_i that had a difference γ in the 4 bytes. Now the initial problem can be seen as

2^{32} independent problems, where the *merge* is determined by the conditions on the last 16 bytes. Each problem can be solved with the instant-matching algorithm with parameters: $N = 4$, $N' = 2$, $z = 16$, $s = 8$, $p_t = 128$, $l_i = 56 - 32 = 24$ and t_j is the $\oplus$ operation. The memory complexity now is $4 \cdot 2^{56} + 2^{24+24} \simeq 2^{58}$ instead of 2^{80} while the time complexity stays the same.

A.2 Example 4: Increasing Time to Reduce Memory

We present here another application example. In this case, we suppose that the memory complexity is the bottleneck of the attack instead of the time complexity (so we are allowed to increase it). We study a case we find when improving the overall complexity of the attack on LANE-512 presented in [12]. We consider steps 7 and 8 of the attack. In this particular case, the time complexity, of 2^{224}, was imposed by another step of the attack. The concrete problem is the following: we have four lists, $L_1, \ldots, L_4$ of size 2^{64} elements. These elements are defined by $s = 8$, $z = 8$. Let ℓ_1 and ℓ_2 be linear permutations. Let $v_1^i, \ldots, v_8^i$ denote an element of L_i. Then we want to find all the 4-tuples of values (v^1, v^2, v^3, v^4) in $L_1 \times L_2 \times L_3 \times L_4$ that verify the following relation over $(\{0,1\}^8)^2$:

$$\ell_1(v_j^1, v_j^2) = \ell_2(v_j^3, v_j^4),\ j \in [1, z].$$

Here we also have $P_t 2^{zs} = 1$. In the attack presented in [12], this part was solved with a complexity of 2^{128} in time and memory. With the improvements that we present in this paper of other steps of the attack in [12], this step would be the bottleneck in memory, but we show here how to reduce this memory complexity to 2^{66} with a correct use of the instant-matching algorithm. We notice that this relation can also be written the following way ($N' = 1$):

$$v_j^1 = \ell'(v_j^2, v_j^3, v_j^4).$$

We can then directly apply the instant-matching algorithm obtaining $2^{192+64-8s} = 2^{128}$ solutions. In this case, each time we obtain a match we can use it instead of storing it. Hence the memory complexity is now 2^{66} and the time complexity $2^{64 \cdot 3} = 2^{192}$, which is still below the bottleneck in time complexity. This way, memory needs are reduced from 2^{128} to 2^{66} while the overall time complexity stays 2^{224}.

B Algorithm for Improving the Complexity of the Third Problem in LANE-512

We consider the path given by Fig. 14. In this case, we are given 6 lists, L_A, L_B, L_C, $L_{A'}$, $L_{B'}$ and $L_{C'}$ of size 2^{32}, where each list L_i contains possible values for the AES state marked in Fig. 14 with an i on state #1. The black bytes represent bytes with differences and have been completely determined in previous inbound phases, *i.e.* the value and the difference in each black byte is fixed. We want to find all the elements from $L_A \times L_B \times L_C \times L_{A'} \times L_{B'} \times L_{C'}$

such that, when we consider the values $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ in the three corresponding states of #1, and we compute the state #15, the difference Δ obtained is the same as the Δ' obtained when we consider the values $\boldsymbol{a'}, \boldsymbol{b'}, \boldsymbol{c'}$ in the parallel lane. The final operations have been omitted for the sake of simplicity, as they are linear and colliding in #15 is equivalent to colliding at the end. We expect to find $2^{32\times 6-128}$ solutions. In [12] these 6-tuples are found by computing a list L_{ABC} with all values in $L_A \times L_B \times L_C$ and the corresponding Δ, and then by checking for all triples $L_{A'} \times L_{B'} \times L_{C'}$ the resulting Δ' belongs to L_{ABC}. The complexity for finding the 2^{64} solutions is 2^{97} in time and 2^{96} in memory. With the other improvements presented in Section 4.6, this memory requirement would be the bottleneck of the attack. We show here how to apply an algorithm for solving this problem that would need the same time complexity but with a memory complexity of only 2^{64}.

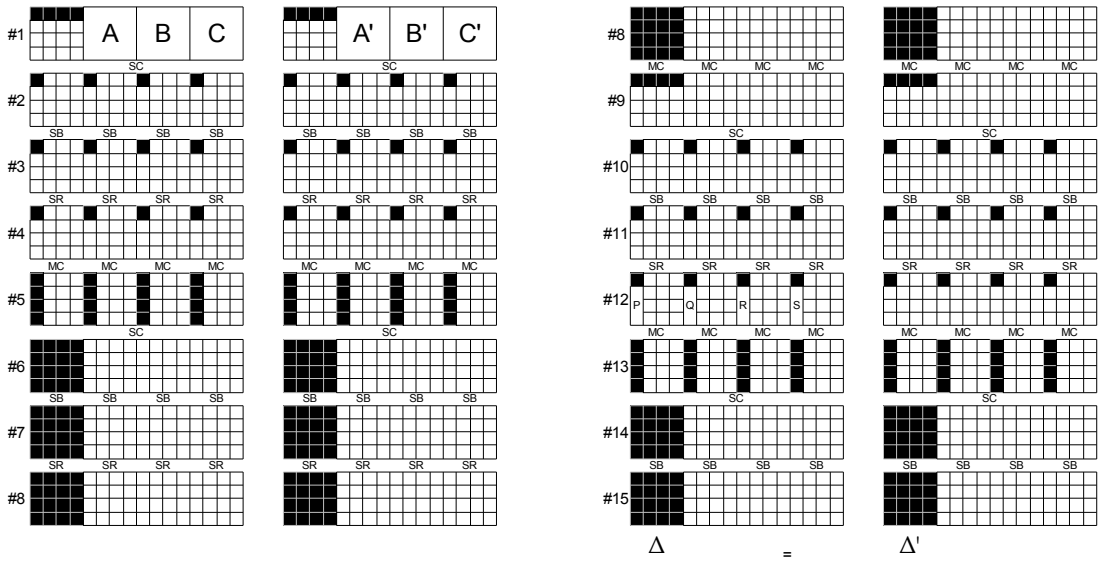


Fig. 14: Part of the differential path on LANE-512 representing the third improved part from Section 4.6

First we remark that if we go through all the 2^{24} possible values for the three bytes in #12 marked with a P, then we will generate all the possible values for the differences in the first column of Δ. That means that this column can only take 2^{24} possible differences among the 2^{32}. The same happens with the groups Q, R and S for the second, third and fourth columns of Δ, respectively. We proceed as follows:

1. We store four tables of size 2^{24}, L_P, L_Q, L_R and L_S, of possible differences in each of the columns of Δ.
2. For each one of the 2^{96} elements in $L_{A'} \times L_{B'} \times L_{C'}$ we compute the associated Δ' and we check if each of its four columns is included in the corresponding list L_P, L_Q, L_R or L_S. For each column, this will be the case with probability $2^{24-32} = 2^{-8}$. Then the probability that Δ' is valid is 2^{-32}.
3. If Δ' is valid we add an element $(\Delta', \boldsymbol{a'}, \boldsymbol{b'}, \boldsymbol{c'})$ to the list $L_{A'B'C'}$. At the end, the size of this list is $2^{96-32} = 2^{64}$.
4. Once $L_{A'B'C'}$ is computed, we can try for all the 3-tuples from $L_A \times L_B \times L_C$ if the Δ they generate belongs to $L_{A'B'C'}$. This will happen with a probability of $2^{-24\times 4} = 2^{-96}$.

The number of solutions is 2^{64}. With our algorithm we can find them with the same time complexity as before and with a reduced memory complexity of 2^{64} instead 2^{96} as was the case in [12].

Rebound Attack on JH42

María Naya-Plasencia[1]*, Deniz Toz[2]†, Kerem Varıcı[2]†

[1] FHNW Windisch, Switzerland and University of Versailles, France
maria.naya-plasencia@prism.uvsq.fr
[2] Katholieke Universiteit Leuven, ESAT/COSIC and IBBT, Belgium
{deniz.toz, kerem.varici}@esat.kuleuven.be

Abstract. The hash function JH [20] is one of the five finalists of the NIST SHA-3 hash competition. It has been recently tweaked for the final by increasing its number of rounds from 35.5 to 42. The previously best known results on JH were semi-free-start near-collisions up to 22 rounds using multi-inbound rebound attacks. In this paper we provide a new differential path on 32 rounds. Using this path, we are able to build various semi-free-start internal-state near-collisions and the maximum number of rounds that we achieved is up to 37 rounds on 986 bits. Moreover, we build distinguishers in the full 42-round internal permutation. These are, to our knowledge, the first results faster than generic attack on the full internal permutation of JH42, the finalist version. These distinguishers also apply to the compression function.

Keywords. hash function, rebound attack, JH, cryptanalysis, SHA-3

1 Introduction

A cryptographic hash function is a one way mathematical function that takes a message of arbitrary length as input and produces an output of fixed length, which is commonly called a fingerprint or message digest. Hash functions are fundamental components of many cryptographic applications such as digital signatures, authentication, key derivation, random number generation, etc. So, in terms of security any hash function should be preimage, second-preimage and collision resistant.

*Supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center of the Swiss National Science Foundation under grant number 5005-67322 and by the French Agence Nationale de la Recherche through the SAPHIR2 project under Contract ANR-08-VERS-014

†This work was sponsored by the Research Fund K.U.Leuven, by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy) and by the European Commission through the ICT Programme under Contract ICT-2007-216676 (ECRYPT II). The information in this paper is provided as is, and no warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

Most of the recent hash functions use either compression functions or internal permutations as building blocks in their design. In addition to the main properties mentioned above, some ideal properties should also be satisfied for the building blocks. This means that the algorithm should not have any structural weaknesses and should not be distinguishable from a random oracle. The absence of these properties on building blocks may not impact the security claims of the hash function immediately but it helps to point out the potential flaws in the design.

Since many of the hash standards [16,19] have been broken in recent years, the National Institute of Standards and Technology (NIST) announced a competition to replace the current standard SHA-2 with a new algorithm SHA-3. The hash function JH [20], designed by Hongjun Wu, is one of the five finalists of this competition. It is a very simple design and efficient in both software and hardware. JH supports four different hash sizes: 224, 256, 384 and 512-bit. It has been tweaked from the second round to the final round by increasing its number of rounds from 35.5 to 42. The new version is called JH42.

Related Work: We recall here the previously best known results on JH. A marginal preimage attack on the 512-bits hash function with a complexity in time and memory of 2^{507} was presented in [1]. Several multi-inbound rebound attacks were presented in [15], providing in particular a semi-free-start collision for 16 rounds with a complexity of 2^{190} in time and 2^{104} in memory and a semi-free-start near-collision for 22 rounds of compression function with a complexity of 2^{168} in time and 2^{143} in memory. In [12, Sec.4.1], improved complexities for these rebound attacks were provided: 2^{97} in time and memory for the 16 round semi-free-start collision and 2^{96} in time and memory for the 22 rounds semi-free-start near-collision for compression function.

Our Contributions: In this paper we apply, as in [15], a multi-inbound rebound attack, using 6 inbounds that cover rounds from 0 to 32. We first find partial solutions for the differential part of the path by using the ideas from [13]. Due to increased number of rounds compared with the previous attacks, the differential path will have several highly active peaks, instead of one as in [15]. This means that, while in the previous attacks finding the whole solution for the path could be easily done without contradicting any of the already fixed values from the inbounds, now finding the complete solution is the most expensive part. We propose here an algorithm that allows us to find whole solutions for rounds from 4 to 26 with an average complexity of 2^{64}. By repeating the algorithm, the attack can be started from round 0 and extended up to 37 rounds for building semi-free-start near-collisions on the internal state, since we have enough degrees of freedom. Based on the same differential characteristic, we also present distinguishers for 42 rounds of the internal permutation which is the first distinguisher on internal permutation faster than generic attack to the best of our knowledge. We summarize our main results in Table 1.

This paper is organized as follows: In Section 2, we give a brief description of the JH hash function, its properties and an overview of the rebound attack. In

Table 1. Comparison of best attack results on JH (sfs: semi-free-start)

target	rounds	time comp.	memory comp.	attack type	generic comp.	sect.
hash function	16	2^{190}	2^{104}	sfs collision	2^{256}	[15]
hash function	16	$2^{96.1}$	$2^{96.1}$	sfs collision	2^{256}	[12]
comp. function	$19-22$	2^{168}	$2^{143.7}$	sfs near-collision	2^{236}	[15]
comp. function	$19-22$	$2^{95.6}$	$2^{95.6}$	sfs near-collision	2^{236}	[12]
comp. function	26	2^{112}	$2^{57.6}$	sfs near-collision	$2^{341.45}$	§3
comp. function	32	2^{304}	$2^{57.6}$	sfs near-collision	$2^{437.13}$	§3
comp. function	36	2^{352}	$2^{57.6}$	sfs near-collision	$2^{437.13}$	§3
comp. function	37	2^{352}	$2^{57.6}$	sfs near-collision	$2^{396.7}$	§3
internal perm.	42	2^{304}	$2^{57.6}$	distinguisher	2^{705}	§4
internal perm.	42	2^{352}	$2^{57.6}$	distinguisher	2^{762}	§4

Section 3, we first describe the main idea of our attack and then give the semi-free internal near-collision results on the tweaked version JH42. Based on this results, we describe a distinguisher in Section 4 for the full internal permutation, that also applies to the full compression function. Finally, we conclude the paper and summarize our results in Section 5.

2 Preliminaries

2.1 The JH42 Hash Function

The hash function JH is an iterative hash function that accepts message blocks of 512 bits and produces a hash value of 224, 256, 384 and 512 bits. The message is padded to be a multiple of 512 bits. The bit '1' is appended to the end of the message, followed by $384-1+(-l \bmod 512)$ zero bits. Finally, a 128-bit block is appended which is the length of the message, l, represented in big endian form. Note that this scheme guarantees that at least 512 additional bits are padded.

In each iteration, the compression function F_d, given in Figure 1, is used to update the 2^{d+2} bits of the state H_{i-1} as follows:

$$H_i = F_d(H_{i-1}, M_i)$$

where H_{i-1} is the previous chaining value and M_i is the current message block. The compression function F_d is defined as follows:

$$F_d(H_{i-1}, M_i) = E_d(H_{i-1} \oplus (M_i || 0^{2^{d+1}})) \oplus (0^{2^{d+1}} || M_i)$$

Here, E_d is a permutation and is composed of an initial grouping of bits followed by $6(d-1)$ rounds, plus a final degrouping of bits. The grouping operation arranges bits in a way that the input to each S-Box has two bits from the message part and two bits from the chaining value. In each round, the input is

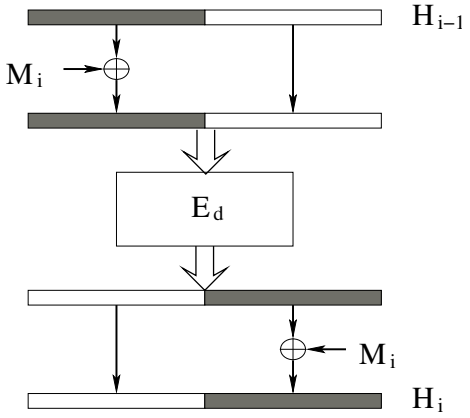


Fig. 1. The compression function F_d that transforms 2^{d+2} bits treated as 2^d words of four bits.

divided into 2^d words and then each word passes through an S-Box. JH uses two 4-bit-to-4-bit S-Boxes ($S0$ and $S1$) and every round constant bit selects which S-Boxes are used. Then two consecutive words pass through the linear transformation L, which is based on a $[4, 2, 3]$ Maximum Distance Separable (MDS) code over $GF(2^4)$. Finally all words are permuted by the permutation P_d. After the degrouping operation each bit returns to its original position.

The initial hash value H_0 is set depending on the message digest size. The first two bytes of H_{-1} are set as the message digest size, and the rest of the bytes of H_{-1} are set as zero. Then, $H_0 = F_d(H_{-1}, 0)$. Finally, the message digest is generated by truncating H_N where N is the number of blocks in the padded message, i.e, the last X bits of H_N are given as the message digest of JH-X where X = 224, 256, 384 and 512.

The official submitted version of JH42 has $d = 8$ and so the number of rounds is 42 and the size of the internal state is 1024 bits. Then, from now on, we will only consider E_8. For a more detailed information we refer to the specification of JH [20].

2.2 Properties of the Linear Transformation L

Since the linear transformation L implements a $[4, 2, 3]$ MDS code, any difference in one of the words of the input (output) will result in a difference in two words of the output (input). For a fixed L transformation, if one tries all possible 2^{16} pairs, the number of pairs satisfying the condition $2 \rightarrow 1$ or $1 \rightarrow 2$ is 3840, which gives a probability of $3840/65536 \approx 2^{-4.09}$. Note that, if the words are arranged in a way that they will be both active this probability increases to $3840/57600 \approx 2^{-3.91}$. For the latter case, if both words remain active ($2 \rightarrow 2$), the probability is $49920/57600 \approx 2^{-0.21}$.

2.3 Observations on the Compression Function

The grouping of bits at the beginning of the compression function assures that the input of every first layer S-Box is xor-ed with two message bits. Similarly, the output of each S-Box is xor-ed with two message bits. Therefore, for a random non-zero 4-bit difference, the probability that this difference is related to a message is $3/15 \approx 2^{-2.32}$.

The bit-slice implementation of F_d uses $d-1$ different round functions. The main difference between these round functions is the permutation function. In each round permutation, the odd bits are swapped by $2^r \bmod (d-1)$ where r is the round number. Therefore, for the same input passing through multiple rounds, the output is identical to the output of the original round function for the $\alpha \cdot (d-1)$-th round where α is any integer.

2.4 The Rebound Attack

The rebound attack was introduced by Mendel et al. [10]. The two main steps of the attack are called inbound phase and outbound phase. In the inbound phase, the available degrees of freedom are used to connect the middle rounds by using the match-in-the-middle technique and in the outbound phase connected truncated differentials are computed in both forward and backward direction.

This attack has been first used for the cryptanalysis of reduced versions of Whirlpool and Grøstl, and then extended to obtain distinguishers for the full Whirlpool compression function [6]. Later, linearized match-in-the-middle and start-from-the-middle techniques are introduced by Mendel et al. [9] to improve the rebound attack. Moreover, a sparse truncated differential path and state is used in the attack on LANE by Matusiewicz et al. [8] rather than using a full active state in the matching part of the attack. Then, these techniques were used to improve the results on AES-based algorithms in the following papers: [2,3,5,11,14,17,18].

3 Semi-free-start internal near-collisions

In this section, we first present an outline for the rebound attack on reduced round versions of JH for all hash sizes. We use a differential characteristic that covers 32 rounds, and apply the start-from-the-middle technique by using six inbound phases with partially active states. We first describe how to solve the multi-inbound phase for the active bytes. Contrary to previous attacks on JH, we now have more fixed values from the inbound phases. So, in order to find a complete solution, we need to merge these fixed values without contradicting any of them. Therefore, we describe next how to match the passive bytes. Finally, we analyze the outbound part.

3.1 Matching the Active Bytes

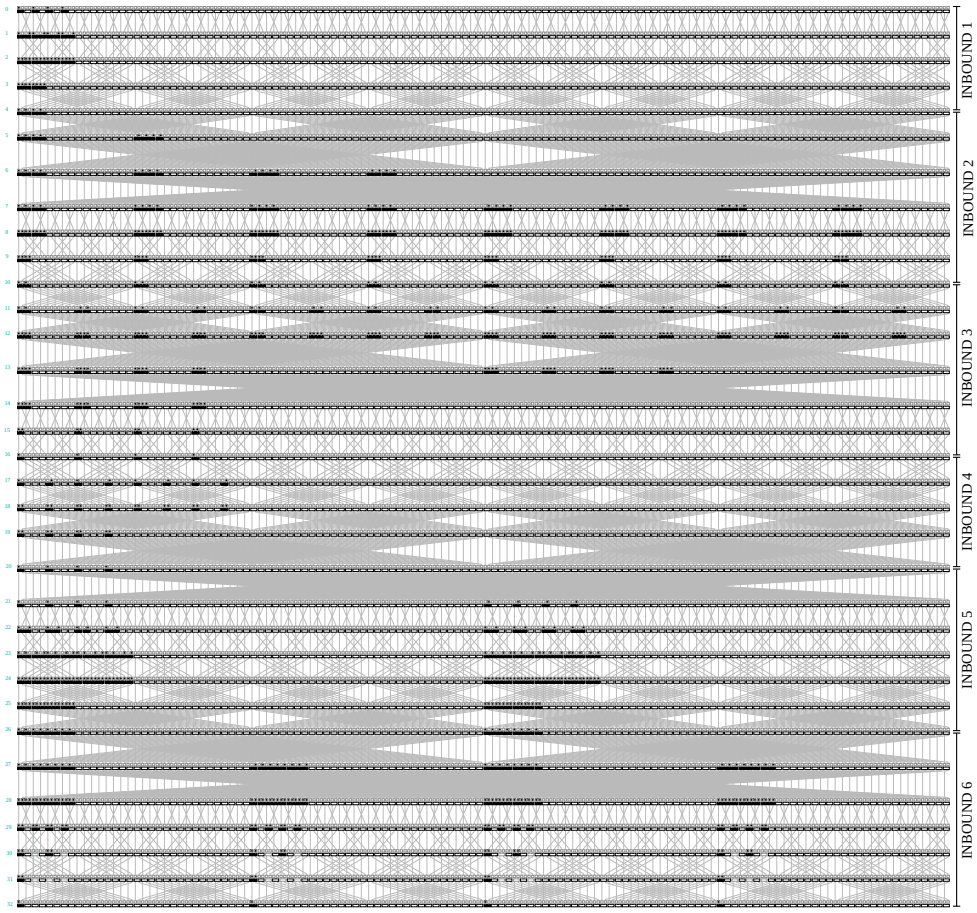


Fig. 2. Differential characteristic for 32 rounds of JH Compression Function (bit-slice representation)

Multi-inbound Phase: The multi-inbound phase of the attack covers 32 rounds and is composed of two parts. In the first part, we apply the start-from-the-middle-technique six times for rounds $0-4$, $4-10$, $10-16$, $16-20$, $20-26$ and $26-32$. In the second part, we connect the resulting active bytes (hence the corresponding state values) by a match-in-the-middle step. The number of active S-Boxes in each of the sets is:

$$4 \leftarrow 8 \leftarrow 16 \rightarrow 8 \rightarrow 4 \tag{1}$$

$$4 \leftarrow 8 \leftarrow 16 \leftarrow 32 \leftarrow 64 \rightarrow 32 \rightarrow 16 \tag{2}$$

$$16 \leftarrow 32 \leftarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \tag{3}$$

$$4 \leftarrow 8 \leftarrow 16 \rightarrow 8 \rightarrow 4 \tag{4}$$

$$4 \leftarrow 8 \leftarrow 16 \leftarrow 32 \leftarrow 64 \rightarrow 32 \rightarrow 16 \tag{5}$$

$$16 \leftarrow 32 \leftarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \tag{6}$$

Here, the arrows represent the direction of the computations for the inbound phases and for a detailed sketch we refer to Figure 2. We start from the middle and then propagate outwards by computing the cross-product[3] of the sets and using the filtering conditions. For each inbound we try all possible 2^{16} pairs in Step 0. The number of sets, the bit length of the middle values (size) of each list, and the number of filtering conditions on words followed by the number of pairs in each set are given in Table 2. The complexities given in the Table 2 are not optimized yet, we will describe the improved complexities later in Section 3.1.

Merging Inbound Phases: The remaining pairs at inbound i are stored on list L_i. Connecting the six lists is performed in three steps as follows:

1. Whenever a pair is obtained from set 2, we check whether it exists in L_3 or not. If it does, another check is done for L_1. Since we have $2^{23.44}$ and $2^{83.96}$ elements in lists 1 and 3 respectively, $2^{83.96}$ pairs passing the second inbound phase, and 32-bit and 128-bit conditions for the matches, the expected number of remaining pairs is $2^{23.44} \cdot 2^{-32} \cdot (2^{83.96} \cdot 2^{-128} \cdot 2^{83.96}) = 2^{31.36}$. We store these these pairs in list A.

2. Similarly, whenever a pair is obtained from set 5, we check whether it exists in L_6 or not. If it does, another check is done for L_4. Since we have $2^{32.72}$ and $2^{83.96}$ elements in lists 4 and 6 respectively, 2^{80} pairs passing the fifth inbound phase, and 32-bit and 128-bit conditions for the matches, the expected number of remaining pairs is $2^{32.72} \cdot 2^{-32} \cdot (2^{83.96} \cdot 2^{-128} \cdot 2^{83.96}) = 2^{40.64}$. We store these pairs in list B.

3. Last step is merging these sets A and B. We have $2^{31.36}$ elements in A and $2^{40.64}$ elements in B and 32 bits of condition. Therefore the total expected number of remaining pairs is $2^{31.36} \cdot 2^{-32} \cdot 2^{40.64} = 2^{40}$.

Improving the complexity of finding a solution for the differential part: We have described how to obtain the existing 2^{40} solutions for the differential part. We are going to describe here a better way of doing the inbounds, as proposed in [12, Sec.4.1]. This new technique allows us to reduce the previous complexity from $2^{99.70}$ in time and $2^{83.96}$ in memory to $2^{69.6}$ in time and $2^{67.6}$ in memory. As in our further analysis we will just use one solution (and not 2^{40}) for the differential part, we will adapt the values being able to finally reduce the complexity of this part of the attack to $2^{59.6}$ in time and $2^{57.6}$ in memory. This memory is the memory bottleneck of all the analysis presented in this paper.

1. We consider the six inbounds as described in the previous section, with the difference that, for inbounds $2, 3, 5$ and 6 we will not perform the last step,

[3] cross-product is an operation on two arrays that results in another array whose elements are obtained by combining each element in the first array with every element in the second array.

Table 2. Overview of inbound phases of the attack on 32 rounds of JH

	Step	Size	Sets	Filtering Conditions	Pairs Remaining	Complexity Backwards	Complexity Forwards
Inbound 1	0	8	8	1	$2^{11.91}$	–	2^{16}
	1	16	4	2	2^{16}	$2^{23.91}$	–
	2	32	2	2	$2^{24.18}$	$2^{32.09}$	–
	3	64	1	4	$2^{32.72}$	$2^{48.46}$	–
	4	64	1	4[a]	$2^{23.44}$		
Inbound 2	0	8	32	1	$2^{11.91}$	–	2^{16}
	1	16	16	2	2^{16}	$2^{23.91}$	–
	2	32	8	2	$2^{24.18}$	–	$2^{32.09}$
	3	64	4	4	$2^{32.72}$	$2^{48.46}$	–
	4	128	2	4	$2^{49.80}$	$2^{65.54}$	–
	5	256	1	4	$2^{83.96}$	$2^{99.70}$	–
Inbound 3	0	8	32	1	$2^{11.91}$	–	2^{16}
	1	16	16	2	2^{16}	$2^{23.91}$	–
	2	32	8	2	$2^{24.18}$	$2^{32.09}$	–
	3	64	4	4	$2^{32.72}$	–	$2^{48.46}$
	4	128	2	4	$2^{49.80}$	–	$2^{65.54}$
	5	256	1	4	$2^{83.96}$	–	$2^{99.70}$
Inbound 4	0	8	8	1	$2^{11.91}$	–	2^{16}
	1	16	4	2	2^{16}	$2^{23.91}$	–
	2	32	2	2	$2^{24.18}$	$2^{32.09}$	–
	3	64	1	4	$2^{32.72}$	$2^{48.46}$	–
Inbound 5	0	8	32	1	$2^{11.91}$	–	2^{16}
	1	16	16	2	2^{16}	$2^{23.91}$	–
	2	32	8	2	$2^{24.18}$	–	$2^{32.09}$
	3	64	4	4	$2^{32.72}$	$2^{48.26}$	–
	4	128	2	4	$2^{49.80}$	$2^{65.54}$	–
	5	256	1	4	$2^{83.96}$	$2^{99.70}$	–
Inbound 6	0	8	32	1	$2^{11.91}$	–	2^{16}
	1	16	16	2	2^{16}	$2^{23.91}$	–
	2	32	8	2	$2^{24.18}$	$2^{32.0}$	–
	3	64	4	4	$2^{32.72}$	–	$2^{48.46}$
	4	128	2	4	$2^{49.80}$	–	$2^{65.54}$
	5	256	1	4	$2^{83.96}$	–	$2^{99.70}$

[a] Check whether the pairs satisfy the desired input difference

but instead we obtain for each inbound $i \in \{2, 3, 5, 6\}$ two lists $L_{A,i}$ and $L_{B,i}$ as a result, each of size $2^{49.80}$ associated to half of the corresponding differential path. As mentioned before, we are only looking to find one solution for the whole differential path. Then, instead of the $2^{49.80}$ existing solutions for each list, we can consider $2^{44.8}$ elements on each list.

2. First, we merge lists $L_{A,2}$ and $L_{A,3}$. We have 16-bit conditions on values and 16-bit conditions on differences. We obtain a new list $L_{A,23}$ of size

$2^{44.8+44.8-32} = 2^{57.6}$. We do the same with $L_{B,2}$ and $L_{B,3}$ to obtain $L_{B,23}$. Note that this list does not need to be stored, as we can perform the following step whenever an element is found.

3. In order to find a whole solution for the differential part of inbounds 2 and 3, one pair of elements from $L_{A,23}$ and from $L_{B,23}$ still needs to satisfy the following conditions: 32 bits from the parts $L_{A,2}$ and $L_{B,3}$, 32 bits from $L_{B,2}$ and $L_{A,3}$, 3.91×4 from the step 5 of inbound 2 that we have not yet verified and 3.91×4 from step 5 of inbound 3 that is not yet verified either. Therefore, we have 95.28-bit conditions in total to merge $L_{A,23}$ and $L_{B,23}$. For each element in $L_{B,23}$ we can check with constant cost if the corresponding element appears in $L_{A,23}$ (it can be done by a lookup in a table, representing the differential transitions of L and next by a lookup in the list $L_{A,23}$ to see if the wanted elements appear. See [13,12] and Figure 3 for more details). When we find a good pair, we store it in the list L_{23} that has a size of about $2^{19.92}$ elements satisfying the differential part of rounds from 4 to 16. The cost of this step is then $2^{57.6+1}$ in time and $2^{59.6}$ in memory.
4. Do the same with inbounds 5 and 6, to obtain list L_{56} of size $2^{19.92}$, with a cost of $2^{57.6+1}$ in time and $2^{57.6}$ in memory.
5. Merge the solutions obtained in the first inbound with the ones in L_{23}, obtaining a new set L_{123} of size $2^{19.92+23.44-32} = 2^{11.36}$.
6. Merge the solutions obtained from step 4 with list L_{56} obtaining a new one, L_{456} of size $2^{19.92+32.72-32} = 2^{20.64}$.
7. Finally, merging L_{123} and L_{456} gives $2^{11.36+20.64-32} = 1$ partial solution for the differential part of the path from round 0 to round 32.

The complexity of obtaining one partial solution for rounds from 0 to 32 is dominated by Steps $2-4$ of the algorithm. As a result, the complexity of matching the active bytes becomes $2^{59.6}$ in time and $2^{57.6}$ in memory.

3.2 Matching The Passive Bytes

In Figure 4, colored boxes denote the S-boxes whose values have already been fixed from the inbound phases. Note that, we have not treated the passive bits yet (i.e., found the remaining values that would complete the path). We will propose a way of finding 2^{32} solutions that verify the path from rounds 4 to 26 with time complexity 2^{96} and memory complexity $2^{51.58}$. This can be done in three steps as follows:

1. (Rounds 10 to 14): The sets of groups of 8 bits denoted by a, b, c, d, e, f in round 14 are independent of each other in this part of the path. In round 10, 32 bits are already fixed for each of these sets (groups of 4 bits denoted by A, B, C, D, E, F). By using all possible values of the remaining 96 passive bits (32 bits not fixed from A, B, C, D, E, F plus 64 from the remaining state at round 10), we can easily compute a list of 2^{96} elements with cost 2^{96} that satisfy the 32 bit conditions for each of the groups.

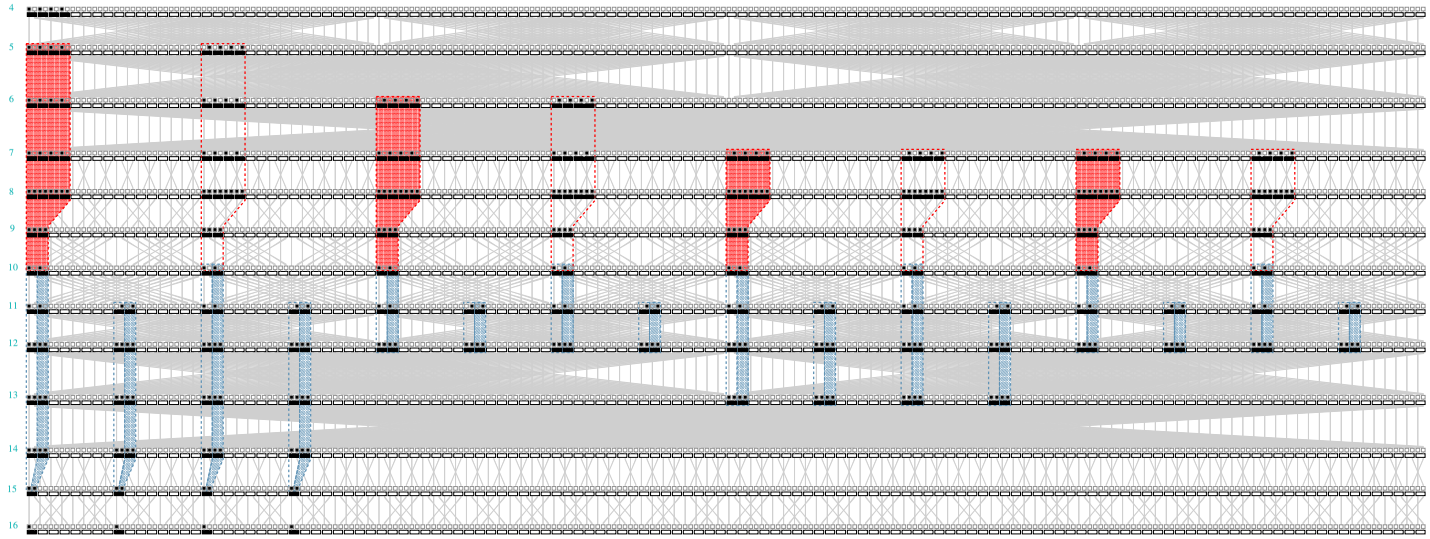


Fig. 3. Improving the complexity of finding a solution for the differential part for rounds $4-16$: Red boxes (above) denote the sets $L_{A,2}$ (filled) and $L_{B,2}$, blue boxes (below) denote the sets $L_{A,3}$ and $L_{B,3}$ (filled).

2. (Rounds 14 to 20): In round 20, we have 256 bits (green S-boxes ■) whose values are fixed from the solutions of the second inbound phase. We can divide the state in round 19 (until the state in round 14) in 4 independent parts (m, n, o, p). In Figure 4, the fixed bits coming from round 20 are denoted by green lines and the ones of the first inbound phase are denoted in blue "■". Note that the three parts m, n, o are identical, while p is different since there are some differences and some additional fixed values in it.

   We fix the parts m and n to some values that satisfy all the conditions of the fixed bits in rounds 19 and 14. This can be done as follows: Similar to what we have done in step 1, we can divide the state of rounds $16 - 19$ (for each part separately) into four groups (x, y, z, u) such that they are independent of each other when computing forwards.

   In round 16, each group has 16 bits whose values have already been fixed and 48 bits of freedom. We see that each group affects only one fourth of the green lines (16 bits in total) in round 19. Therefore, there exist $2^{48-16} = 2^{32}$ possibilities for each group x, y, z, u but we just need one. This one can then be found with a cost of about 2^{16}

3. (Merging) Each of the sets $L_a, \ldots, L_f$ has 2^{96} possible values from step 1, and fixing m and n fixes 64 bits for each of them in round 14. This gives us in average $2^{96-64} = 2^{32}$ possible values for each set in the half of the state associated to o and p in round 14.

   For the part p we use the same idea explained in step 2. Group x is completely fixed due to the differential characteristic, and only the groups y, z, u have freedom, so there exists $(2^{32})^3 = 2^{96}$ possibilities. For each possibility, we compute the part of state in round 14 associated to p. We have 32 bits of condition for each of lists, and in average 2^{32} values are associated to each list. Thus, for each of the computed values, we will have only one remaining element that will determine the values at positions $a - f$ in the part o.

   Now, we have 2^{96} possible o values. The probability that a fixed value verifies the conditions of o in round 19 is $(2^{-4})^{16} = 2^{-64}$. Therefore, we obtain $2^{96-64} = 2^{32}$ solutions that verify the whole path from round 4 to round 26 with a complexity in time of 2^{96}.

Note that we do not need to store the lists $L_a, \ldots, L_f$ of elements from round 14 each of size 2^{96} but we can instead store for each of them two lists of size 2^{48} corresponding to the upper and down halves of the corresponding groups in state 13. Then, when fixing a value of m and n we can check with a cost of 2^{32} which will be the list of 2^{32} values for o and p that we obtained in step 3. Finally, we have obtained 2^{32} complete solutions for the path from 4 to 26 with a cost of 2^{96} in time, and $6 \cdot 2 \cdot 2^{48} \approx 2^{51.58}$ in memory.

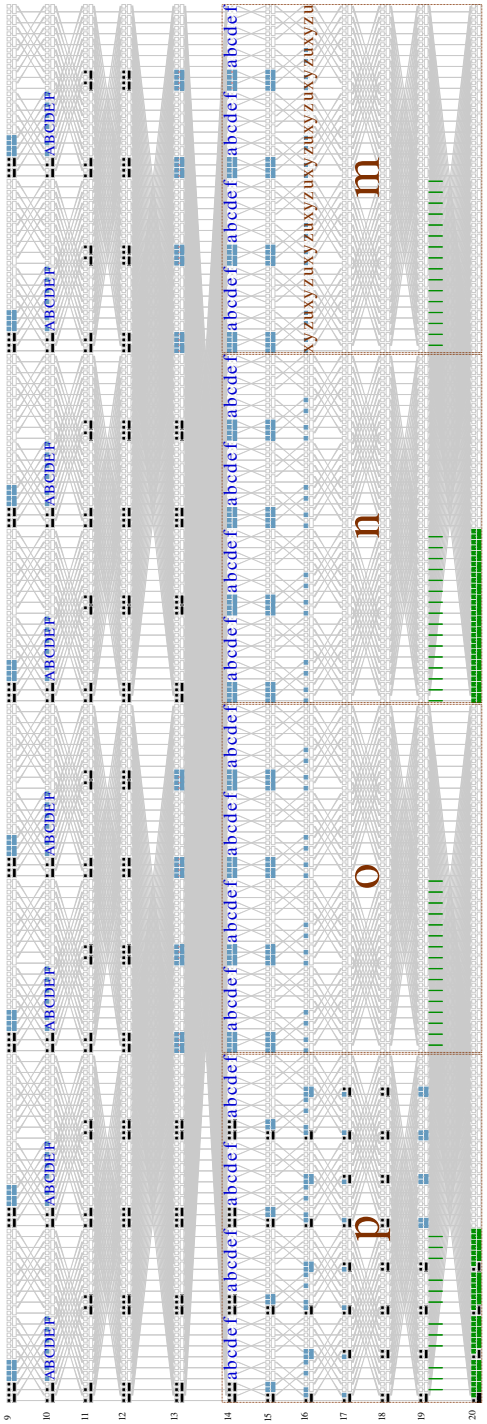


Fig. 4. Matching the passive bytes (bit-slice representation)

Semi-free-start near-collisions up to 32 rounds: Up to now, we have found solutions for the passive bytes from rounds $4-26$. If we want a solution for the path from round 0 to round 26, we will have to repeat the previous procedure of matching the passive bytes 2^{16} times (as the probability of passing from round 0 to 4 is 2^{-48} and we have 2^{32} pairs). Then, we can find a solution for rounds $0-26$ with complexity 2^{112} in time. In order to extend this result to 32 rounds, we have to repeat the previous procedure 2^{192} times (since we have 64 and 128 bits of condition from rounds 26 and 27 respectively). Therefore, the complexity for finding a complete solutions for rounds from 0 to 32 is $2^{112} \cdot 2^{192} = 2^{304}$ in time.

Note that, we still have enough degrees of freedom. In step 1, we started with 768 bits (128×6 from the groups $a-f$) in round 14 and matched 192 bits (32×6 for $A-F$) in round 10. In Step 2, we have 48 bits in round 16 coming from the fourth inbound phase and we matched another 240 bits from the fifth inbound phase in round 19. So in total we have $768-192-48-240 = 288$ bits of degrees of freedom remaining.

3.3 Outbound Phase:

The outbound phase of the attack is composed of 5 rounds in the forward direction. A detailed schema of this trail is shown in Figure 5 in appendix, and for the pairs that satisfy the inbound phase, we expect to see the following differential trail in the outbound phase:

$$\text{Inbound Phase } \rightarrow 4 \rightarrow 8 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 8$$

Semi-free-start near-collisions up to 37 rounds: For 32 rounds of the JH compression function, we obtain a semi-free-start near-collision for 1002 bits. We can simply increase the number of rounds by proceeding forwards in the outbound phase. Note that, we have an additional probability of $2^{-32} \times 2^{-16}$ coming from the eight filtering conditions in round 34 and the four filtering conditions in round 35. Thus, the complexity of the active part of the attack remains the same: $2^{59.6}$ in time and $2^{57.6}$ in memory. This is the case as one solution for the differential part is enough for the attack, as it will have different values at the bits with conditions in the outbound part when the passive part is modified. The complexity of the passive part becomes $2^{304} \cdot 2^{48} = 2^{352}$ in time and $2^{51.58}$ in memory.

The details can be seen in Table 3. We also take into account the colliding bits that we obtain at the output of the compression function after the final degrouping with the differences from the message.

Table 3. Comparison of complexity of the generic attack for near-collisions and our results

#Rounds	# Colliding bits	Generic Attack Complexity	Our Results
23	892	$2^{230.51}$	$2^{59.6}$
$24-26$	762	$2^{99.18}$	$2^{59.6}$ [a]
26	960	$2^{341.45}$	2^{112}
27	896	$2^{236.06}$	2^{112}
32	1002	$2^{437.12}$	2^{304}
33	986	$2^{396.77}$	2^{304}
34	954	$2^{329.97}$	2^{304}
35	986	$2^{396.77}$	2^{336}
36	1002	$2^{437.12}$	2^{352}
37	986	$2^{396.77}$	2^{352}
38	928	$2^{284.45}$	2^{352}

[a] Obtained directly from the solutions of the active part, without need of matching the passive bits

4 Distinguishers on JH

Indifferentiability is considered to be a desirable property of any secure hash function design. Moreover, for many of the designs, the indifferentiability proofs for the mode of operation are based on the assumption that the underlying permutation (function) is ideal (i.e., random permutation). This is the case of the indifferentiability proof of JH [1], that supposes that E_d is a random permutation.

In this section, we present a distinguisher for E_8 showing that it is distinguishable form a random permutation. Using the differential path that we presented in the previous section, we can build the distinguishers on the full 42 rounds of the internal permutation E_8 with no additional complexity. As a result of our distinguisher, the proof from [1] does not apply to JH as the assumption of E_8 behaving like random does not hold. Next, we explain how these distinguishers on the internal permutation can be easily extended to distinguishers on the compression function.

There exists also a known trivial distinguisher on the construction of the compression function of JH: If the chaining value has a difference that can be cancelled by the message block, then the output will have a difference directly related to the one coming from the message block. This implies that both the message and the chaining values have differences. Contrary to the trivial one, our compression function distinguisher exploits the properties of the internal permutation and only needs differences in the message or in the chaining value.

4.1 Distinguishers on the reduced round internal permutation

Let us remark here briefly that if we find solutions for rounds 4 to 20, and then let them spread freely backward (difference in 64 bits) and forward (difference in 256 bits), we can obtain a distinguisher for 26 rounds with a much lower complexity: $2^{59.6}$ in time and $2^{57.6}$ in memory (the cost of the differential part). As in this paper the aim is reaching a higher number of rounds, we do not go further into the details.

4.2 Distinguishers on the full internal permutation

In the previous sections we showed that a solution for 37 rounds can be obtained with a time complexity of 2^{352} in time and $2^{57.6}$ in memory. In Figure 5 from the appendix, we see how these active words diffuse to the state after 42 rounds with probability one. Therefore, before the degrouping operation we have 64 active and 192 passive words in the state. The number of active and passive bits still remain the same after the degrouping operation. It is important to remark that the positions of the active bits are fixed, also after the degrouping operation.

We can then build a distinguisher that will distinguish the 42-round permutation E_8 from a random permutation using this path. This distinguisher aims at finding a pair of input states (A, A') such that $E_8(A) \oplus E_8(A')$ collide in the 768 bits mentioned above. Let $A \oplus A' = \Delta_1$ correspond to the input difference of the differential path, then $|\Delta_1| = 8$ bits. Similarly, let $B = E_8(A)$ and $B' = E_8(A')$, then the output difference is $B \oplus B' = \Delta_2$ where $|\Delta_2| = 256$.

In the case of a random function, we calculate the complexity of such a distinguisher as follows: We fix the values of the passive bits in the input; but not the ones of the active bits. Then, we have $2^{|\Delta_1|}$ possibilities for the values from the active bits. We compute the output of E_8 for each one of these values and store them in a list. From this list we can obtain $\binom{2^{|\Delta_1|}}{2}$ pairs with the given input difference pattern. The probability of satisfying the desired output difference pattern is $2^{|\Delta_2|-1024}$ for each pair, so we repeat the procedure with a new value for the input passive bits until we find a solution. The time complexity of finding such an input pair will be:

$$\frac{2^{|\Delta_1|}}{2^{(|\Delta_1|-1)} \cdot (2^{|\Delta_1|} - 1) \cdot 2^{|\Delta_2|-1024}} = 2^{761}.$$

Instead, in our case the complexity of finding such an input pair is the complexity of finding a solution for the path, that is 2^{352} in time and $2^{57.6}$ in memory.

Another distinguisher of E_8 can be built if we consider the scenario where the differential path for rounds $0-4$ does not need to be verified, i.e., $|\Delta_1| = 64$. In this case, we consider that from round 4 to 0 we obtain the differences that propagate with probability one. Therefore, the matching of the passive part does not need to be repeated 2^{208} times but only 2^{160} (as we do not need 2^{48} extra repetitions for verifying rounds 0 to 4). The complexity of this distinguisher will then be 2^{304}, and provides a pair of inputs A and A' that produce an output

with 768 colliding bits as the ones represented in Figure 5 from appendix. The complexity of such a generic distinguisher would be $\frac{2^{64}}{(2^{64}-1)\cdot 2^{63-768}} = 2^{705}$, while in our case is 2^{304} in time and $2^{57.6}$ in memory.

4.3 Distinguishers on the full compression function

We should emphasize that our distinguishers on E_8 can be easily converted to a distinguisher on the full compression function of JH42. We only need to xor this message difference to the output of E_8 as specified.

For our first distinguisher, the input difference is already arranged such that we only have difference in the message. These active bits coming from the message coincide with the active bits in the output at the xor operation. As a result, we have the same 768 passive bits. The same applies for our second distinguisher when we have differences only in the chaining value.

5 Conclusion

In this paper, we have presented semi-free-start internal near-collisions up to 37 rounds by using rebound attack techniques. We first obtained a 960-bit semi-free-start near-collision for 26 rounds of the JH compression function with a time complexity of 2^{112} and a memory complexity of $2^{57.6}$. We then extended this to 986-bit semi-free-start near-collision for 37 rounds by repeating the algorithm. Time complexity of the attack is increased to 2^{352} and the memory complexity remains the same. We also presented semi-free-start near-collision results for intermediate rounds $26-37$ in Table 3. Our findings are summarized in Table 1.

Even more, we have presented distinguishers on the full 42 rounds of the internal permutation E_8 of the tweaked SHA-3 finalist JH. The best distinguisher has a time complexity of 2^{304} in time and $2^{57.6}$ in memory and provides solutions for the differential path on the 42 rounds. Obtaining such a pair of inputs producing a same truncated differential in the output for a random function would cost 2^{705} in time. Our internal permutation distinguishers can easily be extended to compression function distinguishers with the same complexity.

Although our results do not present a threat to the security of the JH hash function, they invalidate the JH indifferentiability proof presented in [1].

References

1. Bhattacharyya, R., Mandal, A., Nandi, M.: Security Analysis of the Mode of JH Hash Function. In: Hong and Iwata [4], pp. 168–191
2. Burmester, M., Tsudik, G., Magliveras, S.S., Ilic, I. (eds.): Information Security - 13th International Conference, ISC 2010, Boca Raton, FL, USA, October 25-28, 2010, Revised Selected Papers, Lecture Notes in Computer Science, vol. 6531. Springer (2011)
3. Gilbert, H., Peyrin, T.: Super-sbox cryptanalysis: Improved attacks for aes-like permutations. In: Hong and Iwata [4], pp. 365–383

4. Hong, S., Iwata, T. (eds.): Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers, Lecture Notes in Computer Science, vol. 6147. Springer (2010)
5. Ideguchi, K., Tischhauser, E., Preneel, B.: Improved collision attacks on the reduced-round grøstl hash function. In: Burmester et al. [2], pp. 1–16
6. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In: Matsui [7], pp. 126–143
7. Matsui, M. (ed.): Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings, Lecture Notes in Computer Science, vol. 5912. Springer (2009)
8. Matusiewicz, K., Naya-Plasencia, M., Nikolic, I., Sasaki, Y., Schläffer, M.: Rebound Attack on the Full Lane Compression Function. In: Matsui [7], pp. 106–125
9. Mendel, F., Peyrin, T., Rechberger, C., Schläffer, M.: Improved Cryptanalysis of the Reduced Grøstl Compression Function, ECHO Permutation and AES Block Cipher. In: Jr., M.J.J., Rijmen, V., Safavi-Naini, R. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 5867, pp. 16–35. Springer (2009)
10. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In: Dunkelman, O. (ed.) FSE. Lecture Notes in Computer Science, vol. 5665, pp. 260–276. Springer (2009)
11. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Rebound attacks on the reduced grøstl hash function. In: Pieprzyk, J. (ed.) CT-RSA. Lecture Notes in Computer Science, vol. 5985, pp. 350–365. Springer (2010)
12. Naya-Plasencia, M.: How to Improve Rebound Attacks. Cryptology ePrint Archive, Report 2010/607 (2010), http://eprint.iacr.org/2010/607.pdf, (extended version). urlhttp://eprint.iacr.org/
13. Naya-Plasencia, M.: How to Improve Rebound Attacks. In: Advances in Crypology: CRYPTO 2011. Lecture Notes in Computer Science, vol. 6841, pp. 188–205. Springer (2011)
14. Peyrin, T.: Improved differential attacks for echo and grøstl. In: Rabin, T. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 6223, pp. 370–392. Springer (2010)
15. Rijmen, V., Toz, D., Varici, K.: Rebound Attack on Reduced-Round Versions of JH. In: Hong and Iwata [4], pp. 286–303
16. Rivest, R.L.: The MD5 Message-Digest Algorithm. RFC 1321 (1992), http://www.ietf.org/rfc/rfc1321.txt, http://www.ietf.org/rfc/rfc1321.txt
17. Sasaki, Y., Li, Y., Wang, L., Sakiyama, K., Ohta, K.: Non-full-active super-sbox analysis: Applications to echo and grøstl. In: Abe, M. (ed.) ASIACRYPT. Lecture Notes in Computer Science, vol. 6477, pp. 38–55. Springer (2010)
18. Schläffer, M.: Subspace distinguisher for 5/8 rounds of the echo-256 hash function. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 6544, pp. 369–387. Springer (2010)
19. of Standards, N.I., Technology: FIPS 180-1:Secure Hash Standard (1995), http://csrc.nist.gov, http://csrc.nist.gov
20. Wu, H.: The Hash Function JH. Submission to NIST (2008), http://icsd.i2r.a-star.edu.sg/staff/hongjun/jh/jh_round2.pdf, http://icsd.i2r.a-star.edu.sg/staff/hongjun/jh/jh_round2.pdf

A Outbound Phase Figure

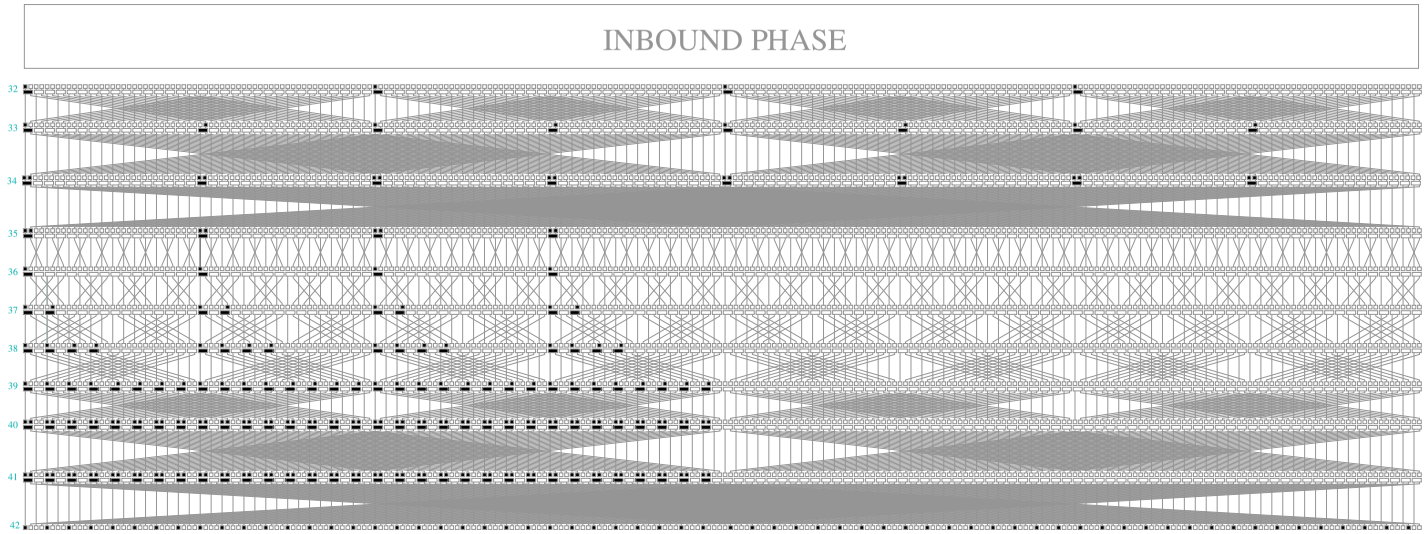


Fig. 5. Differential characteristic for the outbound phase of JH Compression Function (bit-slice representation)

Sieve-in-the-Middle: Improved MITM Attacks (Full Version*)†

Anne Canteaut[1], María Naya-Plasencia[1], and Bastien Vayssière[2]

[1] Inria Paris-Rocquencourt, project-team SECRET
B.P. 105, 78153 Le Chesnay cedex, France
Anne.Canteaut@inria.fr, Maria.Naya_Plasencia@inria.fr
[2] Université de Versailles Saint-Quentin-en-Yvelines
45 avenue des Etats-Unis, 78035 Versailles cedex, France
bastien.vayssiere.w@gmail.com

Abstract. This paper presents a new generic technique, named sieve-in-the-middle, which improves meet-in-the-middle attacks in the sense that it provides an attack on a higher number of rounds. Instead of selecting the key candidates by searching for a collision in an intermediate state which can be computed forwards and backwards, we here look for the existence of valid transitions through some middle sbox. Combining this technique with short bicliques allows to freely add one or two more rounds with the same time complexity. Moreover, when the key size of the cipher is larger than its block size, we show how to build the bicliques by an improved technique which does not require any additional data (on the contrary to previous biclique attacks). These techniques apply to PRESENT, DES, PRINCE and AES, improving the previously known results on these four ciphers. In particular, our attack on PRINCE applies to 8 rounds (out of 12), instead of 6 in the previous cryptanalyses. Some results are also given for theoretically estimating the sieving probability provided by some subsets of the input and output bits of a given sbox.

Keywords. Meet-in-the-middle, block ciphers, bicliques, sbox, matching algorithms.

1 Introduction

Meet-in-the-middle (MITM) attacks are a widely used tool introduced by Diffie and Hellman in 1977. Through the years, they have been applied for analyzing the security of a substantial number of cryptographic primitives, including block ciphers, stream ciphers and hash functions, e.g. [36, 8, 21, 24, 23]. They exploit the fact that some internal state in the middle of the cipher can be computed both forwards from the plaintext and backwards from the ciphertext, and that none of these computations requires the knowledge of the whole master key. The attacker then only keeps the (partial) key candidates which lead to a collision in that internal state and discards all the other keys. This generic attack has drawn a lot of attention and raised many improvements, including the partial matching, where the computed internal states are not completely known, the technique of guessing some bits of the internal state [21], the all-subkeys approach [24], splice-and-cut [3, 4, 22] and bicliques [30]. The most popular application of bicliques is an accelerated exhaustive search on the full AES [6]. But, besides this degenerated application where the whole key needs to be guessed, short bicliques usually allow to increase the number of rounds attacked by MITM techniques without increasing the time complexity, but with a higher data complexity. Moreover, following [11, 12], low-data attacks have attracted a lot of attention, motivated in part by the fact that, in many concrete protocols, only a few plaintext-ciphertext pairs can be eavesdropped. MITM attacks belong

*Full version of the extended abstract published in the proceedings of CRYPTO 2013.

†Partially supported by the French Agence Nationale de la Recherche through the BLOC project under Contract ANR-11-INS-011.

to this class of attacks in most cases (with a few exceptions like bicliques): usually, 1 or 2 known plaintext-ciphertext pairs are enough for recovering the key.

Our contribution. This paper first provides a new generic improvement of MITM algorithms, named *sieve-in-the-middle*, which allows to attack a higher number of rounds. Instead of looking for collisions in the middle, we compute some input and output bits of a particular middle sbox S. The merging step of the algorithm then consists in efficiently discarding all key candidates which do not correspond to a valid transition through S. Intuitively, this technique allows to attack more rounds than classical MITM since it also covers the rounds corresponding to the middle sbox S (e.g. two middle rounds if S is a superbox). This new improvement is related to some previous results, including [3] where transitions through an ARX construction are considered; a similar idea was applied in [29] in a differential attack, and in [13] for side-channel attacks. This new generic improvement can be combined with bicliques, since short bicliques also allow to add a few rounds without increasing the time complexity. But, the price to pay is a higher data complexity. Here, we show that this increased data requirement can be avoided by constructing some improved bicliques, if the key size of the cipher is larger than its block size.

These new improvements and techniques are illustrated with 4 applications which improve previously known attacks. In Section 4, we describe a sieve-in-the-middle attack on 8 rounds of PRESENT, which provides a very illustrative and representative example of our technique. This attack applies up to 8 rounds, while the highest number of rounds reached by classical MITM is only 6. A similar analysis on DES is presented in Section 5; our attack achieves 8 rounds, while the best previous MITM attack (starting from the first one) was on 6 rounds. The cores of these two attacks have been implemented, confirming our theoretical analysis. In Section 6, the sieve-in-the-middle algorithm combined with the improved biclique construction is applied to 8 rounds (out of 12) of PRINCE, with 2 known plaintext-ciphertext pairs, while the previous best known attack was on six rounds. In Section 7, we show that we can slightly improve on some platforms the speed-up factor in the accelerated exhaustive search on the full AES performed by bicliques. The time complexity of the sieve-in-the-middle algorithm highly depends on the sieving probability of the middle sbox, i.e., on the proportion of pairs formed by a partial input and a partial output which correspond to a valid transition for S. We then give some results which allow to estimate the sieving probability of a given sbox. In particular, we show that the sieving probability is related to the branch number of the sbox, and we give a lower bound on the minimal number of known input and output bits which may provide a sieve.

2 The Sieve-in-the-Middle Attack

2.1 Basic idea

The basic idea of the attack is as follows. The attacker knows one pair of plaintext and ciphertext (P, C) (or several such pairs), and she is able to compute from the plaintext and from a part K_1 of the key candidate an m-bit vector u, which corresponds to a part of an intermediate state x. On the other hand, she is able to compute from the ciphertext and another part K_2 of the key candidate a p-bit vector v, which corresponds to a part of a second intermediate state y. Both intermediate states x and y are related by $y = S(x)$, where S is a known function from $\mathbf{F}_2^n$ into $\mathbf{F}_2^{n'}$, possibly parametrized by a part K_3 of the key. In practice, S can be a classical sbox, a superBox or some more complex function, as long as

the attacker is able to precompute and store all possible transitions between the input bits obtained by the forward computation and the output bits obtained backwards (or sometimes, these transitions can even be computed on the fly). In particular, the involved intermediate states x and y usually correspond to partial internal states of the cipher, implying that their sizes n and n' are smaller than the blocksize.

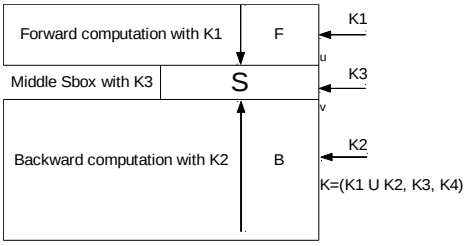


Fig. 1. Generic representation of Sieve-in-the-Middle.

Then, the attacker is able to compute some pairs (u, v) in $\mathbf{F}_2^m \times \mathbf{F}_2^p$ and she wants to determine whether those pairs can be some valid parts of a pair $(x, S(x))$ for some $x \in \mathbf{F}_2^n$ (and for some K_3 if S depends on a part of the key). If it appears that no input $x \in \mathbf{F}_2^n$ can lead to a given (u, v), then the keys (K_1, K_2) from which (u, v) has been obtained do not form a valid candidate for the key. In such a case, the $(m+p)$ positions corresponding to (u, v) can be used as a sieve. The sieving probability is then the proportion of pairs (u, v) corresponding to valid parts of $(x, S(x))$. Obviously, in classical MITM attacks, u and v correspond to the same n-bit part of an intermediate state and $S = \mathrm{Id}_n$; the sieving probability is then equal to 2^{-n}. We now define precisely when a pair (I, J) of input and output positions can be used as a sieve.

Definition 1. *Let S be a function from $\mathbf{F}_2^n$ into $\mathbf{F}_2^{n'}$. Let $I \subset \{1, \ldots, n\}$ and $J \subset \{1, \ldots, n'\}$ be two subsets with respective sizes m and p. The sieving probability of (I, J), denoted by $\pi_{I,J}$, is the proportion of all elements in $\mathbf{F}_2^{m+p}$ which can be written as $(x_i, i \in I; S_j(x), j \in J)$ for some $x \in \mathbf{F}_2^n$. The pair (I, J) is called an (m, p)-sieve for S if $\pi_{I,J} < 1$.*

The smaller $\pi_{I,J}$, the better the sieving, because more candidates will be discarded. If S depends on a k_3-bit value key K_3, the definition similarly applies but S must be seen as a function with $(k_3 + n)$ inputs.

When a large number of inputs and outputs of S can be computed by the attacker, they can be used as a sieve, as shown in the following proposition.

Proposition 1. *Any pair (I, J) of sets of size (m, p) with $m + p > n$ is a sieve for S with sieving probability $\pi_{I,J} \leq 2^{n-(m+p)}$.*

Proof. For any given u, there exists exactly 2^{n-m} values of x such that $(x_i, i \in I) = u$. Thus, $(S_j(x), j \in J)$ can take at most 2^{n-m} different values, implying that $\pi_{I,J} \leq 2^{n-(m+p)}$.

However, smaller subsets I and J may provide a sieve even when $m+p \leq n$. This issue will be extensively discussed in Section 8. More generally, u and v may consist of some information bits of x and y, i.e., of some linear combinations of the bits of x and y. We then define two linear functions $L : x \in \mathbf{F}_2^n \mapsto u \in \mathbf{F}_2^m$ and $L' : y \in \mathbf{F}_2^n \mapsto v \in \mathbf{F}_2^p$. The corresponding sieving

probability π is now the proportion of (u, v) such that there exists $x \in \mathbf{F}_2^n$ with $L(x) = u$ and $L'(S(x)) = v$. Then, π can be seen as the sieving probability of $I = \{1, \ldots, m\}$ and $J = \{1, \ldots, p\}$ for the function $L' \circ S \circ \widetilde{L}^{-1}$ where $\widetilde{L}$ is any linear permutation of $\mathbf{F}_2^n$ such that $(\widetilde{L}(x)_i, i \in I) = L(x)$.

2.2 Description of the attack

We now precisely describe the improved MITM attack and provide its complexity. The secret key K is divided into four (possibly non-disjoint) parts, K_1 to K_4. K_1 (resp. K_2) is the part of the key used in the forward (resp. backward) computation, while K_3 is involved in the middle S function only (see Fig. 1). The key bits corresponding to K_4 are not involved in the MITM step. In the following, k_i denotes the length of the key part K_i, while k is the total key length. Moreover, $K_1 \cap K_2$ denotes the bits shared by K_1 and K_2, and κ corresponds to the size of this intersection.

We denote by I (resp. J) the set of input positions of S (resp. output positions) corresponding to u (resp. v). The fact that a pair (u, v) corresponds to a valid pair of inputs and outputs of S is characterized by a Boolean relation $\mathcal{R}$ with $(m + p)$ inputs defined by

$$\mathcal{R}(u, v) = 1 \text{ if and only if } \exists x \in \mathbf{F}_2^n : (x_i, i \in I) = u \text{ and } (S(x)_j, j \in J) = v \ .$$

The attack proceeds as follows.

for all 2^κ values of $K_1 \cap K_2$ **do**
  $\mathcal{L}_f \leftarrow \emptyset$ and $\mathcal{L}_b \leftarrow \emptyset$
  // Forward computation
  for all $2^{k_1-\kappa}$ values of the remaining bits of K_1 **do**
    compute $u = F_{K_1}(P)$ and add u to $\mathcal{L}_f$
  // Backward computation
  for all $2^{k_2-\kappa}$ values of the remaining bits of K_2 **do**
    compute $v = B_{K_2}(C)$ and add v to $\mathcal{L}_b$
  // Merging step
  Merge $\mathcal{L}_f$ and $\mathcal{L}_b$ with respect to Relation $\mathcal{R}$ and return the merged list $\mathcal{L}_{sol}$.
// Testing the remaining candidates
for all K with (K_1, K_2) in $\mathcal{L}_{sol}$ **do**
  if $E_K(P) = C$ **then**
    return K

Section 2.3 details some efficient algorithms for merging the two lists $\mathcal{L}_f$ and $\mathcal{L}_b$ (i.e. for recovering all the (u, v) which satisfy $\mathcal{R}(u, v) = 1$) with complexity lower than the product of their sizes. Two representative examples of application on PRESENT and DES are provided in Section 4 and in Section 5 respectively.

With a single plaintext-ciphertext pair. Obviously, the whole secret key can be recovered only if the key length does not exceed the blocksize. Otherwise, 2^{k-b} possible keys will be returned in average where b is the blocksize. The time complexity of the attack is given by:

$$2^\kappa \left(2^{k_1-\kappa} c_F + 2^{k_2-\kappa} c_B + C_{\mathsf{merge}}\right) + \pi 2^k c_E \ ,$$

where π is the sieving probability of (I, J) as defined in Definition 1, c_E is the cost of one encryption, while c_F and c_B correspond to the costs of a partial encryption in the forward

and backward directions. In most cases, $c_F \simeq c_B \simeq c_E/2$. C_{merge} is the time complexity of the merging step, and it depends on k_3. Its value is discussed in the following section. The average time complexity of the attack needs to be compared to $2^k c_E$ which is the cost of an exhaustive search for the key. The memory complexity is mainly determined by the memory needed in the merging step. In some cases, it can be improved by storing only one among the two lists $\mathcal{L}_f$ and $\mathcal{L}_b$, when the auxiliary lists used in the merging step remain smaller.

With N plaintext-ciphertext pairs. If N plaintext-ciphertext pairs are available to the attacker, then the average number of keys returned by the attack is 2^{k-Nb}, implying that the whole key will be recovered when $N \geq k/b$. The main modification in the attack concerns the last step where all key candidates in $\mathcal{L}_{sol}$ are tested: before performing an exhaustive search over $(K_1 \cap K_2)$ and K_4 for testing all keys with $(K_1, K_2) \in \mathcal{L}_{sol}$, an additional sieving step is performed in order to reduce the size of $\mathcal{L}_{sol}$. Once a new solution $(K_1, K_2) \in \mathcal{L}_{sol}$ has been found, $(N-1)$ additional pairs (u_i, v_i) generated from the other plaintext-ciphertext pairs are considered, and only the keys for which $\mathcal{R}(u_i, v_i) = 1$ are kept in $\mathcal{L}_{sol}$ (note that, in some very particular situations, it might be more efficient to directly include in $\mathcal{L}_f$ and $\mathcal{L}_b$ the values u and v generated from several plaintext-ciphertext pairs, and then merge the lists). The average size of $\mathcal{L}_{sol}$ after this additional sieving step is then $\pi^N 2^{k_1+k_2-2\kappa}$. But this formula should be adapted to the case where S depends on a part of the secret key K_3: indeed the merging step determines a candidate for (K_1, K_2, K_3). Then, the sieving probability of the additional sieving step π' differs from π since the value of K_3 is now fixed. π' is then the sieving probability of (I, J) for S_{K_3} averaged over all K_3. Then, in the case of N plaintext-ciphertext pairs, the cost of the forward and backward computations are multiplied by N, while the cost of the testing part decreases:

$$2^\kappa \left(N 2^{k_1-\kappa} c_F + N 2^{k_2-\kappa} c_B + C_{\mathsf{merge}} \right) + \pi (\pi')^{N-1} 2^k c_E \ .$$

2.3 Merging the two lists efficiently

Very often, the middle function S can be decomposed into several smaller sboxes, and the merging step can be performed group-wise. The problem of merging two large lists with respect to a group-wise Boolean relation has been defined and addressed by Naya-Plasencia in [33, Section 2]. Here, we focus on three algorithms proposed in [33], namely instant matching, gradual matching and an improvement of the parallel matching due to [19]. We provide general and precise formulas for the average time and memory complexities of these three algorithms. Actually, in our case, the lists to be merged may be small. Then, the construction of some auxiliary tables, which had a negligible cost in [33] for large lists, must now be taken into account. It might even become the bottleneck of the algorithm. Thus, when the involved lists are small, it is harder to determine a priori which algorithm is the most efficient in a given case. Then, in each application, we need to check thoroughly which algorithm provides the best complexity. The optimal case may even sometimes correspond to the combination of two algorithms.

In the following, we consider two lists, $\mathcal{L}_A$ of size 2^{ℓ_A} and $\mathcal{L}_B$ of size 2^{ℓ_B}, whose roles are interchangeable. The elements of both lists can be decomposed into t groups: the i-th group of $a \in \mathcal{L}_A$ has size m_i, while the i-th group of $b \in \mathcal{L}_B$ has size p_i. The Boolean relation $\mathcal{R}$ can similarly be considered group-wise: $\mathcal{R}(a, b) = 1$ if and only $\mathcal{R}_i(a_i, b_i) = 1$ for all $1 \leq i \leq t$. The sieving probability π associated to $\mathcal{R}$ then corresponds to the product of the sieving

probabilities π_i associated to each $\mathcal{R}_i$. Since each $\mathcal{R}_i$ corresponds to an sbox S_i with n_i-bit inputs, a table storing all (a_i, b_i) such that $\mathcal{R}_i(a_i, b_i) = 1$ can be built with time complexity 2^{n_i}, by computing all $(x_i, S_i(x_i)), x_i \in \mathbf{F}_2^{n_i}$. The corresponding memory complexity is proportional to $\pi_i 2^{m_i+p_i}$. This cost won't be included in the cost of the merging algorithm since, in the sieve-in-the-middle process, the tables will be built once for all and not 2^κ times. As we will see, in some situations, these tables can be built "on-the-fly" with much fewer operations.

We now provide complete description of the three matching algorithms. For the sake of simplicity, we assume in the description of the algorithms that the lists are sorted, but in practice we can use standard hash tables for storage and lookup in constant time, since the keys are integers. It is worth noticing that the size of the list $\mathcal{L}_{sol}$ returned by the matching algorithm is not included in the memory complexity since each of its elements can be tested in the attack as soon as it has been found.

Instant Matching. Instant matching successively considers all elements $\mathcal{L}_B$: for each $b \in \mathcal{L}_B$, a list $\mathcal{L}_{aux}$ of all a such that $\mathcal{R}(a, b) = 1$ is built, and each element of $\mathcal{L}_{aux}$ is searched within $\mathcal{L}_A$.

Algorithm 1 Instant matching algorithm of $\mathcal{L}_A$ and $\mathcal{L}_B$ with respect to $\mathcal{R}$.

1: **for** j from 1 to t **do**
2: Build the table T_j such that $T_j[v_j]$ corresponds to all u_j with $\mathcal{R}_j(u_j, v_j) = 1$.
3: **for** each $(b_1, \ldots, b_t) \in \mathcal{L}_B$ **do**
4: $\mathcal{L}_{aux} \leftarrow \emptyset$.
5: **for** j from 1 to t **do**
6: **if** $T_j[b_j]$ is empty, **then** go to 3.
7: Add all tuples $(x_1, \ldots, x_t)$ with $x_j \in T_j[b_j]$, $\forall j$, to $\mathcal{L}_{aux}$.
8: **for** each $(x_1, \ldots, x_t)$ in $\mathcal{L}_{aux}$ **do**
9: **if** $(x_1, \ldots, x_t) \in \mathcal{L}_A$ **then**
10: Add $(x_1, \ldots, x_t, b_1, \ldots, b_t)$ to $\mathcal{L}_{sol}$.
11: Return $\mathcal{L}_{sol}$.

$$\text{Time} = \pi 2^{\ell_B+m} + \pi 2^{\ell_A+\ell_B} \text{ and Memory} = 2^{\ell_A} + 2^{\ell_B} .$$

Gradual Matching. Gradual matching is a recursive procedure as detailed by Algo 2. All elements are decomposed into two parts, the first t' groups and the last $(t - t')$, with $t' < t$. For each possible value β of the first t' groups, the sublist $L_B(\beta)$ is built. It consists of all elements in $\mathcal{L}_B$ whose first t' groups take the value β. Now, for each α such that $\mathcal{R}_i(\alpha_i, \beta_i) = 1, 1 \leq i \leq t'$, $L_B(\beta)$ is merged with the sublist $L_A(\alpha)$ which consists of all elements in $\mathcal{L}_A$ whose first t' groups take the value α. Then, we need to merge two smaller lists, of respective sizes $2^{\ell_A - \sum_{i=1}^{t'} m_i}$ and $2^{\ell_B - \sum_{i=1}^{t'} p_i}$.

$$\text{Time} = \left(\prod_{i=1}^{t'} \pi_i \right) 2^{\sum_{i=1}^{t'} m_i+p_i} C_{\mathsf{merge}} \text{ and Memory} = 2^{\ell_A} + 2^{\ell_B} .$$

where C_{merge} is the cost of merging the two remaining sublists.

Algorithm 2 Gradual matching algorithm of $\mathcal{L}_A$ and $\mathcal{L}_B$ with respect to $\mathcal{R}$.

for j from 1 to t **do**
  Build the table T_j such that $T_j[v_j]$ corresponds to all u_j with $\mathcal{R}_j(u_j, v_j) = 1$.
for each $\beta = (\beta_1, \ldots, \beta_{t'})$ in $(\mathbf{F}_2^{\sum_{j=1}^{t'} p_j})$ **do**
  $L_B(\beta) \leftarrow \{b \in \mathcal{L}_B \text{ with } (b_1, \ldots, b_{t'}) = \beta\}$
  $\mathcal{L}_{aux} \leftarrow \emptyset$.
  for each $(\alpha_1, \ldots, \alpha_{t'})$ with $\alpha_j \in T_j[\beta_j]$, $\forall j \leq t'$ **do**
    add $(\alpha_1, \ldots, \alpha_{t'})$ to $\mathcal{L}_{aux}$.
  for each $\alpha = (\alpha_1, \ldots, \alpha_{t'})$ in $\mathcal{L}_{aux}$ **do**
    $L_A(\alpha) \leftarrow \{a \in \mathcal{L}_A \text{ with } (a_1, \ldots, a_{t'}) = \alpha\}$
    Merge $L_A(\alpha)$ with $L_B(\beta)$ with respect to $\mathcal{R}' = \prod_{j=t'+1}^{t} \mathcal{R}_j$.
    Add the solutions to $\mathcal{L}_{sol}$.
Return $\mathcal{L}_{sol}$.

Parallel Matching without memory. We give here the first general description of the memoryless version of parallel matching. The details are provided by Algo 3. This algorithm applies an idea from [19] to the parallel matching algorithm from [33]: instead of building a big auxiliary list as in the original parallel matching, we here build small ones which do not need any additional memory. In parallel matching, the elements in both lists are decomposed into three parts: the first t_1 groups, the next t_2 groups, and the remaining $(t - t_1 - t_2)$ groups. Both lists $\mathcal{L}_A$ and $\mathcal{L}_B$ are sorted in lexicographic order. Then, $\mathcal{L}_A$ can be seen as a collection of sublists $\mathcal{L}_A(\alpha)$, where $\mathcal{L}_A(\alpha)$ is composed of all elements in $\mathcal{L}_A$ whose first t groups equal α. Similarly, $\mathcal{L}_B$ is seen as a collection of $\mathcal{L}_B(\beta)$. The matching algorithm then proceeds as follows. For each possible value α for the first t groups, an auxiliary list $\mathcal{L}_{aux}$ is built, corresponding to the union of all $\mathcal{L}_B(\beta)$ where (α, β) satisfies the first t relations $\mathcal{R}_j$. The list $\mathcal{L}_{aux}$ is sorted by its next t_2 groups. Then, for each element in $\mathcal{L}_A(\alpha)$, we check if a match for its next t_2 groups exists in $\mathcal{L}_{aux}$. For each finding, the remaining $(t - t_1 - t_2)$ groups are tested and only the elements which satisfy the remaining $(t - t_1 - t_2)$ relations are returned.

Algorithm 3 Memoryless parallel matching algorithm of $\mathcal{L}_A$ and $\mathcal{L}_B$ with respect to $\mathcal{R}$.

for j from 1 to t' **do**
  Build the table T_j such that $T_j[v_j]$ corresponds to all u_j with $\mathcal{R}_j(u_j, v_j) = 1$.
for each $\alpha = (a_1, \ldots, a_{t_1})$ appearing in $\mathcal{L}_A$ **do**
  $\mathcal{L}_A(\alpha) \leftarrow \{a \in \mathcal{L}_A : (a_1, \ldots, a_{t_1}) = \alpha\}$.
  // Compute $\mathcal{L}_{aux}$
  $\mathcal{L}_1 \leftarrow \{\beta : \mathcal{R}_j(\alpha_j, \beta_j) = 1, 1 \leq j \leq t_1\}$
  $\mathcal{L}_{aux} \leftarrow \emptyset$
  for each $\beta \in \mathcal{L}_1$ **do**
    $\mathcal{L}_B(\beta) \leftarrow \{b \in \mathcal{L}_B : (b_1, \ldots, b_{t_1}) = \beta\}$.
    add all elements of $\mathcal{L}_B(\beta)$ to $\mathcal{L}_{aux}$.
  Sort $\mathcal{L}_{aux}$ by $\beta' = (b_{1+t_1}, \ldots, b_{t_1+t_2})$.
  // Merge $\mathcal{L}_A(\alpha)$ and $\mathcal{L}_{aux}$ with respect to the next t_2 groups.
  for each a in $L_A(\alpha)$ **do**
    $\mathcal{L}_2 \leftarrow \{\beta' : \mathcal{R}_j(\alpha_j, \beta'_j) = 1, t_1 < j \leq t_1 + t_2\}$
    for each $\beta' \in \mathcal{L}_2$ **do**
      if $\beta' \in \mathcal{L}_{aux}$ **then**
        for each $b \in \mathcal{L}_{aux}$ with $(b_{t_1+1}, \ldots, b_{t_1+t_2}) = \beta'$ **do**
          if $\mathcal{R}_j(a_j, b_j)$ for all $t_2 < j \leq t$ **then**
            Add (a, b) to $\mathcal{L}_{sol}$.

The time and memory complexities can be evaluated as follows. We first evaluate the average sizes of all lists involved in the algorithm. For each α, the average size of $\mathcal{L}_A(\alpha)$ is $2^{\ell_A - \sum_{i=1}^{t_1} m_i}$. Also, we have

$$|\mathcal{L}_1| = \left(\prod_{i=1}^{t_1} \pi_i\right) 2^{\sum_{i=1}^{t_1} p_i} \ , \ |\mathcal{L}_2| = \left(\prod_{i=t_1+1}^{t_1+t_2} \pi_i\right) 2^{\sum_{i=t_1+1}^{t_1+t_2} p_i}$$

and

$$|\mathcal{L}_{aux}| = \left(\prod_{i=1}^{t_1} \pi_i\right) 2^{\ell_B} \ .$$

Finally, the average number N of elements b which match with a on the first $t_1 + t_2$ groups and that should be tested at Line 18 in the algorithm is $\left(\prod_{i=1}^{t_1+t_2} \pi_i\right) 2^{\ell_B}$. Then, the average time complexity of parallel matching can be decomposed as

$$\begin{aligned} \text{Time} &= 2^{\sum_{i=1}^{t_1} m_i} \left[|\mathcal{L}_{aux}| + |\mathcal{L}_A(\alpha)| \left(|\mathcal{L}_2| + N\right)\right] \\ &= \left(\prod_{i=1}^{t_1} \pi_i\right) 2^{\ell_B + \sum_{i=1}^{t_1} m_i} + \left(\prod_{i=t_1+1}^{t_1+t_2} \pi_i\right) 2^{\ell_A + \sum_{i=t_1+1}^{t_1+t_2} p_i} + \left(\prod_{i=1}^{t_1+t_2} \pi_i\right) 2^{\ell_A + \ell_B} \ . \end{aligned}$$

It is worth noticing that the two lists $\mathcal{L}_1$ and $\mathcal{L}_2$ do not need to be stored since their elements are entirely defined by the tables T_j describing the valid transitions for S_j. The average memory required by the algorithm then corresponds to

$$\text{Memory} = |\mathcal{L}_A| + |\mathcal{L}_B| + |\mathcal{L}_{aux}| = 2^{\ell_A} + 2^{\ell_B} + \left(\prod_{i=1}^{t_1} \pi_i\right) 2^{\ell_B} \ .$$

3 Combining Sieve-in-the-Middle and Bicliques

Sieve-in-the-middle, as a generic technique, can be combined with other improvements of MITM attacks, in particular with bicliques [6, 30]. The general purpose of bicliques is to increase the number of rounds attacked by MITM techniques. Here, we briefly describe how bicliques can increase the number of rounds attacked by the previously described sieve-in-the-middle algorithm. This can be done at no computational cost, but requires a higher data complexity. In order to avoid this drawback, we then present an improvement of bicliques which applies when the key length exceeds the block size of the cipher.

3.1 Sieve-in-the-middle and classical bicliques

The combination of both techniques is depicted on Figure 2: the bottom part is covered by bicliques, while the remaining part is covered by a sieve-in-the-middle algorithm. In the following, $H_{K_8} : X \mapsto C$ denotes the function corresponding to the bottom part of the cipher, and K_8 represents the key bits involved in this part. Then, K_8 is partitioned into three disjoint subsets, K_5, K_6 and K_7. The value taken by K_i with $5 \leq i \leq 7$ will be represented by an integer in $\{0, \ldots, 2^{k_i} - 1\}$. A biclique can be built if the active bits in the computation of $H_{K_8}(X)$ when K_6 varies and the active bits in the computation of $H_{K_8}^{-1}(C)$ when K_5 varies are two disjoint sets. In this case, an exhaustive search over K_7 is performed and a biclique

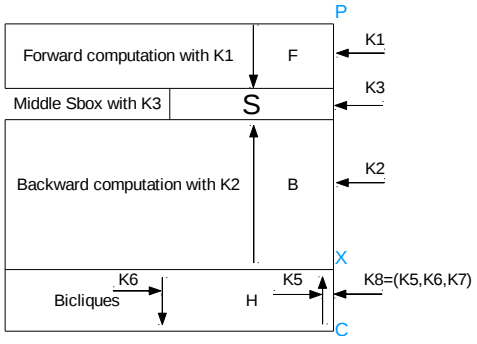


Fig. 2: Generic representation of Sieve-in-the-Middle and bicliques

is built for each value h of K_7 as follows. We start from a given ciphertext C^0 and a chosen key $K_8^0 = (0, 0, h)$ formed by the candidate for K_7 and the zero value for K_5 and K_6. We compute $X_h^0 = H_{0,0,h}^{-1}(C^0)$. Next, we compute backwards from C^0 the intermediate state $X_h^i = H_{i,0,h}^{-1}(C^0)$ for each possible value i for K_5. Similarly, we compute forwards from X_h^0 the ciphertext $C_h^j = H_{0,j,h}(X_h^0)$ for each possible value j of K_6. Since the two differential paths are independent, we deduce that $H_{i,j,h}(X_h^i) = C_h^j$ for all values (i, j) of (K_5, K_6).

Then, the sieve-in-the-middle algorithm can be applied for each K_7 and each value for $(K_1 \cap K_2)$. The list $\mathcal{L}_b$ of all output vectors v is computed backwards from X_h^i for each value i of K_5 and each value of $K_2 \setminus (K_1 \cap K_2)$. The list $\mathcal{L}_f$ of all input vectors u is computed forwards from all plaintexts P_h^j corresponding to C_h^j for each value j of K_6 and each value of $K_1 \setminus (K_1 \cap K_2)$. We then merge those two lists of respective sizes $2^{|K_2 \cup K_5|}$ and $2^{|K_1 \cup K_6|}$.

As in classical MITM with bicliques, the decomposition of K_8 should be such that the bits of K_5 do not belong to K_1, the bits of K_6 do not belong to K_2 and the bits of K_7 should lie in $(K_1 \cap K_2)$. The best strategy here seems to choose (K_5, K_6) such that the bits of K_5 belong to $K_2 \setminus (K_1 \cap K_2)$, and the bits of K_6 belong to $K_1 \setminus (K_1 \cap K_2)$. In this case, we have to add to the time complexity of the attack the cost of the construction of the bicliques, i.e., $2^{k_7}(2^{k_5} + 2^{k_6})c_H$ (very rarely the bottleneck), where c_H is the cost of the partial encryption or decryption corresponding to the rounds covered by the bicliques. The main change is that the data complexity has increased since the attack now requires the knowledge of all plaintext-ciphertext pairs (P_h^j, C_h^j) corresponding to all possible values (j, h) for (K_6, K_7). The data complexity then would correspond to $2^{k_6+k_7}$ pairs of plaintext-chosen ciphertexts, but it is usually smaller since the ciphertexts C_h^j only differ on a few positions.

3.2 Improved bicliques for some scenarios

Now, we describe a generic idea for improving bicliques in certain scenarios and reducing the data complexity to a single plaintext-ciphertext pair. Our improvement usually applies when the total key size of the cipher is larger than the block size. This occurs for instance when whitening keys are used. A detailed and successful application is demonstrated on PRINCE in Section 6. The main idea of our improvement is to gather some parts of the partial exhaustive search over K_7 into different groups such that, within a group, all obtained ciphertexts C^j are equal to C^0.

We consider a biclique repartition of keys consistent with the sieve-in-the-middle part: we choose $K_5 \subset K_2 \setminus (K_1 \cap K_2)$ as previously, and some set $K'_6 \subset K_1$ (this differs from the classical biclique construction where we had $K_6 \subset K_1 \setminus (K_1 \cap K_2)$). Let Δ_6^C be the positions of the bits of C which may be affected by K'_6 when computing forward from X, and let Δ_6^X be the positions of the bits of X which may be affected by Δ_6^C and K'_6 during the backward computation. In classical bicliques, the path generated in the backward direction by the different K_5 must be independent from the path generated in the forward direction by the different K'_6. Here, we also require this first path generated by K_5 to be independent from the backward path generated when the ciphertext bits in positions Δ_6^C vary. For instance, in the example depicted on Figure 3, H follows the Even-Mansour construction, i.e., it is composed of an unkeyed permutation H' and the addition of two whitening keys K_a and K_b. The positions of K_5 and K'_6 are represented in red and blue respectively, and it can be checked that the corresponding paths are independent.

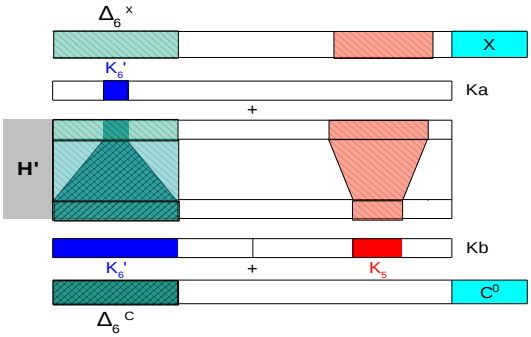
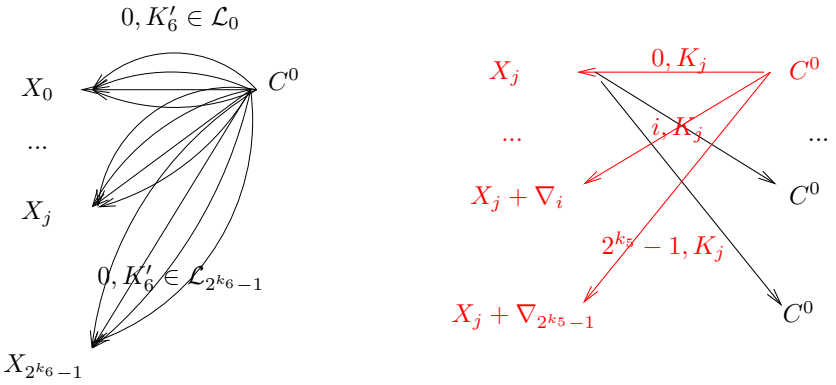


Fig. 3: Example of the improved biclique construction.

In this situation, an improved biclique without any additional data can be built if the size of Δ_6^X is smaller than k'_6. In our context, the algorithm has to be repeated for each value h for $K'_7 = K_8 \setminus (K_5 \cup K'_6)$, but the index h will be omitted in the description. First, we precompute the values obtained from a chosen C^0 when K'_6 takes all possible values. If the number of information bits in Δ_6^X is less than k'_6, all $2^{k'_6}$ transitions can be represented by several lists $\mathcal{L}_j$, each containing the different values of K'_6 which all map C^0 to the same value of the state X, X_j (see Figure 4(a)). For the sake of simplicity, we assume that all these lists have the same size 2^ℓ. In most cases, we have $\ell = k'_6 - |\Delta_6^X|$. For the example depicted on Figure 3, we assume that H' is such that the function obtained by restricting its inputs to the positions in Δ_6^X and its outputs to the positions in Δ_6^C is a permutation. Then, it clearly appears that the number of bits in Δ_6^X is equal to the number of bits of $K'_6 \cap K_b$, and thus strictly smaller than the number of bits of K'_6. More precisely, there are exactly 2^ℓ values of K'_6, with $\ell = |K'_6 \cap K_a|$, which provide the same value of $X = H'^{-1}(C^0 + K_b) + K_a$ when K'_6 varies and all other bits are fixed.

Now, for each of the $2^{k'_6-\ell}$ values of X_j, all transitions from C^0 to X_j through different values of $K'_6 \in \mathcal{L}_j$ can also be seen as the 2^ℓ biclique transitions from X_j to C^0 through some particular values of the key K'_6 (these transitions are represented in black on Figure 4(b)).



(a) Step 1

(b) Step 2: to be repeated for the $2^{k'_6-\ell}$ values of j

Fig. 4: Improved bicliques construction.

Now, the second step consists in building the bicliques in the other direction: from C^0 for each value of X_j. For each of the $2^{k'_6-\ell}$ values of j, we fix the value of K'_6 to a constant value K_j appearing in $\mathcal{L}_j$. This way, the part of X corresponding to Δ_6^X is the same for all the transitions of the bicliques, and this property holds even when K_5 is modified since both corresponding paths are independent. We then consider the 2^{k_5} possible values i for K_5 and compute the corresponding $X = X_j + \nabla_i$ (see Figure 4(b)). We then deduce the $2^{k_5+k'_6}$ transitions $H(X_j + \nabla_i)_{(i,K'_6)} = C^0$ for all $K'_6 \in \mathcal{L}_j$, from $(2^{k'_6} + 2^{k'_6-\ell+k_5})$ computations of the function. Indeed, the first term in the complexity corresponds to the precomputation phase (Step 1), and the second one to the number of lists $\mathcal{L}_j$, $2^{k'_6-\ell}$, multiplied by the cost for building the bicliques in the other direction. The main advantage of this construction is that it can be combined with the sieve-in-the-middle part as previously described, but it now requires a single plaintext-ciphertext pair, the one formed by (P^0, C^0).

Finally, we assume that the bits of K_5 belong to $K_2 \setminus (K_1 \cap K_2)$, the bits of K'_6 belong to K_1 and the bits of K'_7 are the bits from $(K_1 \cup K_2) \setminus (K_5 \cup K'_6)$, the time complexity of the attack is:

$$2^{k'_7}\left(2^{k'_6} + 2^{k'_6-\ell+k_5}\right) c_H + 2^{k_1} c_F + 2^{k_2} c_B + 2^{\kappa} C_{\mathsf{merge}} + \pi 2^k c_E$$

where C_{merge} is the cost of merging the lists of size $2^{k_1-\kappa}$ and $2^{k_2-\kappa}$ with respect to the sieving conditions.

A similar idea can also be used for choosing an appropriate K_5 which delays the propagation of the unknown bits during the forward computation. This will be shown in the case of Prince.

4 Application to PRESENT

We here discuss an application example on the block cipher PRESENT-80, which illustrates our ideas. The number of rounds reached when using the new improvement will be seven, while it can be proved that classical meet-in-the-middle attacks do not apply on more than six rounds. By using bicliques, we can directly extend the attack to 8 rounds with the same computations and a data complexity of 2^6 instead of 2. We can similarly apply our attack to 9 and 10 rounds of PRESENT-128. These results are far from reaching the number of rounds of the best known attacks, but are the first ones with (very) low data complexity. Actually, as pointed out in [11, 12], it is important to analyze the primitives when only few data are available. PRESENT could to some extend be considered as one of the most important lightweight block ciphers, and PRESENT-like functions might be used in further constructions as in [10, 15, 31]. Determining the number of rounds which can be attacked with a single (or only a few) pair of plaintext-ciphertext is then important to better understand its security.

4.1 Brief description of PRESENT

PRESENT is an ultra-lightweight block cipher proposed by Bogdanov *et al.* [7], which has been standardized by ISO in 2011. Its original structure has attracted the attention of the community, and a large number of results on reduced versions have been published [38, 16, 39, 35, 2, 34, 17, 27, 32, 5]. All these attacks need a large number of plaintext-ciphertext pairs, which in most cases reaches the full codebook.

Two versions of PRESENT have been proposed, with an 80-bit key and with a 128-bit key. Besides the key length, both versions only differ in the key schedule. PRESENT operates on 64-bit blocks. For encrypting the plaintext, 31 rounds of the following round-function are applied, followed by a last whitening subkey addition (sk_{32}).

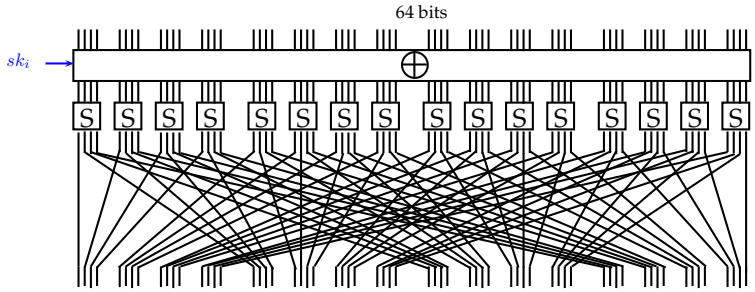


Fig. 5: One round of PRESENT.

The round function consists of 3 transformations, as depicted on Figure 5:

1. The subkey addition: at the beginning of each round i, for $i \in [1, \ldots, 31]$, the corresponding subkey sk_i of 64 bits is xored to the internal state.
2. The non-linear transformation: 16 4×4-bit sboxes S are applied in parallel to the 16 groups of 4 consecutive bits. PRESENT sbox is represented in hexadecimal notation by:

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S[x]	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

3. A bit-wise permutation P, which operates on the 64 bits as follows:

$$P(i) = \begin{cases} 16i \bmod 63 & \text{for } 0 \leq i \leq 62 \\ 63 & \text{for } i = 63 \end{cases}$$

Key schedule for the 80-bit version. Given an 80-bit key $K = k_{79}, \ldots, k_0$, the subkeys, $(sk_1, \ldots, sk_{31}, sk_{32})$ that are xored to the internal state at each round, where sk_{32} is the final whitening key, are computed from the key bits in the following way: For i from 1 to 32

1. $sk_i = k_{79}, \ldots, k_{16}$,
2. $k_{79}, k_{78}, \ldots, k_1, k_0 = k_{18}, k_{17}, \ldots, k_0, k_{79}, \ldots, k_{20}, k_{19}$,
3. $k_{79}k_{78}k_{77}k_{76} = S(k_{79}k_{78}k_{77}k_{76})$,
4. $k_{19}k_{18}k_{17}k_{16}k_{15} = k_{19}k_{18}k_{17}k_{16}k_{15} \oplus$ roundcounter.

4.2 Sieve-in-the-Middle on 7 and 8 rounds of PRESENT-80

The attack exploits the fact that the subkeys do not involve all bits of the key, and also that the values of some bits in the "middle" can be computed without knowing the whole state. Figure 6 represents the consecutive internal states in the MITM attack on 7 rounds of PRESENT-80. One round at the end will then be added with bicliques. Each square represents a nibble (of the key or of the state). Colored bits correspond to known bits, and white bits are unknown. The colored Sboxes in the middle are those involved in the sieve-in-the middle procedure. In the forward computation, all 80 key bits are known except 9, namely bits 3, 4 and 8 to 14. Then, with the notation introduced in Section 2.2, we have $k_1 = 71$. In the backward direction, all key bits but 6 (bits 57, 60, 61, 64, 65 and 68) are known, implying $k_2 = 74$. Clearly, we have $\kappa = |K_1 \cap K_2| = 65$ and $k_3 = k_4 = 0$.

Fig. 6: MITM attack on 7 rounds of PRESENT.

The middle sieving. Figure 7 focuses on the middle part of the cipher. The size of the known input vector u is $m = 27$ and the size of the known output vector v is $p = 15$. The middle sbox S is formed by nine independent 4×4 sboxes, i.e. $t = 9$. For six of these small sboxes, $m = 3$ input bits and $p = 2$ output bits are known. For the other three sboxes, we have $m = 3$ and $p = 1$. As the configuration is the same for each of these two groups of sboxes, the total sieving probability equals $\pi = \pi_1^3 \pi_2^6$, where π_1 and π_2 are the sieving probability of the sbox corresponding to $p = 1$ and $p = 2$ respectively.

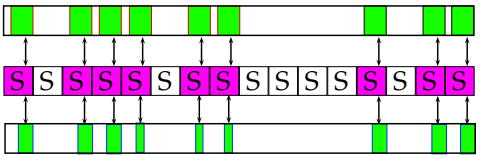


Fig. 7: Middle sieving.

An upper bound on the sieving probability π_2 can be deduced from Proposition 1 since $n = 4$, $m = 3$ and $p = 2$. Then, we have that $\pi_2 \leq 1/2$. From Prop. 4 in Section 8, we can easily deduce that equality holds in this particular case. When $p = 1$, the bound in Proposition 1 is not relevant, but Prop. 4 in Section 8 shows that $\pi_1 = 1 - \frac{1}{8} = 0.875$. Therefore, the total sieving probability is $\pi = 2^{-6.58}$. This means that in the testing phase, we only have to examine $2^{80-6.58} = 2^{73.42}$ potential key candidates and a set of $2^{80-64} = 2^{16}$ possible keys will be recovered. If two pairs of plaintext-ciphertext are known, a proportion of $\pi^2 = 2^{-13.16}$ of the keys needs to be tested, and the attack recovers the whole key, instead of a set of 2^{16} candidates.

The merging step in the attack consists in merging two lists $\mathcal{L}_A$ of size 2^6, containing elements formed by $t = 9$ groups with $m_i = 3$ bits and $\mathcal{L}_B$ of size 2^9, containing elements formed by 6 groups with $p_i = 2$ bits and 3 groups with $p_i = 1$ bits. With these parameters, the best algorithm among the ones presented in Section 2.3 is the memoryless parallel matching applied with the following parameters: $t_1 = 1$, $t_2 = 4$, where the first five groups correspond to the groups with $p = 2$. From the formula given in Section 2.3, we deduce that the time complexity of the merging step is $2^{11} + 2^{10} + 2^{10} = 2^{12}$. For each guess of the 65 bits in $K_1 \cap K_2$, the cost of merging the two lists is then 2^{12}, and we obtain in average $2^{8.42}$ solutions corresponding to the possible key candidates, among the 2^{15} initial ones.

The total time complexity of the attack is then: $2^{65}(2^6 c_F + 2^9 c_B + 2^{12}) + 2^{73.42} c_E \simeq 2^{73.42} c_E$, while the cost of the exhaustive key search is $2^{80} c_E$. The memory complexity is 2^9, and the data complexity is a single pair of known plaintext-ciphertext.

One more round with bicliques. If an 8th round is added, the differential paths in this 8th round generated by the non-common key bits are independent. Then, this last round can be covered by classical bicliques, with no additional time complexity and a data complexity of 2^6.

Experimental results. These results have been partially implemented, and we have verified the sieving part as follows: we have assumed that 48 among the 65 bits in $K_1 \cap K_2$ are

known. Then, we succeeded in recovering the 32 remaining key bits with an average predicted complexity of 2^{29} plus 2^{26} encryptions, proving that our merging phase works as predicted.

The attacks on the 128-bit version are similar to the previous ones, and the highest number of attacked rounds is 9, and 10 with bicliques, but we only gain a factor two on the exhaustive key search.

5 Application to DES

The Data Encryption Standard (DES) appeared in 1977, and was replaced as the official standard of block cipher by the AES in 2000. DES is a 16-round Feistel cipher, which encrypts 64-bit blocks with 56-bit keys. Sixteen balanced Feistel rounds are iteratively applied to the plaintext block, with a F-function composed of a layer of non-injective SBoxes followed by a bit permutation. The F-function, described in Figure 8, accommodates a 32-bit input along with

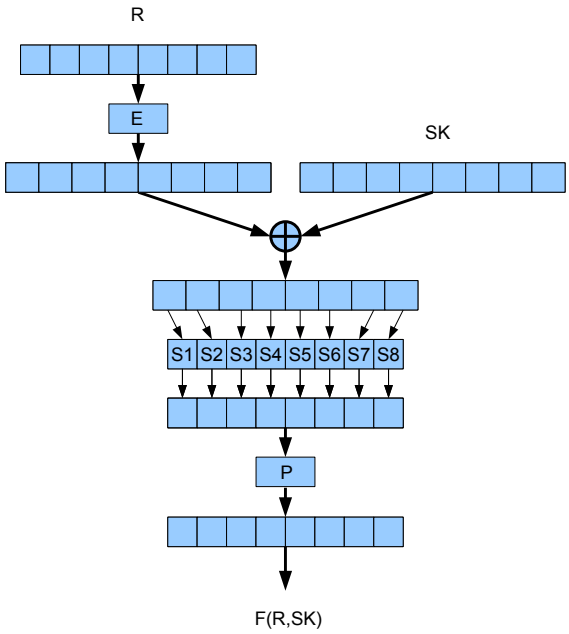
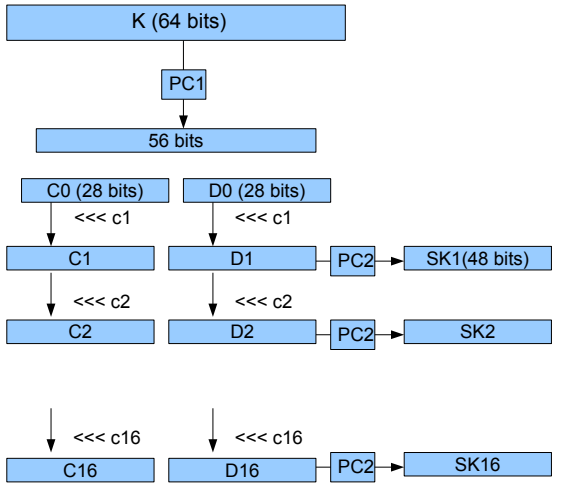


Fig. 8: F-function used in a DES round.

a 48-bit subkey. The input is expanded into 48 bits, and the expanded input is XORed with the subkey. Eight groups of 4 bits are computed by eight 6×4 S-boxes $S1$,$S2$,...,$S8$. Sixteen subkeys are derived from the key with the algorithm described in Figure 9. The effective key length is 56 bits. In the meet-in-the-middle cryptanalysis, we consider the master key as its image under $PC1$, i.e., the initial content (C_0, D_0) of the key register of Figure 9.

5.1 Previous MITM cryptanalyses of reduced DES

MITM cryptanalysis was first proposed for analyzing the extension of double DES. This cryptanalysis proved that the security of double DES would not improve on the security of



c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15	c16
1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Fig. 9: Key scheduling algorithm of DES

DES. Secondly, it was used on the DES itself by Chaum and Evertse [14]. They showed that a meet-in-the-middle key recovery could be applied to six rounds and they gave an upper bound of seven DES rounds to the range of their method. Latter in [21] the efficiency of the meet-in-the-middle was improved for reduced versions of DES to 4, 5, and 6 rounds. Though there exists a meet-in-the-middle attack on 7 rounds of DES that does not start from the first round, ours is the first to reach 7 rounds starting from the beginning (and also 8).

Time complexity	4 rounds	5 rounds	6 rounds	7 rounds	8 rounds
MITM [14]	2^{35}	$2^{45.5}$	$2^{52.9}$	upper bound	
MITM partial matching [21]	2^{20}	$2^{35.5}$	$2^{51.8}$		
MITM partial matching through S-boxes (this section)				2^{53}	2^{53}

Table 1. Complexity of previous meet-in-the-middle from the first round on truncated versions of DES

5.2 Sieve-in-the-middle on 7 and 8 rounds (starting from the first one)

As this analysis is very similar to the one on PRESENT, we briefly explain here its procedure. We first determine the parameters of this attack, which uses a single plaintext-ciphertext pair. In the forward direction we guess all the key bits but the ones at positions $\{19, 26, 36, 55\}$. In the backward direction, the ones missing are $\{2, 6, 9, 21\}$. This means that $k_1 = k_2 = 52$ and $\kappa = |K_1 \cup K_2| = 48$.

With this key decomposition, we can compute 5 input bits of the sbox $S7$ in the fifth round for the forward direction, and the four bits of output of the same sbox in the backward

direction. In this case, we have $S = S7$, and so $t = 1$. Let us recall here that the DES sboxes are not bijective, but 6-bit to-4 bit sboxes. Therefore, the knowledge of all 4 output bits does not determine the input. Instead, for $m_1 = 5$ known input bits and $p_1 = 4$ known output bits, the sieving probability is $\pi = 2^{-(n'-1)} = 2^{-3}$. Indeed, Proposition 4 implies that any $(n-1, n')$-sieve has maximal probability $2^{-(n'-1)}$ if and only if there is no pair of elements at Hamming distance 1 which have both the same value under the sbox. This is the case of all DES Sboxes, which have been designed such that any two inputs which correspond to the same output always differ on at least 2 positions.

The size of both lists L_b and L_f is 2^4. We can perform an instant matching (with $L_A = L_b$ and $L_B = L_f$) for finding the $2^{4+4-3} = 2^5$ solutions with a complexity of $2^{4+1} = 2^5$. The memory complexity of this phase is determined by the size of both lists, as we do not need to store the transition tables and we can compute the output for the possible missing input bit on the fly. The total time complexity of the attack will be:

$$2^{48}(2^4 c_F + 2^4 c_B + 2^5 + 2^5 c_E) \approx 2^{53} c_E,$$

so we have won 3 bits on the exhaustive search ($2^{56} c_E$).

This attack has been implemented considering 24 of the κ bits of $(K_1 \cap K_2)$ as known and recovering the remaining 32, and we have obtained the expected complexities, verifying our theoretical approaches.

We can add one additional round with bicliques: with the previously described configuration, the paths generated by K_6 and K_5 are not independent. For that, we need to transform one of the bits of $K_1 \setminus (K_1 \cap K_2)$ into a a common bit, i.e. also included in K_2. In this case the attack with the biclique covering the 8-th round can be applied in a similar way as before, with time complexity:

$$2^{49}(2^3 c_f + 2^4 c_b + 2^4 + 2^4 c_E) \approx 2^{53} c_E,$$

and a data complexity of 2^4 plaintext-ciphertext pairs.

6 Application to PRINCE

PRINCE is a lightweight block cipher designed by Borghoff *et al.* [9]. Though being very recent, it has already waked the interest of many cryptanalysts [37, 26, 1]. The best known attacks so far on the proposed cipher, including the security analysis performed by the authors, reach 6 rounds. In particular, MITM with bicliques (without guessing the whole key) is said to reach at most 6 rounds (out of 12). In [26], a reduction of the security by one bit is presented, and in [1] an accelerated exhaustive search using bicliques is presented. Here, we describe how to build sieve-in-the-middle attacks on 8 rounds with data complexity 1 (or 2 if we want to the whole key instead of a set of candidates). In addition to the new sieve-in-the-middle technique, we use the improved method for constructing bicliques presented in Section 3.2.

6.1 Brief description of PRINCE

PRINCE operates on 64-bit blocks and uses a 128-bit key composed of two 64-bit elements, K_a and K_b. Its structure is depicted on Figure 10. PRINCE is based on the so-called FX-construction: two whitening keys $W_{in} = (K_a + K_b)$ and $W_{out} = (K'_a + K_b)$ are xored respectively to the input and to the output of a 12-round core cipher parametrized

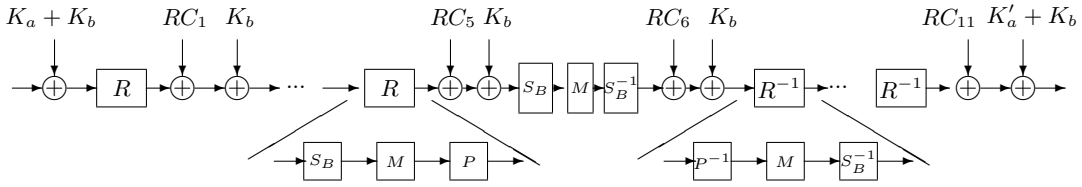


Fig. 10: Structure of PRINCE.

by K_b only. The value of K'_a involved in the post-whitening key is derived from K_a by $K'_a = (K_a \ggg 1) \oplus (K_a \gg 63)$.

The round function is composed of:

- a non-linear layer S_B corresponding to 16 parallel applications of a 4×4 sbox σ.
- a linear layer $P \circ M$, where M is the parallel application of 4 involutive mixcolumns operations on 16 bits each (defined either by $\hat{M}^{(0)}$ or by $\hat{M}^{(1)}$). This transformation is then followed by a permutation P of the 16 nibbles defined by

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

$\longrightarrow$

0	5	10	15	4	9	14	3	8	13	2	7	12	1	6	11

- the addition of a round constant RC_i and of the subkey K_b.

The first 5 rounds in PRINCE correspond to the previously described round permutation R, while the last 5 rounds are defined by the inverse permutation R^{-1}. The two middle rounds correspond to the successive applications of S_B, M and S_B^{-1}.

6.2 Sieve-in-the-middle and improved bicliques on 8 rounds

Sieve-in-the-middle on six rounds. We first describe the sieve-in-the-middle part of the attack, which covers Rounds 1 to 6 (see Figure 11). The internal state X after Round 6 is supposed to be known, as well as the plaintext. The sieving step is done with respect to a function S which covers Round 3 and the S_B level of Round 4. This middle function S can then be decomposed as four 16×16 superboxes: the colored nibbles in the middle of Figure 11 represent the nibbles belonging to the same superbox.

The 128 keybits in PRINCE are then decomposed as depicted on Figure 12:

- K_1, i.e. the keybits known in the forward direction, are represented in white and in blue in K_b and the first whitening key W_{in}. They correspond to all bits K_b and W_{in} except the 11 leftmost bits of the third 16-bit group in K_b.
- K_2, i.e. the keybits known in the backward direction, are represented in white and in red in K_b and W_{in}. They correspond to all bits of K_b and W_{in} except the leftmost nibble of K_b and the 16 bits at positions 0 and 49 to 63 in W_{in}.

It follows that the intersection $(K_1 \cap K_2)$ consists of $\kappa = 97$ information bits of (K_a, K_b): the 49 white bits in K_b and the 48 white bits in W_{in}.

The algorithm is described on Figure 11, where each nibble which contains 'K' is known in the backward computation, each nibble which contains 'k' is known in the forward computation and '1' means that there is a known bit in the nibble. The right part of the figure represents the key. We will exploit the fact that, for each 16×16 mixcolumns operation, there

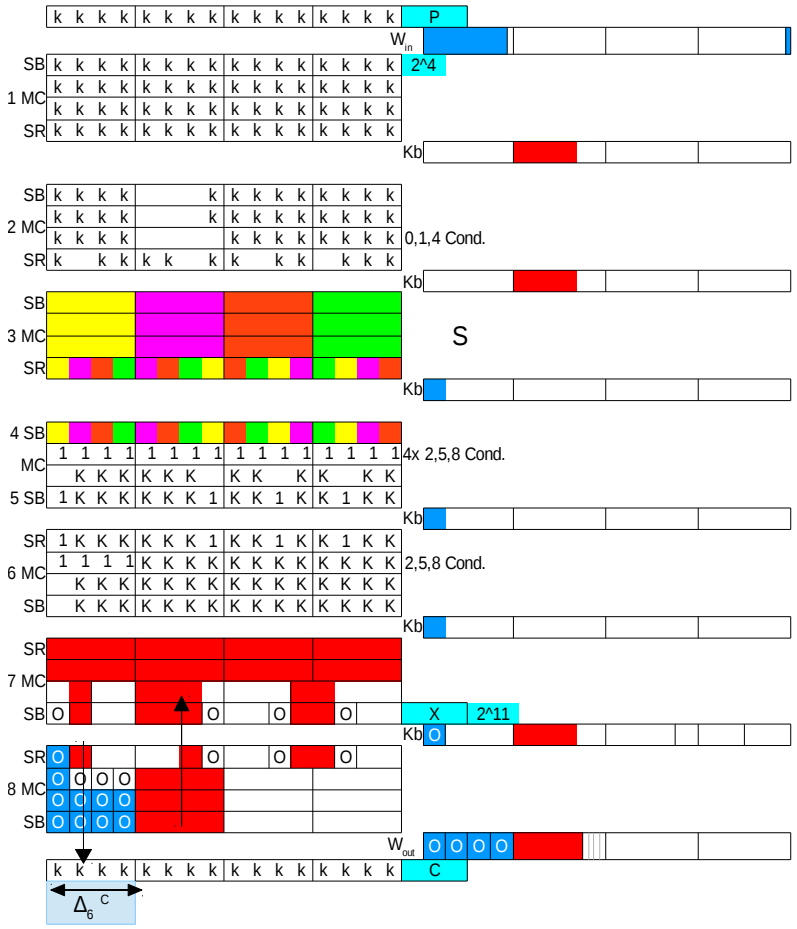


Fig. 11: Sieve-in-the-middle attack on 8 rounds of PRINCE with data complexity of 1.

exist 4 output bits (one per nibble), as well as 8 information bits of the output, which do not depend on a given input nibble. Each of these 8 information bits corresponds to the sum of two output bits. Indeed, the 16×16 transformation $\hat{M}_0$ is defined by

$$\begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ b_0 & b_1 & b_2 & b_3 \\ c_0 & c_1 & c_2 & c_3 \\ d_0 & d_1 & d_2 & d_3 \end{pmatrix} \longmapsto \begin{pmatrix} b_0+c_0+d_0 & a_1+c_1+d_1 & a_2+b_2+d_2 & a_3+b_3+c_3 \\ a_0+b_0+c_0 & b_1+c_1+d_1 & a_2+c_2+d_2 & a_3+b_3+d_3 \\ a_0+b_0+d_0 & a_1+b_1+c_1 & b_2+c_2+d_2 & a_3+c_3+d_3 \\ a_0+c_0+d_0 & a_1+b_1+d_1 & a_2+b_2+c_2 & b_3+c_3+d_3 \end{pmatrix},$$

where a, b, c and d represent the four input nibbles. Then, it can be checked for instance that the four output bits a'_0, b'_1, c'_2 and d'_3 do not depend on nibble a, as well as the eight information bits $b'_0 + c'_0$, $b'_0 + d'_0$, $a'_1 + c'_1$, $a'_1 + d'_1$, $a'_2 + b'_2$, $a'_2 + d'_2$, $a'_3 + b'_3$ and $a'_3 + c'_3$. The same situation holds for every input nibble, as also for the other mixcolumns transformation $\hat{M}_1$.

In the backward computation, from State X and K_2, we can compute 3 nibbles of each input of the mixcolumns operations at Round 5. Then, we deduce one bit in each nibble of

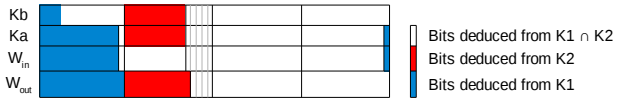


Fig. 12: Decomposition of the key in the attack on 8 rounds of PRINCE. $W_{in} = K_a \oplus K_b$ and $W_{out} = (K_a \ggg 1) \oplus (K_a \gg 63) \oplus K_b$.

the output of the middle function S, as well as 32 information bits which involve the outputs of two different superboxes. When considering $s < 4$ superboxes together, the number of information bits known is reduced to 8 if $s = 2$, and to 20 if $s = 3$.

In the forward computation, from the plaintext P and K_1, we compute three input nibbles of each superbox. From the mixcolumns operation in Round 2 whose input is partially known, we can also have 4 additional information bits on the input of the middle function S. When considering $s < 4$ superboxes together, the number of information bits known is reduced to 0 if $s = 2$ and to 1 if $s = 3$.

Then, we need to merge the two lists $\mathcal{L}_f$ and $\mathcal{L}_b$ of respective sizes 2^4 and 2^{11}. Since $m = 4 \times 12 + 4 = 52$ input bits and $p = 4 \times 4 + 32 = 48$ output bits are known, the total sieving probability π is at most $2^{64-(52+48)} = 2^{-36}$. In the following, the tables T_j providing all transitions for the four superboxes S_j are supposed to be known[3].

We are going to first apply the instant matching on the first two blocks (orange and green), i.e., we apply Algorithm 1 described on Page 6 with parameters $n_1 = n_2 = 16$ and $m_1 = m_2 = 12$ and $p_1 + p_2 = 8 + 8 = 16$. The sieving probability of these two superboxes together is then $\pi_{1,2} = 2^{32-(24+16)} = 2^{-8}$. We consider $\mathcal{L}_A = \mathcal{L}_b$ and $\mathcal{L}_B = \mathcal{L}_f$. From the corresponding formula in Section 2.3, we get that the time complexity of this step is $2^{-8}2^{4+16} + 2^{-8}2^{15} \approx 2^{12}$. With this complexity we have found $2^{15}\pi_{1,2} = 2^7$ input-output pairs of S which are valid for the first two superboxes. We can now check whether each of these pairs is also valid for the two remaining superboxes. Now, the sieving probability for the remaining part is at most $2^{-36} \times 2^{+8} = 2^{-28}$ as the total sieving probability is at most 2^{-36}.

Therefore, at the end of the merging step, for each guess of the $\kappa = 113$ bits of $(K_1 \cap K_2)$, we have a probability of $2^{7-28} = 2^{-21}$ of finding a correct configuration for the 15 remaining bits of (K_1, K_2). This means that the testing step will consider $2^{113-21} = 2^{92}$ keys, and it will recover 2^{64} possible candidates for the whole key. If two plaintext-ciphertext pairs are available, the testing step will consider $2^{92-36} = 2^{56}$ keys instead of 2^{92}, leading to performing a test over 2^{56} candidates for recovering the correct key.

Improved bicliques part. Our attack combines the previous sieve-in-the-middle algorithm with bicliques built as described in Section 3.2, without increasing the data complexity. We define K'_6 as the five nibbles corresponding to the union of the leftmost nibble of K_b and the four leftmost nibbles of the whitening key $W_{out} = (K'_a + K_b)$. Then, Δ_6^C is represented on Figure 11 by the four 'O' symbols in the line before C. Also, Δ_6^X then corresponds to the 'O'

[3]The orange and green superboxes that involve common key bits only can be computed on the fly and will be used first for the instant matching. For each pair we obtain, the whole key is already known, so we can repeat the on-the-fly procedure.

symbols in X. Then, $|\Delta_6^C| = 16$ and $|\Delta_6^X| = 16$. The remaining 'O' show the path from Δ_6^C to Δ_6^X. All 2^{20} transitions obtained when K_6' varies correspond, for each one of the 2^{16} possible values of j, to 2^4 biclique transitions from X_j to C. Then, K_5 is defined as the 11 leftmost bits of the third 16-bit group of K_b, implying that K_5 is equal to $K_2 \setminus (K_1 \cap K_2)$. The path generated in the backward direction, represented in red, is then independent from the blue path generated by K_6', and also from the path with 'O' symbols from Δ_6^C to Δ_6^X.

The complete algorithm then consists in performing an exhaustive search over the $\kappa = 97$ common bits corresponding to the white bits of K_b and W_{in} in Figure 12. The previously described bicliques determine 2^{16} states X_j, and 2^4 transitions from each X_j to C. Then, for each X_j, we examine the corresponding 2^4 values of K_6'. For those K_6', we compute forwards from the plaintext P the list of all 2^4 vectors u. It is worth noticing that even if the red bits of K_a and K_b are unknown in the forward direction, their sum is known (see Fig. 12). Similarly, the list $\mathcal{L}_b$ of all vectors v is computed backwards from the 2^{11} X^i and their associated value i for K_5. From the formula given in Section 3.2, we deduce that, for one plaintext-ciphertext pair, the time complexity is

$$\text{Time} = 2^{97}\left(2^{20} + 2^{16+11}\right) c_H + 2^{117} c_F + 2^{113} c_B + 2^{97} \times 2^{12} + 2^{-36} \times 2^{128} c_E \simeq 2^{124} c_H \ .$$

We have then gained more than four bits over the exhaustive search ($2^{128} c_E$). The memory complexity is of 2^{20}, corresponding to the precomputed table in the construction of the improved bicliques, since the transition tables for the superboxes can be computed on the fly.

7 Application to AES Biclique

In the analysis on the full-round AES-128 proposed by Bogdanov *et al.* in [6], though the whole key needs to be guessed, the authors count the number of sboxes which need to be recomputed for each guess, among all sboxes involved in one encryption. Then, despite a loop of size 2^{128}, a speed-up factor of 0.27 is obtained, leading to a complexity of $2^{126.14}$ encryptions.

Our sieve-in-the-middle technique can be applied and it may improve the complexity on some platforms. We can consider that the middle function S in our algorithm is defined by a 32×32 superbox. By precomputing and storing the possible transitions for the superbox in a table of size 2^{32}, we do not need to recompute the five sboxes included in S each time. Instead, we determine by a table-lookup whether a transition from one known input byte to 4 known output bytes is possible. If the attack is performed on a platform on which a look-up in a table of size 2^{32} is faster than five evaluations of the sbox, this variant is slightly faster than the original attack. Because of the branch number of MixColumns, the corresponding sieving probability is 2^{-8}, implying that the sieving performed is the same as in the original analysis (where a collision on 8 bits is considered). Our technique seems similar to the concept of MITM through linear relations, introduced in [25, 28]. However, in our case, we do not check linear relations, but possible transitions for a nonlinear function, since the function which provides the sieving is a superbox and involves sboxes. For this reason, this technique allows us to decrease the number of sbox evaluations: compared to an exhaustive search for the key, the proportion of sbox evaluations is now 0.203.

It is worth noticing that the precomputation of the tables has a negligible cost, as they only need to be computed once for each guess of the 2^{32} values of key bits involved in S. Also, we are able to choose an S involving a part of the key which does not need to be recomputed.

Though the improvement is very tiny, we believe that our technique can also be useful for future biclique attacks which aim at an accelerated exhaustive search for the key.

8 Sieving Probability and Related Properties of the Sbox

8.1 General properties

In this section, we focus on the general problem of theoretically estimating the sieving property provided by two subsets $I \subset \{1, \ldots, n\}$ and $J \subset \{1, \ldots, n'\}$, with respective sizes m and p, for a given function S from $\mathbf{F}_2^n$ into $\mathbf{F}_2^{n'}$. In particular, we provide some results on the minimal value of $(m+p)$ for which a sieve exists. In the following, S_J denotes the function from $\mathbf{F}_2^n$ into $\mathbf{F}_2^p$ corresponding to the p coordinates of S defined by J. Also, for any affine subspace W, $S_{|W}$ denotes the restriction of S to W, i.e., the function defined on W by $S_{|W}(x) = S(x)$. Obviously, $S_{|W}$ can be identified with a function of $\dim W$ input variables.

For a given input set I, V denotes the linear subspace $V = \{x \in \mathbf{F}_2^n \ : x_i = 0, i \in I\}$. Then, the sieving probability of (I, J) can be expressed in terms of the sizes of all $\mathsf{Range}(S_J)_{|u+V}$ when u varies.

Proposition 2. *Let A_k, $1 \leq k \leq 2^{\min(p,n-m)}$, be the number of cosets $u+V$ such that $\#\mathsf{Range}(S_J)_{|u+V} = k$ when u varies in $\mathbf{F}_2^m$. Then, the sieving probability of (I, J) is equal to*

$$\pi_{I,J} = 2^{-p} + 2^{-(p+m)} \sum_{k=2}^{2^{\min(p,n-m)}} (k-1)A_k \ .$$

Proof. The value of $\pi_{I,J}$ is deduced from the number of valid pairs (u, v), which equals

$$\sum_{k=1}^{\min(2^p, 2^{n-m})} kA_k = 2^m + \sum_{k=2}^{\min(2^p, 2^{n-m})} (k-1)A_k$$

where the last equality comes from the fact that the sum of all A_k equals 2^m. □

Then, we deduce the following corollary by using that $\pi_{I,J} = 2^{-p}$ if and only if $A_1 = 2^m$. This means that S_J is constant on all cosets of V.

Corollary 1. *The sieving probability of (I, J) satisfies $\pi_{I,J} \geq 2^{-p}$, with equality if and only if S_J does not depend on its inputs at positions in $\{1, \ldots, n\} \setminus I$.*

Link with the branch number of S. We associate to S the (nonlinear) code $\mathcal{C}_S$ of length $(n+n')$ and of size 2^n defined by $\mathcal{C}_S = \{(x, S(x)), x \in \mathbf{F}_2^n\}$. The minimum distance of $\mathcal{C}_S$ is the lowest value of $wt(x+y) + wt(S(x)+S(y))$ for distinct x, y. It corresponds to the *branch number* of S. Obviously, when $m+p > n$, the sieving probability of any (I, J) of size (m, p) is at most $2^{n-(m+p)}$ (see Proposition 1). Now, the following proposition shows that this upper bound is tight when $(m+p)$ exceeds some bound depending on the branch number of S.

Proposition 3. *Let $I \subset \{1, \ldots, n\}$ and $J \subset \{1, \ldots, n'\}$ be two subsets with respective sizes m and p with $m+p \geq n$. Then, the following three statements are equivalent:*

(i) $\pi_{I,J} < 2^{n-(p+m)}$

(ii) *there exist two distinct elements x and y in $\mathbf{F}_2^n$ such that*

$$\mathsf{Supp}(x+y) \subseteq \overline{I} \text{ and } \mathsf{Supp}(S(x)+S(y)) \subseteq \overline{J}$$

(iii) *there exist some input difference of the form $a = (0_I, \alpha)$ and some output difference $b = (0_J, \beta)$ such that the entry of index (a, b) in the difference table of S is non-zero.*

Most notably, all (m,p)-sieves have probability $2^{n-(m+p)}$ if and only if $m+p > n+n'-d_{\min}$ where $d_{\min}$ is the branch number of S (i.e., the minimal distance of $\mathcal{C}_S$).

Proof. The last two statements are clearly equivalent. Then, we will prove the equivalence between the first two. For any $u \in \mathbf{F}_2^m$, the restriction of S_J to $u+V$ can take at most 2^{n-m} values. Then, $\pi_{I,J} = 2^{n-(m+p)}$ if and only if, for any $u \in \mathbf{F}_2^m$, all values of $S_J(x)$ are distinct when x varies in $u+V$. This equivalently means that there is no pair of inputs x_1 and x_2 which coincide on I (i.e., which have the same u) such that $S(x_1)$ and $S(x_2)$ coincide on all positions in J. Thus, $\pi_{I,J} < 2^{n-(m+p)}$ if and only if there exists x_1 and x_2 such that $\mathsf{Supp}(x_1+x_2) \subset \overline{I}$ and $\mathsf{Supp}(S(x_1)+S(x_2)) \subset \overline{J}$. Then, the Hamming distance between $(x_1, S(x_1))$ and $(x_2, S(x_2))$ equals $n - +n' - (m+p)$, implying that such a pair of elements exists if and only if $n+n'-(m+p) < d_{\min}$. □

For instance, the branch number of the 4×4 PRESENT sbox is equal to 3. It follows that any (m,p) sieve with $m+p \geq 6$ has probability $2^{n-(m+p)}$.

Lower bound on the minimal value of $(m+p)$. Even if the code $\mathcal{C}_S$ is a nonlinear code, its dual distance can be defined as follows (if $\mathcal{C}_S$ is linear, this definition coincides with the minimum distance of the dual code $\mathcal{C}_S^{\perp}$).

Definition 2. *Let $\mathcal{C}$ be a code of length N and size M over $\mathbf{F}_q$ and $A = (A_0, \ldots, A_N)$ be its distance distribution, i.e., $A_i = \frac{1}{M}\#\{(x,y) \in \mathcal{C} \times \mathcal{C} \ : d_H(x,y) = i\}$.*

Let $A' = (A'_0, \ldots, A'_N)$ be the image of A under the MacWilliams transform, $A'(X,Y) = A(X+(q-1)Y, X-Y)$ where $A(X,Y) = \sum_{i=0}^{N} A_i X^{N-i} Y^i$ and $A'(X,Y) = \sum_{i=0}^{N} A'_i X^{N-i} Y^i$. The dual distance *of $\mathcal{C}$ is the smallest nonzero index i such that $A'_i \neq 0$.*

The dual distance of $\mathcal{C}_S$ is a lower bound on the lowest $(m+p)$ for which an (m,p)-sieve exists. Indeed, we can use the following theorem due to Delsarte.

Theorem 1. *[18] Let $\mathcal{C}$ be a code of length N and size M over $\mathbf{F}_q$. Then, the words of $\mathcal{C}$ restricted to any t positions take all the q^t possible values exactly M/q^t times if and only if $t < d^{\perp}$ where $d^{\perp}$ is the dual distance of $\mathcal{C}$.*

Then, we derive the following result.

Theorem 2. *Let $d^{\perp}$ be the dual distance of the code $\mathcal{C}_S$. Then, for any (m,p) such that $m+p < d^{\perp}$, there is no (m,p)-sieve for S. Moreover, there exists no (m,p)-sieve for S with $m+p \leq n$ if and only if $\mathcal{C}_S$ is an MDS code, which cannot occur if S is defined over $\mathbf{F}_2$.*

Proof. The first part of the theorem is a direct consequence of Delsarte's theorem (Theorem 1).

The second part comes from the fact that, for $m+p = n$, (I,J) is not an (m,p)-sieve if and only if $(x_i, i \in I; S_j(x), j \in J)$ takes all possible values in $\mathbf{F}_q^n$ exactly once. From Delsarte's theorem, this situation occurs for all (I,J) with $m+p = n$ if and only if the dual distance

of $\mathcal{C}$ is greater than or equal to $(n+1)$. But, as noted in [18, Page 426], $d^{\perp} = n+1$ implies that the minimum distance of $\mathcal{C}$ is also maximal, i.e., $d_{\min} = n'+1$ (or equivalently that $\mathcal{C}$ is MDS). In this case, we deduce from Prop 3 that all (m,p) sieves with $m+p \geq n$ have efficiency $2^{n-(m+p)}$. □

In some scenarios, S is defined over a larger alphabet, and I and J may be defined as two sets of byte (or nibble) positions. Then, the previous theorem proves that, if the corresponding code $\mathcal{C}_S$ is an MDS code, there is no (m,p)-sieve for $m+p \leq n$, and we deduce also from Proposition 3 that all (m,p)-sieve with $m+p > n$ have probability $2^{n-(m+p)}$.

8.2 Sieving probability for some particular values of (m,p)

$(m,1)$-sieves and nonlinearity. When $p=1$, a pair $(I,\{j\})$ of size $(m,1)$ is a sieve if and only if S_j is constant on some coset $u+V$. Therefore, if $(I,\{j\})$ is a sieve, then S_j is $(n-m)$-normal, i.e. constant on an affine subspace of dimension $(n-m)$. In particular, it can be approximated by an affine function with a probability at least $\frac{1}{2}(1+2^{-m})$ [20]. It follows that, if S provides the best resistance to linear cryptanalysis for even n, then it has no sieve $(I,\{j\})$ with $|I| < \frac{n}{2}-1$. As an example, the AES Sbox does not have any $(2,1)$-sieve.

$(n-1,p)$-sieves. When $m = n-1$, the sieving probability can be easily determined by the difference table of S.

Proposition 4. *Let $I = \{1,\ldots,n\} \setminus \{\ell\}$ and let $J \subset \{1,\ldots,n'\}$ with $|J| = p$. Then,*

$$\pi_{I,J} = 2^{-(p-1)} - 2^{-(p+n)} \sum_{\beta \in \mathbf{F}_2^{n'-p}} \delta(e_\ell, (0_J, \beta)) \ ,$$

where $\delta(a,b) = |\{x \in \mathbf{F}_2^n : S(x+a)+S(x) = b\}|$ is the element of index (a,b) in the difference table of S, and e_ℓ is the input vector with a 1 at position ℓ. Thus, $(I,\{j\})$ is a sieve except if S_j is linear in x_ℓ.

Proof. From Prop. 2, we have

$$\pi_{I,J} = 2^{-p} + 2^{-(p+n-1)} A_2$$

where A_2 is the number of u such that $S_J(x)$ takes two values when x varies in $\{u, u+e_\ell\}$. We can compute A_2 from the difference table of S:

$$\begin{aligned} A_2 &= \frac{1}{2}\#\{x \in \mathbf{F}_2^n : S(x+e_\ell)+S(x) = (\alpha_J, \beta), \text{ with } \alpha_J \neq 0\} \\ &= \frac{1}{2}(2^n - \#\{x \in \mathbf{F}_2^n : S(x+e_\ell)+S(x) = (0_J, \beta)\} \\ &= \frac{1}{2}(2^n - \sum_{\beta \in \mathbf{F}_2^{n'-p}} \delta_S(e_\ell, (0_J, \beta)) \ . \end{aligned}$$

It follows that $(I,\{j\})$ is not a sieve if and only if the function $x \mapsto S_j(x+e_\ell)+S_j(x)$ is the all-one function. This equivalently means that S_j is linear in x_ℓ. □

For instance, since the branch number of the PRESENT sbox is 3, Prop. 3 implies that (m,p)-sieves with $m+p=5$ exist for this sbox. Indeed, by considering its difference table, we get that all (I,J) of size $(3,2)$ correspond to a sieving probability $\pi_{I,J} \in \{\frac{1}{2}, \frac{1}{2}-\frac{1}{32}, \frac{1}{2}-\frac{1}{16}\}$. It is worth noticing that the sieve used in the attack presented in Section 4, $I = \{0,1,2\}$ and $J = \{0,1\}$ has probability $\frac{1}{2}$. We also derive from Prop. 4 the exact sieving probability involved in the attack on the DES presented in Section 5.

9 Conclusions

The main contributions of this paper are a generic improvement of MITM attacks, the sieve-in-the-middle technique, which allows to attack more rounds, and an improved biclique construction which avoids the need of additional data. These two methods have been applied to PRESENT, DES, AES and PRINCE. Moreover, some general results on the sieving probability of an sbox are given, which allow to theoretically estimate the complexity of the attack.

A future possible line of work is to investigate some possible combinations with other existing MITM improvements: with the guess of intermediate state bits [21], or with the all-subkeys approach [24]. A promising direction would be to try to make a first selection within each of the two lists before the merging step, by keeping only the input values (resp. output values) which have the lowest probability of corresponding to a valid transition. This introduces some non-detection probability, since some correct candidates would be discarded, but the sieving would be improved. Such an approach does not seem easy, but it would surely be a big step forward for further improving MITM attacks.

Acknowledgements

We thank Dmitry Khovratovich for his valuable comments, and all CryptoExperts members for their kindness and hospitality.

References

1. Farzaneh Abed, Eik List, and Stefan Lucks. On the Security of the Core of PRINCE Against Biclique and Differential Cryptanalysis. Cryptology ePrint Archive, Report 2012/712, 2012. `http://eprint.iacr.org/2012/712`.
2. Martin R. Albrecht and Carlos Cid. Algebraic Techniques in Differential Cryptanalysis. In *FSE 2009*, volume 5665 of *Lecture Notes in Computer Science*, pages 193–208. Springer, 2009.
3. Kazumaro Aoki and Yu Sasaki. Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In *Selected Areas in Cryptography - SAC 2008*, volume 5381 of *Lecture Notes in Computer Science*, pages 103–119. Springer, 2008.
4. Kazumaro Aoki and Yu Sasaki. Meet-in-the-Middle Preimage Attacks Against Reduced SHA-0 and SHA-1. In *CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 70–89. Springer, 2009.
5. Céline Blondeau and Benoît Gérard. Multiple Differential Cryptanalysis: Theory and Practice. In *FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 35–54. Springer, 2011.
6. Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique Cryptanalysis of the Full AES. In *ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371. Springer, 2011.
7. Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.
8. Andrey Bogdanov and Christian Rechberger. A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN. In *Selected Areas in Cryptography - SAC 2010*, volume 6544 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2010.
9. Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif B. Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications. In *ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.
10. Julia Borghoff, Lars R. Knudsen, Gregor Leander, and Søren S. Thomsen. Cryptanalysis of PRESENT-Like Ciphers with Secret S-Boxes. In *FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 270–289. Springer, 2011.
11. Charles Bouillaguet, Patrick Derbez, Orr Dunkelman, Pierre-Alain Fouque, Nathan Keller, and Vincent Rijmen. Low-data complexity attacks on AES. *IEEE Transactions on Information Theory*, 58(11):7002–7017, 2012.

12. Charles Bouillaguet, Patrick Derbez, and Pierre-Alain Fouque. Automatic Search of Attacks on Round-Reduced AES and Applications. In *CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 169–187. Springer, 2011.
13. Billy Bob Brumley, Risto M. Hakala, Kaisa Nyberg, and Sampo Sovio. Consecutive S-box Lookups: A Timing Attack on SNOW 3G. In *Information and Communications Security - ICICS 2010*, volume 6476 of *Lecture Notes in Computer Science*. Springer, 2010.
14. David Chaum and Jan-Hendrik Evertse. Crytanalysis of DES with a Reduced Number of Rounds: Sequences of Linear Factors in Block Ciphers. In *CRYPTO'85*, volume 218 of *Lecture Notes in Computer Science*, pages 192–211. Springer, 1985.
15. Huiju Cheng, Howard M. Heys, and Cheng Wang. PUFFIN: A Novel Compact Block Cipher Targeted to Embedded Digital Systems. In *DSD*, pages 383–390. IEEE, 2008.
16. Joo Yeon Cho. Linear Cryptanalysis of Reduced-Round PRESENT. In *CT-RSA 2010*, volume 5985 of *Lecture Notes in Computer Science*, pages 302–317. Springer, 2010.
17. Baudoin Collard and François-Xavier Standaert. A Statistical Saturation Attack against the Block Cipher PRESENT. In *CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*. Springer, 2009.
18. Philippe Delsarte. Four fundamental parameters of a code and their combinatorial signifiance. *Information and Control*, 23(5):407–438, December 1973.
19. Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Efficient Dissection of Composite Problems, with Applications to Cryptanalysis, Knapsacks, and Combinatorial Search Problems. In *CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 719–740. Springer, 2012.
20. Hans Dobbertin. Construction of Bent Functions and Balanced Boolean Functions with High Nonlinearity. In *FSE'94*, volume 1008 of *Lecture Notes in Computer Science*, pages 61–74. Springer, 1994.
21. Orr Dunkelman, Gautham Sekar, and Bart Preneel. Improved Meet-in-the-Middle Attacks on Reduced-Round DES. In *INDOCRYPT 2007*, volume 4859 of *Lecture Notes in Computer Science*, pages 86–100. Springer, 2007.
22. Jian Guo, San Ling, Christian Rechberger, and Huaxiong Wang. Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2. In *ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 56–75. Springer, 2010.
23. Takanori Isobe. A Single-Key Attack on the Full GOST Block Cipher. In *FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2011.
24. Takanori Isobe and Kyoji Shibutani. All Subkeys Recovery Attack on Block Ciphers: Extending Meet-in-the-Middle Approach. In *Selected Areas in Cryptography - SAC 2012*, volume 7707 of *Lecture Notes in Computer Science*, pages 202–221. Springer, 2012.
25. Takanori Isobe and Kyoji Shibutani. Security Analysis of the Lightweight Block Ciphers XTEA, LED and Piccolo. In *Australasian Conference on Information Security and Privacy - ACISP 2012*, volume 7372 of *Lecture Notes in Computer Science*, pages 71–86. Springer, 2012.
26. Jérémy Jean, Ivica Nikolic, Thomas Peyrin, Lei Wang, and Shuang Wu. Security Analysis of PRINCE. In *FSE 2013*, Lecture Notes in Computer Science. Springer, 2013. To appear.
27. Stéphanie Kerckhof, Baudoin Collard, and François-Xavier Standaert. FPGA Implementation of a Statistical Saturation Attack against PRESENT. In *AFRICACRYPT 2011*, volume 6737 of *Lecture Notes in Computer Science*, pages 100–116. Springer, 2011.
28. Dmitry Khovratovich, Gaëtan Leurent, and Christian Rechberger. Narrow-Bicliques: Cryptanalysis of Full IDEA. In *EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 392–410. Springer, 2012.
29. Dmitry Khovratovich, María Naya-Plasencia, Andrea Röck, and Martin Schläffer. Cryptanalysis of *Luffa* v2 Components. In *Selected Areas in Cryptography - SAC 2012*, volume 6544 of *Lecture Notes in Computer Science*, pages 388–409. Springer, 2010.
30. Dmitry Khovratovich, Christian Rechberger, and Alexandra Savelieva. Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family. In *FSE 2012*, volume 7549 of *Lecture Notes in Computer Science*, pages 244–263. Springer, 2012.
31. Lars R. Knudsen, Gregor Leander, Axel Poschmann, and Matthew J. B. Robshaw. PRINTcipher: A Block Cipher for IC-Printing. In *CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 16–32. Springer, 2010.
32. Jorge Nakahara, Pouyan Sepehrdad, Bingsheng Zhang, and Meiqin Wang. Linear (Hull) and Algebraic Cryptanalysis of the Block Cipher PRESENT. In *Cryptology and Network Security - CANS 2009*, volume 5888 of *Lecture Notes in Computer Science*. Springer, 2009.
33. María Naya-Plasencia. How to Improve Rebound Attacks. In *CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 188–205. Springer, 2011.

34. Kenji Ohkuma. Weak Keys of Reduced-Round PRESENT for Linear Cryptanalysis. In *Selected Areas in Cryptography - SAC 2009*, volume 5867 of *Lecture Notes in Computer Science*, pages 249–265. Springer, 2009.
35. Onur Özen, Kerem Varici, Cihangir Tezcan, and Çelebi Kocair. Lightweight Block Ciphers Revisited: Cryptanalysis of Reduced Round PRESENT and HIGHT. In *Australasian Conference on Information Security and Privacy - ACISP 2009*, volume 5594 of *Lecture Notes in Computer Science*, pages 90–107. Springer, 2009.
36. Yu Sasaki. Meet-in-the-Middle Preimage Attacks on AES Hashing Modes and an Application to Whirlpool. *IEICE Transactions*, 96-A(1):121–130, 2013.
37. Hadi Soleimany, Céline Blondeau, Xiaoli Yu, Wenling Wu, Kaisa Nyberg, Huiling Zhang, Lei Zhang, and Yanfeng Wang. Reflection Cryptanalysis of PRINCE-like Ciphers. In *FSE 2013*, Lecture Notes in Computer Science. Springer, 2013. To appear.
38. Meiqin Wang. Differential Cryptanalysis of Reduced-Round PRESENT. In *AFRICACRYPT 2008*, volume 5023 of *Lecture Notes in Computer Science*, pages 40–49. Springer, 2008.
39. Muhammad Reza Z'aba, Håvard Raddum, Matthew Henricksen, and Ed Dawson. Bit-Pattern Based Integral Attack. In *FSE 2008*, volume 5086 of *Lecture Notes in Computer Science*, pages 363–381. Springer, 2008.

J. Cryptol. (2014) 27: 772–798
DOI: 10.1007/s00145-013-9156-7

Journal of
CRYPTOLOGY

Improved Cryptanalysis of AES-like Permutations*

Jérémy Jean
École Normale Supérieure, Paris, France
Jeremy.Jean@ens.fr

María Naya-Plasencia
INRIA Paris-Rocquencourt, Paris, France
Maria.Naya_Plasencia@inria.fr

Thomas Peyrin
Nanyang Technological University, Singapore, Singapore
thomas.peyrin@gmail.com

Communicated by Meier.

Received 27 July 2012
Online publication 17 July 2013

Abstract. AES-based functions have attracted of a lot of analysis in the recent years, mainly due to the SHA-3 hash function competition. In particular, the rebound attack allowed to break several proposals and many improvements/variants of this method have been published. Yet, it remained an open question whether it was possible to reach one more round with this type of technique compared to the state-of-the-art. In this article, we close this open problem by providing a further improvement over the original rebound attack and its variants, that allows the attacker to control one more round in the middle of a differential path for an AES-like permutation. Our algorithm is based on lists merging as defined in (Naya-Plasencia in Advances in Cryptology: CRYPTO 2011, pp. 188–205, 2011) and we generalized the concept to non-full active truncated differential paths (Sasaki et al. in Lecture Notes in Computer Science, pp. 38–55, 2010).

As an illustration, we applied our method to the internal permutations used in Grøstl, one of the five finalist hash functions of the SHA-3 competition. When entering this final phase, the designers tweaked the function so as to thwart attacks from Peyrin (Peyrin in Lecture Notes in Computer Science, pp. 370–392, 2010) that exploited relations between the internal permutations. Until our results, no analysis was published on Grøstl and the best results reached 8 and 7 rounds for the 256-bit and 512-bit versions, respectively. By applying our algorithm, we present new internal permutation distinguishers on 9 and 10 rounds, respectively.

Key words. Cryptanalysis, Hash function, AES, SHA-3, Grøstl, Rebound attack.

* It was solicited as one of the best papers from FSE 2012.

1. Introduction

Hash functions are one of the most important primitives in symmetric-key cryptography. They are simply functions that, given an input of variable length, produce an output of a fixed size. They are needed in several scenarios, like integrity check, authentication, digital signatures, so we want them to verify some security properties, for instance: preimage resistance, collision resistance (i.e., for an n-bit hash function, finding two distinct inputs mapping to the same output should require at least $2^{n/2}$ computations), second preimage resistance, and so on.

Since 2005, several new attacks on hash functions have appeared. In particular, the hash standards MD5 and SHA-1 were cryptanalyzed by Wang et al. [26,27]. Due to the resemblance of the standard SHA-2 with SHA-1, the confidence in the former was also somewhat undermined. This is why the American National Institute of Standards and Technology (NIST) decided to launch in 2008 a competition in order to find a new hash standard, SHA-3. This competition received 64 hash function submissions and accepted 51 to enter the first round. Three years and two rounds later, only 5 hash functions remained in the final phase of the competition.

Among the candidates, many functions were AES-based (they reuse some AES components or the general AES design strategy), like the SHA-3 finalist Grøstl [6]. This design trend is at the origin of the introduction of the rebound attack [18], a new cryptanalysis technique that has been widely deployed, improved and applied to a large number of SHA-3 candidates, hash functions and other types of AES-based constructions (such as block ciphers in the known/chosen-key model). It has become one of the most important tools used to analyze the security margin of many SHA-3 candidates as well as their building blocks.

The rebound attack was proposed as a method to derive a pair of internal states that verifies some truncated differential path with lower complexity than a generic attack. It was formed by two steps: a first one, *the controlled part (or inbound)*, where solutions for two rounds of an unkeyed AES-like permutation were found with negligible complexity, and a second one, *uncontrolled part (or outbound)*, where the solutions found during the inbound phase were used to verify probabilistically the remaining differential transitions. Assuming an AES-like internal state composed of a $t \times t$ matrix of c-bit cells, the rebound attack was then extended to three rounds by the start-from-the-middle [17] and the *SuperSBox* variants [7,14] for a negligible average complexity per found pair, but with a higher minimal complexity of $2^{t \cdot c}$ computations. Since most rebound-based attacks actually required many such pairs, this was not much of a constraint. In parallel, other improvements on the truncated differential paths utilized [25] or on methods to merge lists [21] were proposed.

In this article, we describe a method based on lists merging in order to control truncated differences over four rounds of an unkeyed AES-like permutation [12] with complexity $2^{t \cdot c \cdot x}$ computations, where x is a parameter depending on the differential path considered. While the cost per pair found in the controlled part is much increased, solving four rounds directly allows to handle much better truncated differential paths for the uncontrolled part. Note that whether it was possible or not to reach four rounds remained an open problem among the research community. We also generalize the global

Table 1. Best attacks on targets where our analysis is applicable. By best analysis, we mean the ones on the highest number of rounds.

Target	Subtarget	Rounds	Time	Memory	Ideal	Reference
Grøstl-256	Permutation	8 (dist.)	2^{112}	2^{64}	2^{384}	[7]
		8 (dist.)	2^{48}	2^{8}	2^{96}	[25]
		9 (dist.)	2^{368}	2^{64}	2^{384}	Sect. 3
		10 (zero-sum)	2^{509}	–	2^{512}	[3]
Grøstl-512	Permutation	7 (dist.)	2^{152}	2^{56}	2^{512}	[25]
		8 (dist.)	2^{280}	2^{64}	2^{448}	Appendix A
		9 (dist.)	2^{328}	2^{64}	2^{384}	Appendix A
		10 (dist.)	2^{392}	2^{64}	2^{448}	Appendix A
PHOTON-224/32/32	Permutation	8 (dist.)	2^{8}	2^{4}	2^{10}	[8]
		9 (dist.)	2^{184}	2^{32}	2^{192}	Appendix B

reasoning by considering as well non-fully-active truncated differential paths [25] during both the controlled and uncontrolled phases, eventually obtaining the best known results for many attack scenarios of an AES-like permutation.

As an application, we concentrated our efforts on the Grøstl internal permutation. Rebound-like attacks on this function have already been applied and improved in several occasions [7,17,19,21,24], Grøstl being one of the most studied SHA-3 candidates. When entering the final round, a tweak of the function was proposed, which prevents the application of the attacks from [24]. We denote Grøstl-0 the original submission [5] of the algorithm and Grøstl its tweaked version [6]. Apart from the rebound results, the other main analysis communicated on Grøstl is a higher order property on 10 rounds of its internal permutation [3] with a complexity of 2^{509} computations. In Table 1, we give a summary of the best known results on both the 256- and 512-bit tweaked versions of Grøstl, including the ones that we present in this article.

Namely, we provide the best known rebound distinguishers on 9 rounds of the internal permutation and we show how to make some nontrivial observations on the corresponding compression function, providing the best known analysis of the Grøstl compression function exploiting the properties of the internal permutations. For Grøstl-512, we considerably increase the current largest number of analyzed rounds, from 7 to 10. Additionally, we provide in Appendix the direct application of our new techniques to the AES-based hash function PHOTON [8].

These results do not threaten the security of Grøstl, but we believe they will play an important role in better understanding its security, and AES-based functions in general. In particular, we believe that our work will help determining the bounds and limits of rebound-like attacks in this type of constructions.

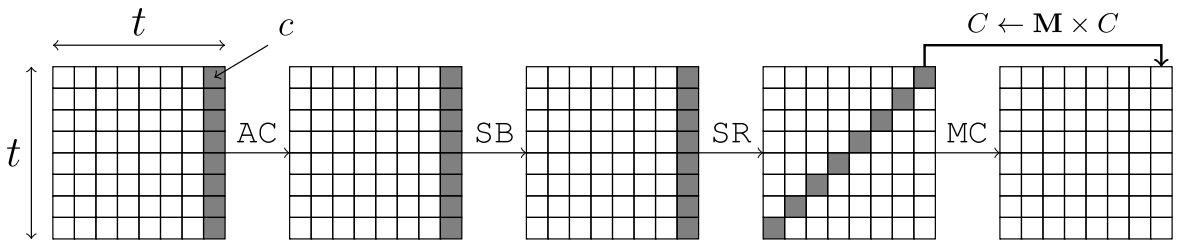


Fig. 1. One round of the AES-like permutation instantiated with $t = 8$.

2. Generalities

In this section, we start by describing a generic view of an AES-like permutation to capture various cryptographic primitives such as AES [4], Grøstl [5], ECHO [2], Whirlpool [1], LED [9], or PHOTON [8].

2.1. *Description of* AES*-like Permutations*

We define an AES-like permutation as a permutation that applies N_r rounds of a round function to update an internal state viewed as a square matrix of t rows and t columns, where each of the t^2 cells has a size of c bits. As we will show later, our techniques can also be adapted when the matrix is not square (as it is the case for Grøstl-512), but we focus on square matrices for ease of description.

The round function (Fig. 1) starts by xoring a round-dependent constant to the state in the *AddRoundConstant* operation (AC). Then, it applies a substitution layer *SubBytes* (SB) which relies on a $c \times c$ nonlinear bijective SBox. Finally, the round function performs a linear layer, composed of the *ShiftRows* transformation (SR), that moves each cell belonging to the x-th row by x positions to the left in its own row, and the *MixCells* transformation (MC), that linearly mixes all the columns C of the matrix separately by multiplying each one with a matrix $\mathbf{M}$ implementing a Maximum Distance Separable (MDS) code: $C \leftarrow \mathbf{M} \times C$.

Note that this description encompasses permutations that really follow the AES design strategy, but very similar designs (for example with a slightly modified *ShiftRows* function or with a *MixCells* layer not implemented with an MDS matrix) are likely to be attacked by our techniques as well. In the case of AES-like block ciphers analyzed in the known/chosen-key model, the subkeys generated by the key schedule are incorporated into the known constant addition layer *AddRoundConstant*. We note that all the rounds considered in this article are *full* rounds: they all have the *MixCells* transformation, even the last one as opposed to the full version of the AES.

2.2. *Description of* Grøstl

The hash function Grøstl-0 has been submitted to the SHA-3 competition under two different versions: Grøstl-0-256, which outputs a 256-bit digest and Grøstl-0-512 with a 512-bit one. For the final round of the competition, the candidate has been tweaked to Grøstl, with corresponding versions Grøstl-256 and Grøstl-512.

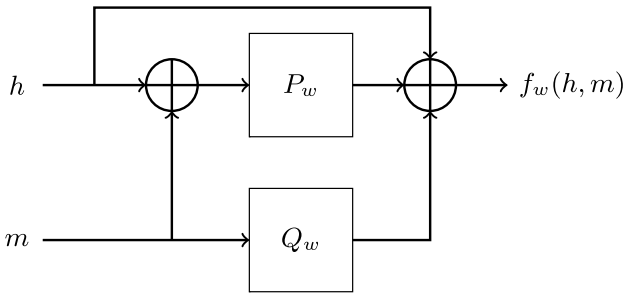


Fig. 2. The compression function of Grøstl using the permutations P_w and Q_w, with $w \in \{256, 512\}$.

The Grøstl hash function handles messages[1] by dividing them into blocks after some padding and uses them to update iteratively an internal state (initialized to a predefined IV) with a compression function. This function is itself built upon two different permutations, namely P and Q. Each of those two permutations are built upon the well-understood wide-trail strategy of the AES. As an AES-like Substitution-Permutation Network, Grøstl enjoys a strong diffusion in each of the two permutations and by its wide-pipe design, the size of the internal state is ensured to be at least twice as large as the final digest.

The compression function f_{256} of Grøstl-256 uses two 256-bit permutations, P_{256} and Q_{256}, which are similar to the two 512-bit permutations, P_{512} and Q_{512}, used in the compression function f_{512} of Grøstl-512. More precisely, for a chaining value h and a message block m, the compression function (Fig. 2) produces the output ($\oplus$ denotes the XOR operation):

$$f_{256}(h, m) = P_{256}(h \oplus m) \oplus Q_{256}(m) \oplus h, \quad \text{or:} \tag{1}$$

$$f_{512}(h, m) = P_{512}(h \oplus m) \oplus Q_{512}(m) \oplus h. \tag{2}$$

The internal states are viewed as matrices of bytes of size 8×8 for the 256-bit version and 8×16 for the 512-bit one. The permutations strictly follow the design of the AES and are constructed as N_r iterations of the composition of four basic transformations:

$$R \stackrel{\text{def}}{=} \mathit{MixCells} \circ \mathit{ShiftBytes} \circ \mathit{SubBytes} \circ \mathit{AddRoundConstant}. \tag{3}$$

All the linear operations are performed in the same finite field $GF(2^8)$ as in the AES, defined via the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$ over $GF(2)$. The *AddRoundConstant* (AC) operation adds a predefined round-dependent constant, which significantly differs between P and Q to prevent the internal differential attack [24] that takes advantage of the similarities between P and Q. The *SubBytes* (SB) layer is the nonlinear layer of the round function R and applies the same SBox as in the AES to all the cells of the internal state. The *ShiftBytes* (Sh) transformation shifts cells in row i by $\tau_P[i]$ positions to the left for permutation P and $\tau_Q[i]$ positions for permutation Q. We note that τ also differs from P to Q to emphasize the asymmetry between the two permutations. Finally, *MixCells* (MC) is implemented

[1] Messages are of maximal bit-length $2n \cdot (2^{64} - 1) - 64 - 1$ for Grøstl-n, with $n \in \{256, 512\}$.

in Grøstl by the **MixBytes** (Mb) operation that applies a circulant MDS constant matrix M independently to all the columns of the state. In Grøstl-256, $N_r = 10$, $\tau_P = [0, 1, 2, 3, 4, 5, 6, 7]$ and $\tau_Q = [1, 3, 5, 7, 0, 2, 4, 6]$, whereas for Grøstl-512, $N_r = 14$ and $\tau_P = [0, 1, 2, 3, 4, 5, 6, 11]$ and $\tau_Q = [1, 3, 5, 11, 0, 2, 4, 6]$.

Once all the message blocks of the padded input message have been processed by the compression function, a final output transformation is applied to the last chaining value h to produce the final n-bit hash value $h' = \text{trunc}_n(P(h) \oplus h)$, where trunc_n only keeps the last n bits.

2.3. *Distinguishers*

In this article, we describe algorithms that find input pairs (X, X') for an AES-like permutation P, such that the input difference $\Delta_{IN} = X \oplus X'$ belongs to a subset of size IN and the output difference $\Delta_{OUT} = P(X) \oplus P(X')$ belongs to a subset of size OUT. The best known generic algorithm (this problem is different than the one studied in [14] where linear subspaces are considered) in order to solve this problem, known as limited-birthday problem, has been given in [7] and later a very close lower bound has been proven in [22]. For a randomly chosen n-bit permutation π, the generic algorithm can find such a pair with complexity

$$\max\left\{\min\left\{\sqrt{2^n/IN}, \sqrt{2^n/OUT}\right\}, 2^n/(IN \cdot OUT)\right\}. \tag{4}$$

If one is able to describe an algorithm requiring less computation power, then we consider that a distinguisher exists on the permutation π.

In the case of Grøstl, it is also interesting to look at not only the internal permutations P and Q, but also the compression function f itself. For that matter, we will generate compression function input values (h, m) such that $\Delta_{IN} = m \oplus h$ belongs to a subset of size IN, and such that $\Delta_{IN} \oplus \Delta_{OUT} = f(h, m) \oplus f(m, h) \oplus h \oplus m$ belongs to a subset of size OUT. Then, one can remark that:

$$\begin{aligned} &f(h, m) \oplus f(m, h) \\ &\quad = P_{256}(h \oplus m) \oplus Q_{256}(m) \oplus P_{256}(m \oplus h) \oplus Q_{256}(h) \oplus h \oplus m, \end{aligned} \tag{5}$$

$$f(h, m) \oplus f(m, h) = Q_{256}(m) \oplus Q_{256}(h) \oplus h \oplus m. \tag{6}$$

Hence, it follows that:

$$f(h, m) \oplus f(m, h) \oplus h \oplus m = Q_{256}(m) \oplus Q_{256}(h). \tag{7}$$

Since the permutation Q is supposed to have no structural flaw, the best known generic algorithm requires $\max\{\min\{\sqrt{2^n/IN}, \sqrt{2^n/OUT}\}, 2^n/(IN \cdot OUT)\}$ operations (the situation is exactly the same as the permutation distinguisher with permutation Q) to find a pair (h, m) of inputs such that $h \oplus m \in IN$ and $f(h, m) \oplus f(m, h) \oplus h \oplus m \in OUT$. Note that both IN and OUT are specific to our attacks.

We emphasize that even if trivial distinguishers are already known for the Grøstl compression function (for example fixed-points), no distinguisher is known for the internal permutations. Moreover, our observations on the compression function use the differential properties of the internal permutations.

2.4. *Truncated Differential Characteristics*

In the following, we will consider truncated differential characteristics, originally introduced by Knudsen [13] for block cipher analysis. With this technique, already proven to be efficient for AES-based hash functions cryptanalysis [10,11,16,18,23], the attacker only checks if there is a difference in a cell (active cell, denoted by a black square in the figures) or not (inactive cell, denoted by an empty square in the figures) without caring about the actual value of the difference.

In this model, all *AddRoundConstant* and *SubBytes* layers can be ignored since they have no impact on truncated differences. *ShiftBytes* will only move the difference positions and the diffusion will come from the *MixCells* layers. More precisely, we denote $x \rightarrow y$ a non-null truncated differential transition mapping x active cells to y active cells in a column through a *MixCells* (or $MixCells^{-1}$) layer, and the MDS property ensures $x + y \geq t + 1$. Its differential probability is determined by the number $(t - y)$ of inactive cells on the output: $2^{-c(t-y)}$ if the MDS property is verified, 0 otherwise.

3. Distinguishers for AES-like Permutations

In this section, we describe a distinguisher for 9 rounds of an AES-like permutation with certain parameters t and c. For the sake of clarity, we first describe the attack for a truncated differential characteristic with three fully active states in the middle, but we will generalize our method in the next section by introducing a characteristic parameterized by variables controlling the number of active cells in some particular states.

Let us remark that before our work, the best known such distinguishers on this type of constructions could only reach 8 rounds, being an open problem whether reaching more rounds would be possible.

3.1. *A First Truncated Differential Characteristic*

The truncated differential characteristic we use has the sequence of active cells

$$t \xrightarrow{R_1} 1 \xrightarrow{R_2} t \xrightarrow{R_3} t^2 \xrightarrow{R_4} t^2 \xrightarrow{R_5} t^2 \xrightarrow{R_6} t \xrightarrow{R_7} 1 \xrightarrow{R_8} t \xrightarrow{R_9} t^2, \tag{8}$$

where the sizes of the input and output difference subsets are both $IN = OUT = 2^{ct}$, since there are t active c-bit cells in the input of the truncated characteristic, and the t^2 active cells in the output are linearly generated from only t active cells. The actual truncated characteristic instantiated with $t = 8$ is described in Fig. 3.

Note that we have three fully active internal states in the middle of the differential characteristic, and this kind of path is impossible to solve with previous rebound or *SuperSBox* techniques since the number of controlled rounds would be too small and the cost for the uncontrolled part would be extremely high.

3.2. *Finding a Conforming Pair*

The method to find a pair of inputs conforming to this truncated differential characteristic is similar to the rebound technique: we first find many solutions for the middle

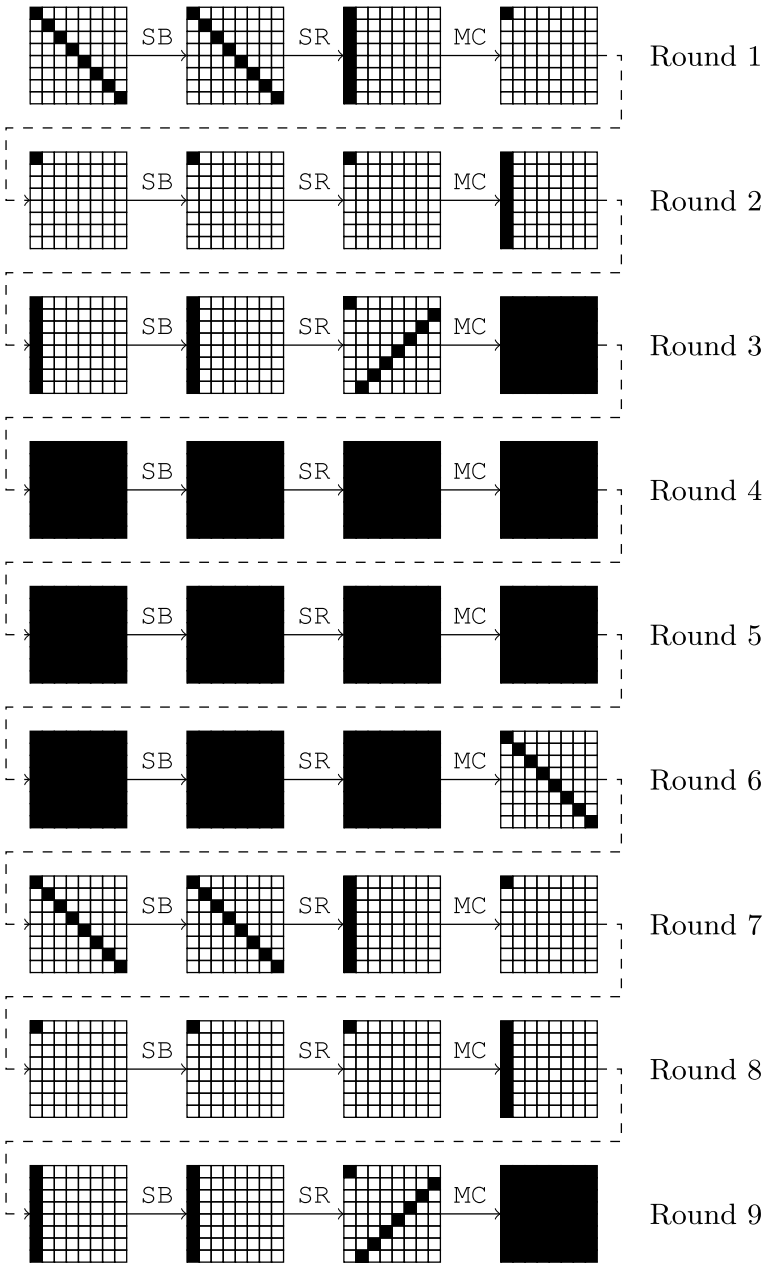


Fig. 3. The 9-round truncated differential characteristic used to distinguish an AES-like permutation from an ideal permutation.

rounds (beginning of round 3 to the end of round 6) and then we filter them out during the outward probabilistic transitions through the *MixCells* layers (round 2 and round 7). Since in our case we have two *MixCells* transitions $t \rightarrow 1$ (see Fig. 3), the outbound phase has a success probability of $2^{-2c(t-1)}$ and is straightforward to handle once we found enough solutions for the inbound phase.

In order to find solutions for the middle rounds (see Fig. 4), we propose an algorithm inspired by the ones in [20,21]. As in [7,14], instead of dealing with the classical t^2 parallel c-bit *SubBytes* SBox applications, one can consider t parallel tc-bit SBoxes (named *SuperSBoxes*) each composed of two SBox layers surrounding one *MixCells* and one *AddRoundConstant* function. Indeed, the *ShiftBytes* can be taken out from the *SuperSBoxes* since it commutes with *SubBytes*. The part of the internal state modified

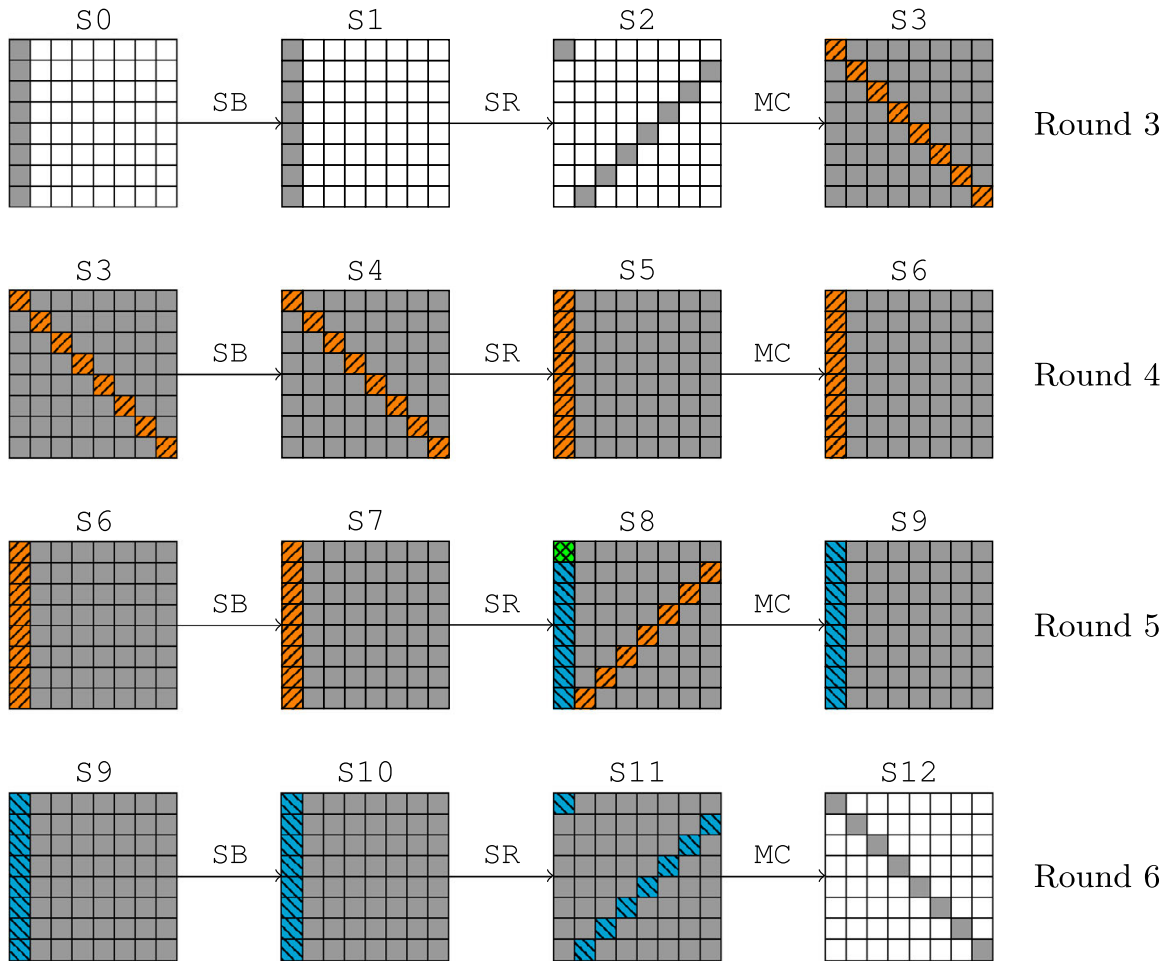
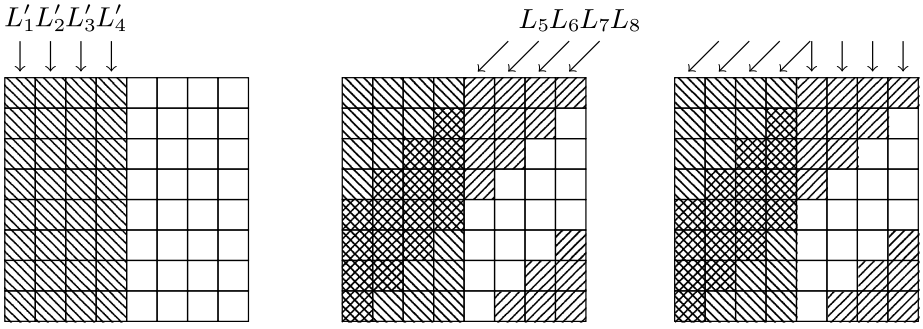


Fig. 4. Inbound phase for the 9-round distinguisher attack on an AES-like permutation instantiated with $t = 8$. The four rounds represented are the rounds 3 to 6 from the whole truncated differential characteristic. A *gray cell* indicates an active cell; *hatched* and *colored cells* emphasize one *SuperSBox* set: there are seven similar others for each one of the two hatched senses. (Color figure online)

by one *SuperSBox* is a *SuperSBox* set. The total state is formed by t such sets, and their particularity is that their transformation through the *SuperSBox* can be computed independently.

We start by choosing the input difference δ_{IN} after the first *SubBytes* layer in state S1 and the output difference δ_{OUT} after the last *MixCells* layer in state S12. Both δ_{IN} and δ_{OUT} are exact differences, not truncated ones, but they are chosen so that they are compliant with the truncated characteristic in S0 and S12. Since we have t active cells in S1 and S12, there are as many as 2^{2ct} different ways of choosing $(\delta_{IN}, \delta_{OUT})$. Note that differences in S1 can be directly propagated to S3 since *MixCells* is linear. We continue by computing the t forward *SuperSBox* sets independently by considering the 2^{ct} possible input values for each of them in state S3. This generates t independent lists, each of size 2^{ct} and composed by paired values in S3 (that can be used to compute the corresponding paired values in S8). Doing the same for the t backward *SuperSBox* sets from state S12, we again get t independent lists of 2^{ct} elements each, and we can compute for each element of each list the pair of values of the *SuperSBox* set in state S8, where the t forward and the t backward lists overlap. In the sequel, we denote L_i the ith forward *SuperSBox* list and L'_i the ith backward one, for $1 \leq i \leq t$.



(a) Step 1. (b) Steps 2 and 3. (c) Step 4.

Fig. 5. In the case where $t = 8$, the figure shows the steps to merge the $2 \times t$ lists. *Gray cells* denote cells fully constrained by a choice of elements in $L'_1, \ldots, L'_{t/2}$ during the first step.

In terms of freedom degrees in state S8, we want to merge $2t$ lists of 2^{ct} elements each for a merging condition on $2 \times ct^2$ bits, where we use the definition of list merging from [21] (ct^2 for values and ct^2 for differences) since the merging state is fully active: we then expect $2^{2t \times ct} 2^{-2ct^2} = 1$ solution as a result of the merging process on average.

In the following, we describe a method to find this solution and compute its complexity afterwards (see Fig. 5). In comparison to the algorithm suggested in [12] where the case $t = 8$ is treated, we generalize the concept to any t, even odd ones where the direct extension of [12] is not applicable. To detail this algorithm, we use a temporary parameter $t' \in [1, t]$ such that the time complexity will be written in terms of t'. In the end, we give the best choice for t' such that the time complexity is minimal for any t.

Step 1. We start by considering every possible combination of elements in each of the t' first lists $L'_1, \ldots, L'_{t'}$ There are $2^{c \cdot t \cdot t'}$ possibilities.

Step 2. Each choice in Step 1 fixes the first t' columns of the internal state (both values and differences) and thus forces $2c$ constraints on t' cells in each of the t lists L_i, $1 \leq i \leq t$. For each list L_i, we then expect on average $2^{ct} 2^{-2ct'} = 2^{c(t-2t')}$ elements to match this constraint for each choice in Step 1, and these elements can be found with one operation by sorting the lists L_i beforehand.[2]

Step 3. We continue by considering every possible combination of elements in each of the $t - t'$ last lists $L_{t-t'+1}, \ldots, L_t$. Depending on the value of t', we may have different scenarios at this point: if $t - 2t' \geq 0$, then the time complexity is multiplied by $2^{c(t-2t')(t-t')}$, which is the number of expected elements in the lists. Otherwise, the probability of success decreases from 1 to $2^{c(t-2t')(t-t')}$, as the constraints imposed by the previous step are too strong and elements in those lists would exist only with probability smaller than 1.

Step 4. We now need to ensure that the t' first lists $L_1, \ldots, L_{t'}$ and the $t - t'$ last lists $L'_{t-t'+1}, \ldots, L'_t$ contain a candidate fulfilling the constraints deduced in the previous steps. In the L'_i lists, we already determined $2c(t - t')$ bits so that there are $2^{ct-2c(t-t')}$ elements remaining in each of those. Again, we can check if these elements exist with one operation by sorting the lists beforehand. Finally, the value and difference of all

[2] We consider lists for the sake of clarity, but we can reach the constant-time access of elements using hash tables. Otherwise, it would introduce a logarithmic factor.

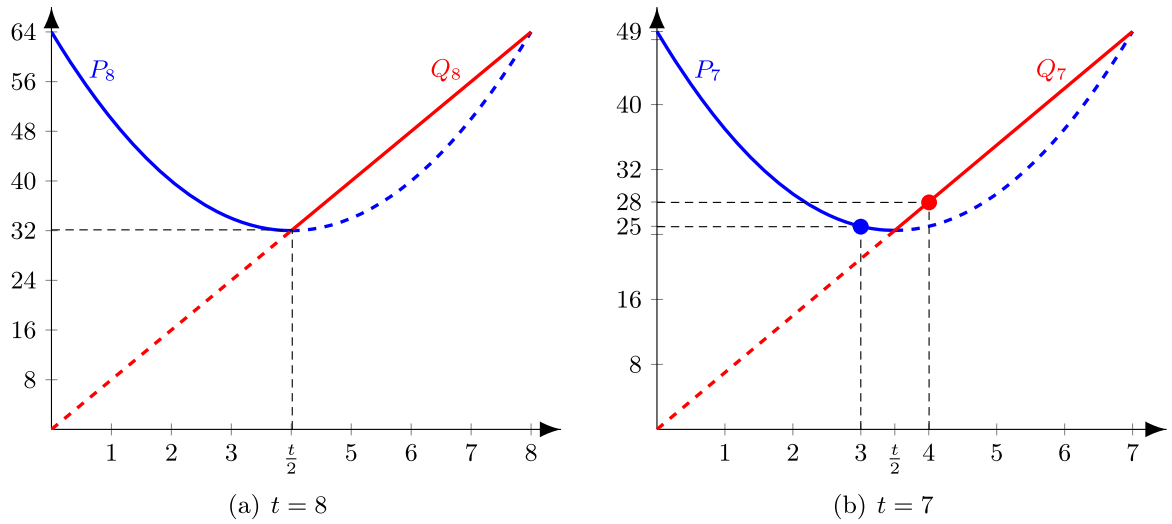


Fig. 6. Plot of the two polynomials P_t and Q_t in two cases: $t = 8$ and $t = 7$.

the cells have been determined, which leads to a probability $2^{ct-2ct} = 2^{-ct}$ of finding a valid element in each of the t' first lists L_i.

All in all, trying all the $2^{c \cdot t \cdot t'}$ elements in Step 1, we find

$$2^{c \cdot t \cdot t' + c(t-2t')(t-t') + (ct - 2c(t-t'))(t-t') - ct \cdot t'} = 1$$

solution during the merge process. We find this solution in time T_m operations, with two cases to consider. Either $t - 2t' \geq 0$, in which case we enumerate $2^{c \cdot t \cdot t'}$ elements in Step 1 followed by the enumeration of $2^{c(t-2t')}$ elements in Step 2. In that case, we have $\log_2(T_m) = ctt' + c(t-2t')(t-t') = 2t'^2 - 2tt' + t^2$. If $t - 2t' \leq 0$, the conditions imposed by the elements enumerated in the first steps make the lists from Step 2 to be nonempty with probability smaller than 1. Hence, we simply have $\log_2(T_m) = ctt'$. This can be summarized by:

$$\log_2(T_m) = \begin{cases} c \cdot P_t(t') & \text{if } t - 2t' \geq 0 \text{ with } P_t = 2X^2 - 2tX + t^2, \\ c \cdot Q_t(t') & \text{if } t - 2t' \leq 0 \text{ with } Q_t = tX. \end{cases} \tag{9}$$

To find the value t' that minimizes the time complexity, we need to determine for which value the minimum of both polynomials P_t and Q_t is reached. For P_t, we get $\frac{t}{2}$ and the nearest integer value satisfying $t - 2t' \geq 0$ is $\lceil \frac{t}{2} \rceil$. For Q_t, we get $\lfloor \frac{t}{2} \rfloor$. For example, see Figs. 6a and 6b, when t equals 8 and 7, respectively.

Consequently, if t is even we set $t' = \frac{t}{2}$, which leads to an algorithm running in $2^{ct^2/2}$ operations and $t' \cdot 2^{ct}$ memory. If t is odd, then we need to decide whether t' should be $\lfloor \frac{t}{2} \rfloor$ or $\lceil \frac{t}{2} \rceil$. If we write $t = 2k + 1$, this is equivalent to find the smallest value between $P_t(k)$ and $Q_t(k+1)$. We find $P_t(k) = 2k^2 + 2k + 1$ and $Q_t(k+1) = 2k^2 + 3k + 1$ so that $P_t(k) < Q_t(k+1)$ (see for example Fig. 6b when $t = 7$). Hence, when t is odd, we fix $t' = \lceil \frac{t}{2} \rceil$. Note that $\frac{t}{2} = \lceil \frac{t}{2} \rceil$ if t is even.

Summing up, for any t, our algorithm performing the merge runs in T_m operations, with:

$$\log_2(T_m) = c \cdot P_t\left(\left\lceil \frac{t}{2} \right\rceil\right) = ct^2 - 2c\left\lfloor \frac{t}{2} \right\rfloor\left\lceil \frac{t}{2} \right\rceil \tag{10}$$

and a memory requirement of $2t' \cdot 2^{ct}$.

Hence, from a pair of random fixed differences $(\delta_{IN}, \delta_{OUT})$, we show how to find a pair of internal states of the permutation that conforms to the middle rounds. To pass the probabilistic transitions of the outbound phase, we need to repeat the merging $2^{2c(t-1)}$ times by picking another couple of differences $(\delta_{IN}, \delta_{OUT})$. In total, we find a pair of inputs to the permutation that conforms to the truncated differential characteristic in time $T_9 = 2^{2c(t-1)} \cdot T_m$ operations, that is:

$$\log_2(T_9) = ct(t+2) - 2c\left(\left\lfloor \frac{t}{2} \right\rfloor\left\lceil \frac{t}{2} \right\rceil + 1\right) \tag{11}$$

with a memory requirement of $t \cdot 2^{ct}$.

3.3. *Comparison with the Ideal Case*

In the ideal case [7], obtaining a pair whose input and output differences lie in a subset of size $IN = OUT = 2^{ct}$ for a ct^2-bit permutation requires

$$2^{\max\{ct(t-1)/2, ct^2-ct-ct\}} = 2^{ct(t-2)}, \tag{12}$$

computations (assuming $t \geq 3$). Therefore, our algorithm distinguishes an AES-like permutation from a random one if and only if its time complexity is smaller than the generic one. This occurs when $\log_2(T_9) \leq ct(t-2)$, which happens as soon as $t \geq 8$. Note that for the AES in the known-key model, we have $t = 4$ and thus our attack does not apply.

One can also derive slightly cheaper distinguishers by aiming at less rounds: instead of using the 9-round truncated characteristic from Fig. 3, it is possible to remove either round 2 or 8 and spare one $t \to 1$ truncated differential transition. Overall, the generic complexity remains the same and this gives a distinguishing attack on the 8-round reduced version of the AES-like permutation with T_8 computations, with:

$$\log_2(T_8) = \log_2(T_m) + c(t-1) = ct(t+1) - c\left(2\left\lfloor \frac{t}{2} \right\rfloor\left\lceil \frac{t}{2} \right\rceil + 1\right) \tag{13}$$

and still 2^{ct} memory provided that $t \geq 6$. If we spare both $t \to 1$ transitions, we end up with a 7-round distinguishing attack with time complexity $T_7 = T_m$ and $t \cdot 2^{ct}$ memory for any $t \geq 4$. Note that those reduced versions of this attack can have a greater time complexity than other techniques: we provide them only for the sake of completeness.

4. Using Non-fully-active Characteristics

4.1. *The Generic Truncated Characteristic*

In [25], Sasaki et al. present new truncated differential characteristics that are not totally active in the middle. Their analysis allows to derive distinguishers for 8 rounds of AES-like permutations with no totally-active state in the middle, provided that the state-size

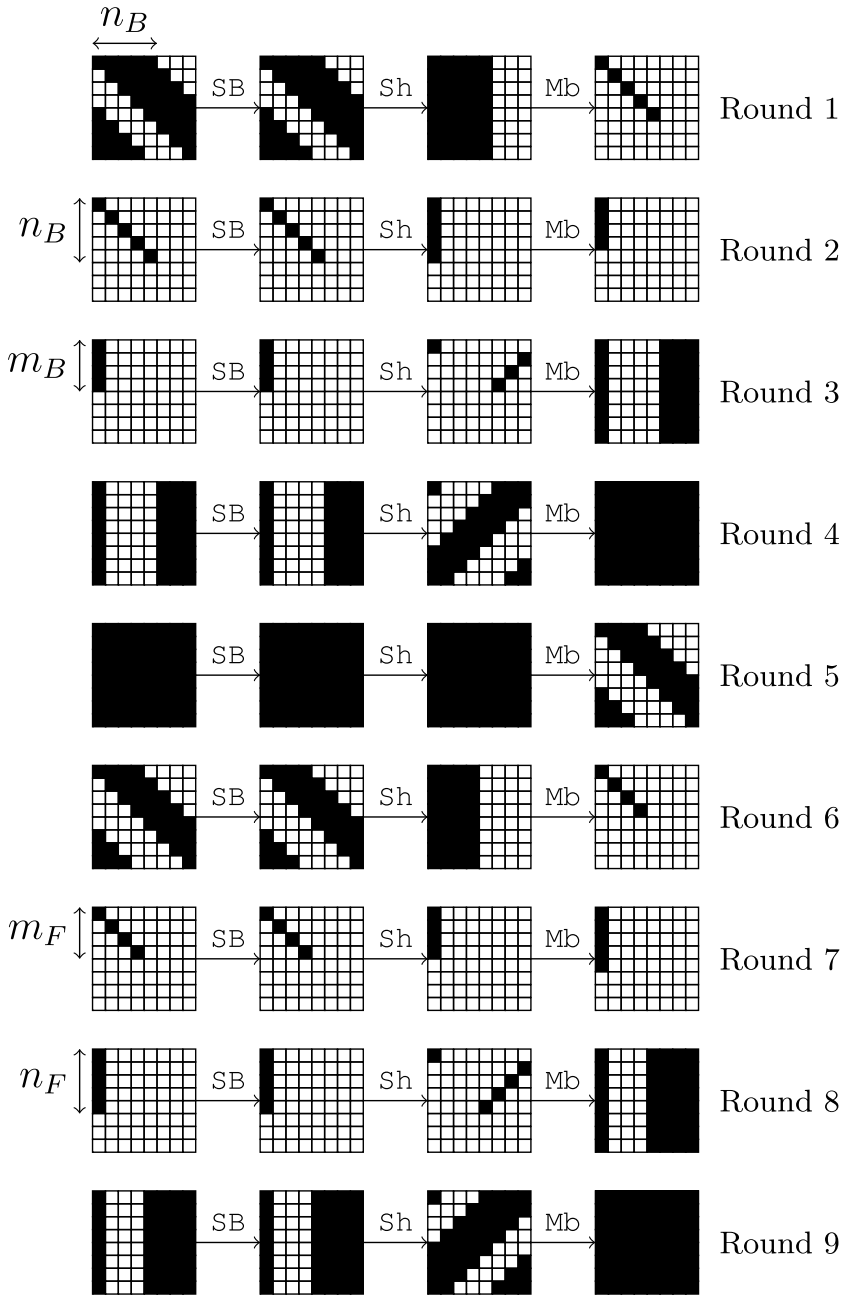


Fig. 7. Non-fully-active truncated differential characteristic on 9 rounds of an AES-like permutation instantiated with $t = 8$.

verifies $t \geq 5$. In this section, we reuse their idea by introducing an additional round in the middle of their trail, which is the unique fully active state of the characteristic. With a similar algorithm as in the previous section, we show how to find a pair conforming to that case.

To keep our reasoning as general as possible, we parameterize the truncated differential characteristic by four variables (see Fig. 7) such that trade-offs will be possible by finding the right values for each one of them. Namely, we denote n_B the number of active diagonals in the plaintext (alternatively, the number of active cells in the second round), n_F the number of active diagonals in the ciphertext (alternatively, the number

of active cells in the eighth round), m_B the number of active cells in the third round and m_F the number of active cells in the seventh round.

Hence, the sequence of active cells in the truncated differential characteristic becomes:

$$\begin{aligned} tn_B \xrightarrow{R_1} n_B \xrightarrow{R_2} m_B \xrightarrow{R_3} tm_B \xrightarrow{R_4} t^2 \xrightarrow{R_5} tm_F \\ \xrightarrow{R_6} m_F \xrightarrow{R_7} n_F \xrightarrow{R_8} tn_F \xrightarrow{R_9} t^2, \end{aligned} \tag{14}$$

with the constraints $n_F + m_F \geq t + 1$ and $n_B + m_B \geq t + 1$ that come from the MDS property. The amount of solutions that can be generated for the differential path equals to ($\log_2$):

$$\begin{aligned} &ct^2 + ctn_B - c(t-1)n_B - c(t-m_B) - ct(t-m_F) - c(t-1)m_F - c(t-n_F) \\ &\quad = c(n_B + n_F + m_B + m_F - 2t). \end{aligned} \tag{15}$$

From the MDS constraints $m_B + n_B \geq t + 1$ and $m_F + n_F \geq t + 1$, we can bound the amount of expected solutions by $2^{c(t+1+t+1-2t)} = 2^{2c}$. This means that, there will always be at least 2^{2c} freedom degrees, independently of t.

4.2. *Finding a Conforming Pair*

As in the previous case, the algorithm that finds a pair of inputs conforming to this characteristic first produces many pairs for the middle rounds and then exhausts them outwards until one passes the probabilistic filter. The cost of those uncontrolled rounds is given by:

$$2^{c(t-n_B)} 2^{c(t-n_F)} = 2^{c(2t-n_B-n_F)}, \tag{16}$$

since we need to pass one $n_B \leftarrow m_B$ transition in the backward direction and one $m_F \rightarrow n_F$ in the forward direction.

We now detail a way to find a solution for the middle rounds (Fig. 8) when the input difference δ_{IN} after the first *SubBytes* layer in state S1 and the output difference δ_{OUT} after the last *MixCells* layer in state S12 are fixed in a way that the truncated characteristic holds in S0 and S12. The beginning of the attack is exactly the same as before in the sense that once the output differences have been fixed, we generate the $2t$ lists that contains the paired values of the t forward *SuperSBox* sets and the t backward *SuperSBox* sets. Again, the same $2t$ lists overlap and we show how to find the solution of the merging problem in $2^{ct \cdot \min(m_F, m_B, \lceil t/2 \rceil)}$ operations and $m_B \cdot 2^{ct}$ memory. We recall that L_i is the ith forward *SuperSBox* list (orange) and L'_i is the ith backward one (blue), for $1 \leq i \leq t$.

We proceed in three steps, where the first guesses the elements from some lists, this determines the remaining cells and we finish by checking probabilistic events. Without loss of generality, we assume in the sequel that $m_B \leq m_F$; if this is not the case, then we start Step 1 by guessing elements of lists L_i in S8. We split the analysis into two cases, whether $m_B \leq \lceil \frac{t}{2} \rceil$ or $m_B > \lceil \frac{t}{2} \rceil$.

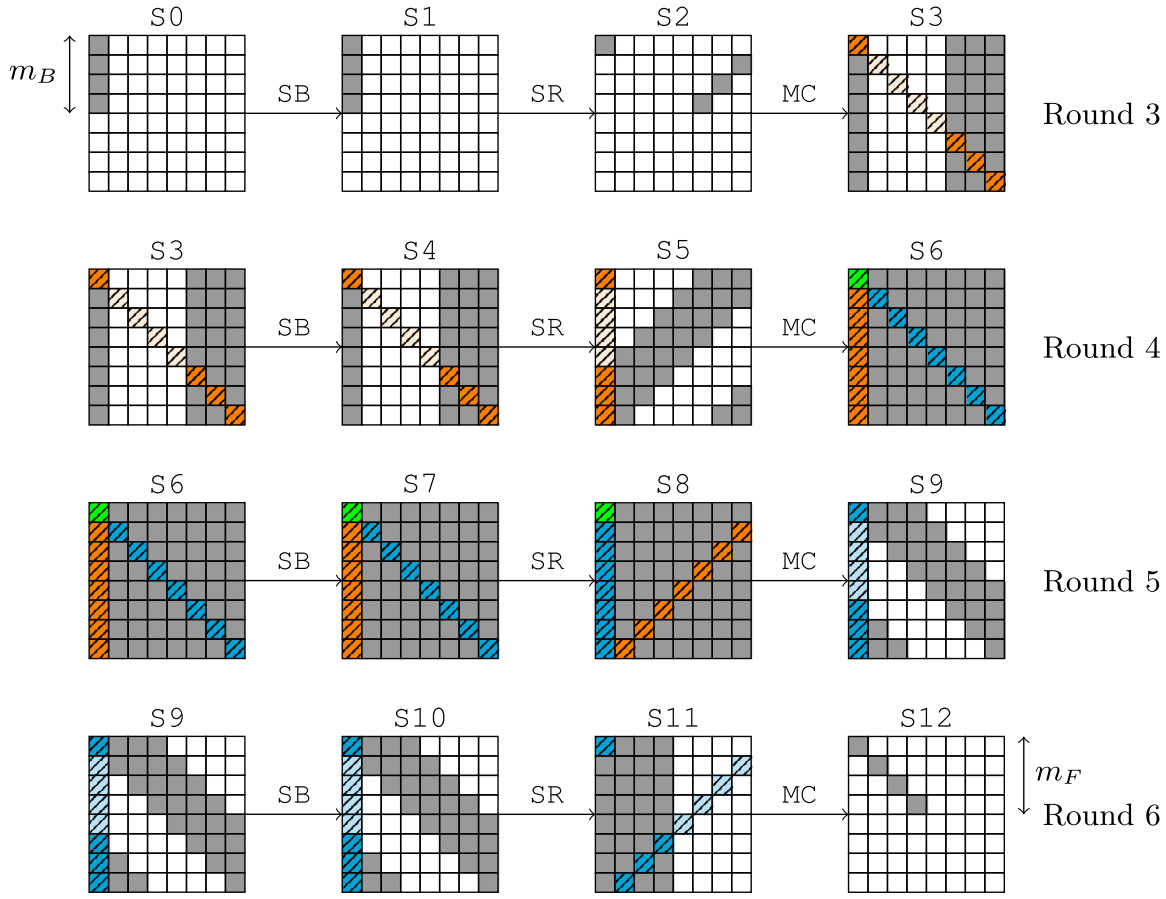


Fig. 8. Inbound phase for the 9-round distinguisher attack on an AES-like permutation instantiated with $t = 8$ with a single fully-active state in the middle. A *gray cell* indicates an active cell; *hatched* and *colored cells* emphasize one *SuperSBox* set: there are seven similar others. (Color figure online)

First case: $m_B \leq \lceil \frac{t}{2} \rceil$. In this case, we use the strong constraints on the vector spaces spanned by the m_B differences on each columns to find a solution to the merge problem.

Step 1. We start by guessing the elements of the m_B lists $L'_1, \ldots, L'_{m_B}$ in state S6. There is a total of 2^{ctm_B} possible combinations.

Step 2. In particular, the previous step sets the differences of the m_B first diagonals of S6 such that there are exactly m_B known differences on each of the t columns of the state. This allows to determine all the differences in S5 since there are exactly m_B independent differences in each column of that state. Consequently, we linearly learn all the differences of S6.

Step 3. Since all differences are known in S6, we determine 1 element in each of the $t - m_B$ remaining L'_i lists: they are of size 2^{ct} and we count ct bits of constraints coming from t differences. From the known differences, we also get a suggestion of 2^{ct-cm_B} values for the cells of each column. Indeed, the elements of the t lists L_i in S5 can be represented as disjointed sets regarding the values of the differences, since the differences can only take 2^{cm_B} values per column. Assuming that they are uniformly distributed,[3] we get $2^{ct}/2^{cm_B} = 2^{ct-cm_B}$ elements per disjointed set

[3] This is a classical assumption, and here it is due to the nonlinear SBox.

for each list: they all share the same value of the differences, but have different values. Additionally, the ct-bit constraints of each list L_i allows to find one element in each, and therefore a solution to the merge problem, with probability $2^{((ct-cm_B)-ct)t} = 2^{-ctm_B}$.

Step 4. Finally, trying all the 2^{ctm_B} elements in $(L'_1, \ldots, L'_{m_B})$, we expect to find $2^{ctm_B}2^{-ctm_B} = 1$ solution that gives a pair of internal states conforming to the four middle rounds with a few operations.

Second case: $m_B > \lceil \frac{t}{2} \rceil$. The columns of differences are less constrained, and it is enough to guess $\lceil \frac{t}{2} \rceil$ lists in the first step to find a solution to the merge problem.

Step 1. We start by guessing the elements of the $\lceil \frac{t}{2} \rceil$ lists $L'_1, \ldots, L'_{m_B}$ in state S6. There is a total of $2^{ct\lceil t/2 \rceil}$ possible combinations.

Step 2. The previous step allows to filter $2^{c(t-2\lceil t/2 \rceil)}$ elements in each of the t lists L_i. Depending of the parity of t, we get 1 element per list for even t, and 2^{-c} for odd ones.[4] In the latter case, there are then a probability 2^{-ct} that the t elements are found in the t lists L_i.

Step 3. In the event that elements have been found in the previous step, we determine completely the remaining $2ct(t - \lceil \frac{t}{2} \rceil)$ values and differences of the remaining $t - \lceil \frac{t}{2} \rceil = \lfloor \frac{t}{2} \rfloor$ lists L'_i. We find a match in those lists with probability $2^{-ct} \times 2^{(ct-2ct)(t-\lceil t/2 \rceil)} = 2^{-ct(1+\lfloor t/2 \rfloor)}$.

Step 4. Finally, trying all the $2^{ct\lceil \frac{t}{2} \rceil}$ elements in $(L'_1, \ldots, L'_{\lceil t/2 \rceil})$, we expect to find $2^{ct\lceil t/2 \rceil}2^{-ct(1+\lfloor t/2 \rfloor)} = 1$ solution that gives a pair of internal states that conforms to the four middle rounds with a few operations.

Hence, in any case, from random differences $(\delta_{IN}, \delta_{OUT})$, we find a pair of internal states of the permutation that conforms to the middle rounds in time $2^{ct\min(m_B, \lceil \frac{t}{2} \rceil)}$ and memory $m_B 2^{ct}$. To pass the probabilistic transitions of the outbound phase, we need to repeat the merging $2^{c(2t-n_B-n_F)}$ times by picking another couple of differences $(\delta_{IN}, \delta_{OUT})$. In total, we find a pair of inputs to the permutation conforming to the truncated differential characteristic in time complexity $2^{ct\min(m_B, \lceil t/2 \rceil)}2^{c(2t-n_B-n_F)} = 2^{c(t(\min(m_B, \lceil t/2 \rceil)+2)-n_B-n_F)}$ and memory complexity $m_B \cdot 2^{ct}$.

Finally, without assuming $m_B \leq m_F$, the time complexity T of the algorithm generalizes to:

$$\log_2(T) = c\left(t \cdot \min\left\{m_B, m_F, \left\lceil \frac{t}{2} \right\rceil\right\} + 2t - n_B - n_F\right), \tag{17}$$

with $n_F + m_F \geq t + 1$ and $n_B + m_B \geq t + 1$, and memory requirement of $m_B \cdot 2^{ct}$.

4.3. *Comparison with Ideal Case*

In the ideal case, the generic complexity $C(a, b)$ is given by the limited birthday distinguisher:

$$\log_{2^c}\big(C(a,b)\big) = \max\left\{\min\left\{\frac{t^2-a}{2}, \frac{t^2-b}{2}\right\}, t^2 - a - b\right\}, \tag{18}$$

[4] Indeed, $t - 2\lceil \frac{t}{2} \rceil = \lfloor \frac{t}{2} \rfloor - \lceil \frac{t}{2} \rceil$ equals 0 when t is even, and -1 when t is odd.

since we get an input space of size $IN = 2^{c \cdot a}$ and output space of size $OUT = 2^{c \cdot b}$. Without loss of generality, assume that $a \leq b$: this only selects whether we attack the permutation or its inverse. In that case, we have:

$$\log_{2^c}\big(C(a,b)\big) = \begin{cases} C_1(a,b) := (t^2 - b)/2, & \text{if: } t^2 < 2a + b, \\ C_2(a,b) := a, & \text{if: } t^2 = 2a + b, \\ C_3(a,b) := t^2 - a - b, & \text{if: } t^2 > 2a + b. \end{cases} \tag{19}$$

In the case of the 9-round distinguisher, the generic complexity equals $C(t \cdot n_B, t \cdot n_F)$ since there are n_B active diagonals at the input, and n_F active diagonals at the output. Let us compare T and the case of $C_3(t \cdot n_B, t \cdot n_F)$ where $t > 2n_B + n_F$ corresponding to the limited birthday distinguisher. We want to find set of values for the parameters (t, n_F, n_B, m_F, m_B) such that our algorithm runs faster that the generic one, that is T is smaller than $C_3(t \cdot n_B, t \cdot n_F)$. In the event that $\min(m_F, m_B, \lceil \frac{t}{2} \rceil)$ is either m_F or m_B, we can show that T is always greater than $C_3(t \cdot n_B, t \cdot n_F)$, and so are the cases involving $C_2(t \cdot n_B, t \cdot n_F)$ and $C_1(t \cdot n_B, t \cdot n_F)$.

We consider the case $\min(m_F, m_B, \lceil \frac{t}{2} \rceil) = \lceil \frac{t}{2} \rceil$:

$$\begin{aligned} &\log_{2^c}\big(C_3(t \cdot n_B, t \cdot n_F)\big) - \log_{2^c}(T) \\ &\quad = t(t - n_F - n_B) - t\left\lceil \frac{t}{2} \right\rceil - 2t + n_B + n_F. \end{aligned} \tag{20}$$

With t as a parameter and $n_F, n_B \in \{1, \ldots, t\}$, our algorithm turns out to be a distinguisher when the quantity from (20) is positive, which is true as soon as

$$(n_B + n_F)(1 - t) + t\left(t - 2 - \left\lceil \frac{t}{2} \right\rceil\right) \geq 0. \tag{21}$$

Since $t - \lceil \frac{t}{2} \rceil = \lfloor \frac{t}{2} \rfloor$, we can show that if $n_F \in \{1, \ldots, t\}$ and $n_B \in \{1, \ldots, t\}$ are chosen such that

$$2 \leq n_F + n_B \leq \frac{t}{t-1}\left(\left\lfloor \frac{t}{2} \right\rfloor - 2\right), \tag{22}$$

then our algorithm is more efficient than the generic one. Note that this may happen only when $t \geq 8$ and that m_F and m_B are still constrained by the MDS bound: $n_F + m_F \geq t + 1$ and $n_B + m_B \geq t + 1$.

We can also consider an 8-round case by considering the characteristic from Fig. 7 where the last round is removed:[5] the generic complexity becomes $C(t \cdot n_B, n_F)$. Note that the complexity of our algorithm remains unchanged: there are still two probabilistic transitions to pass. For $t \geq 4$, we can show that there are many ways to set the parameters (n_F, n_B, m_F, m_B) so that $T \geq C(t \cdot n_B, n_F)$, and the best choice providing the most efficient distinguisher happens when the MDS bounds are tight, i.e.: $n_F + m_F = t + 1$ and $n_B + m_B = t + 1$.

[5] We still assume that $n_B \leq n_F$. If not, then the generic complexity becomes $C(n_B, t \cdot n_F)$ by removing the first round.

Table 2. Examples of reached time complexities for several numbers of rounds and different (t, c) scenarios.

Rounds	Cipher		Parameters				Complexities	
	t	c	n_B	m_B	m_F	n_F	$\log_2(T)$	$\log_2(C)$
9	8	8	1	8	8	1	368	$\log_2 C(t \cdot n_B, t \cdot n_F) = 384$
8	8	8	8	1	4	5	88	$\log_2 C(n_B, t \cdot n_F) = 128$
8	8	8	5	4	1	8	88	$\log_2 C(t \cdot n_B, n_F) = 128$
7	8	8	8	1	1	8	64	$\log_2 C(n_B, n_F) = 384$
8	7	8	7	1	4	4	80	$\log_2 C(n_B, t \cdot n_F) = 112$
8	7	8	4	4	1	7	80	$\log_2 C(t \cdot n_B, n_F) = 112$
7	7	8	7	1	1	7	56	$\log_2 C(n_B, n_F) = 280$
8	4	8	4	1	4	1	56	$\log_2 C(n_B, t \cdot n_F) = 64$
8	4	8	1	4	1	4	56	$\log_2 C(t \cdot n_B, n_F) = 64$
7	4	8	4	1	1	4	32	$\log_2 C(n_B, t \cdot n_F) = 64$

For the sake of completeness, we can also derive distinguishers for 7-round of the permutation by considering the characteristic from Fig. 7 where the first and last rounds are removed, as soon as $t \geq 4$. The generic complexity in that scenario is $C(n_B, n_F)$. Again, there are several ways to set the parameters, but the one that minimizes the runtime T of our algorithm also verifies the MDS bounds: $n_B = 1$, $m_B = t$, $m_F = 1$ and $n_F = t$.

We give examples of more different cases in Table 2, which for instance match AES and Grøstl instantiation. We note that the complexities of our algorithm may be worse that other published results.

5. Applications to Grøstl-256 Permutations

The permutations of the Grøstl-256 hash function implement the previous generic algorithms will the following parameters: $t = 8$, $c = 8$ and $N_r = 10$.

Three Fully-Active States From the analysis of Sect. 3, we can directly conclude that this leads to a distinguishing attack on the 9-round reduced version of the Grøstl-256 permutation with $2^{c(t^2/2+2(t-1))} = 2^{368}$ computations and $2^{ct} = 2^{64}$ memory, when the ideal complexity requires $2^{ct(t-2)} = 2^{384}$ operations.

As detailed previously, we could derive distinguishers for 8-round Grøstl-256 with $2^{c(t^2/2+t-1)} = 2^{312}$ operations and for 7-round Grøstl-256 with $2^{ct^2/2} = 2^{256}$, but those results are more costly than previous known results.

Similarly, as explained in Sect. 2.3, this result also induces a nontrivial observation on the 9-round reduced version of the Grøstl-256 compression function with identical complexity.

Non-fully-active Characteristic With the generic analysis of Sect. 4 that uses a single fully-active middle state, $t = 8$ only allows to instantiate the parameterized truncated differential characteristic with $n_F = n_B = 1$, which determines $m_F = m_B = 8$. Indeed, (22) imposes $2 \leq n_B + n_F \leq \frac{16}{7}$, which gives integer values $n_F = n_B = 1$. Note that it is exactly the case of the three fully-active states in the middle treated in Sect. 3, with the same complexities.

For 8-round distinguishers, the case $t = 8$ where $n_B \leq n_F$ may give the parameters $n_B = 5, m_B = 4, m_F = 1$ and $n_F = 8$ with the last round of the characteristic of Fig. 7 is removed. If $n_B > n_F$, we instantiate the characteristic with the first round removed with the values $n_B = 8$, $m_B = 1$, $m_F = 4$ and $n_F = 5$. In both cases, the time complexity of the distinguishers is 2^{88} operations with 2^{64} of memory requirement, whereas the generic algorithm terminates in about 2^{128} operations. As for 7-round distinguishers, removing both first and last rounds of the characteristic of Fig. 7 leads to an efficient distinguishers for `Grøstl-256` when $n_B = 8$, $m_B = 1$, $m_F = 1$ and $n_F = 8$. The corresponding algorithm runs in 2^{64} operations with 2^{64} of memory requirement, when the corresponding generic algorithm needs 2^{384} operations to terminate. We note that those 8- and 7-round distinguishers are not as efficient as other available techniques: we provide them for the sake of completeness.

6. Conclusion

In this article, we have provided a new and improved cryptanalysis method for `AES`-like permutations, by using a rebound-like approach as well as an algorithm that allows us to control four rounds in the middle of a truncated differential path, with a lower complexity than a general probabilistic approach. To the best of our knowledge, all previously known methods only manage to control three rounds in the middle and we close the open problem whether this was possible or not.

We apply our algorithm on several algorithms and in particular on the building blocks of both the 256 and 512-bit versions of the `SHA-3` finalist `Grøstl`. We could provide the best known distinguishers on 9 rounds of the internal permutations of `Grøstl-256`, while for `Grøstl-512`, we have considerably increased the number of analyzed rounds, from 7 to 10.

These results do not threaten the security of `Grøstl`, but we believe they will have an important role in better understanding `AES`-based functions in general. In particular, we believe that our work will help determining the bounds and limits of rebound-like attacks in these types of constructions. Future works could include the study of more `AES`-like functions in regards to this new cryptanalysis method.

Acknowledgements

We would like to thank the anonymous referees for their valuable comments on our paper. Jérémy Jean is partially supported by the French National Agency of Research through the SAPHIR2 project under Contract ANR-08-VERS-014 and by the French *Délégation Générale pour l'Armement* (DGA). Thomas Peyrin is supported by the Singapore National Research Foundation Fellowship 2012 NRF-NRFF2012-06. This work was partially supported by the French National Agency of Research: ANR-11-INS-011.

Appendix A. Distinguish Attack on 10-Round Grøstl-512

The 512-bit version of the Grøstl hash function uses a non-square 8×16 matrix as 1024-bit internal state, which therefore presents a lack of optimal diffusion: a single difference generates a fully active state after three rounds where a square-state would need only two. This enables us to add an extra round to the generalization of the regular 9-round characteristic of AES-like permutation (Sect. 3) to reach 10 rounds.

A.1. *The Truncated Differential Characteristic*

To distinguish its permutation P_{512}[6] reduced to 10 rounds, we use the truncated differential characteristic with the sequence of active bytes

$$64 \xrightarrow{R_1} 8 \xrightarrow{R_2} 1 \xrightarrow{R_3} 8 \xrightarrow{R_4} 64 \xrightarrow{R_5} 128 \xrightarrow{R_6} 64 \xrightarrow{R_7} 8 \xrightarrow{R_8} 1 \xrightarrow{R_9} 8 \xrightarrow{R_{10}} 64, \quad \text{(A.1)}$$

where the size of the input differences subset is $IN = 2^{512}$ and the size of the output differences subset is $OUT = 2^{64}$.

The actual truncated characteristic is represented on Fig. A.1. Again, we split the characteristic into two parts: the inbound phase involving a merging of lists in the four middle rounds (round 4 to round 7), and an outbound phase that behaves as a probabilistic filter ensuring both $8 \longrightarrow 1$ transitions in the outward directions. Again, passing those two transitions with random values occurs with probability 2^{-112}.

A.2. *Finding a Conforming Pair*

In the following, we present an algorithm to solve the middle rounds in time 2^{280} and memory 2^{64}. In total, we will need to repeat this process 2^{112} times to get a pair of internal states that conforms to the whole truncated differential characteristic, which would then cost $2^{280+112} = 2^{392}$ in time and 2^{64} in memory. The strategy of this algorithm (see Fig. A.2) is similar to the ones presented in [20,21] and the one from the previous section: we start by fixing the difference to a random value δ_{IN} in S1 and δ_{OUT} in S12 and linearly deduce the difference δ'_{IN} in S3 and δ'_{OUT} in S10. Then, we construct the 32 lists corresponding to the 32 *SuperSBoxes*: the 16 forward *SuperSBoxes* have an input difference fixed to δ'_{IN} and cover states S3 to S8, whereas the 16 backward *SuperSBoxes* spread over states S10 to S6 with an output difference fixed to δ'_{OUT}. In the sequel, we denote L_i the 16 forward *SuperSBoxes* and L'_i the backward ones, $1 \leq i \leq 16$.

The 32 lists overlap in S8, where we merge them on 2048 bits[7] to find $2^{64\times32}2^{-2048} = 1$ solution, since each list is of size 2^{64}. The naive way to find the solution would cost 2^{1024} in time by considering each element of the Cartesian product of the 16 lists L_i to check whether it satisfies the output 1024 bit difference condition. We describe now the algorithm that achieves the same goal in time 2^{280}.

First, we observe that due to the geometry of the non-square state, any list L_i intersects with only half of the L'_i. For instance, the first list L_1 associated with the first column of state S7 intersects with lists L'_1, L'_6, L'_{11}, L'_{12}, L'_{13}, L'_{14}, L'_{15} and L'_{16}. We

[6] It would work exactly the same way for the other permutation Q_{512}.

[7] The 2048 bits come from 1024 bits of values and 1024 bits of differences.

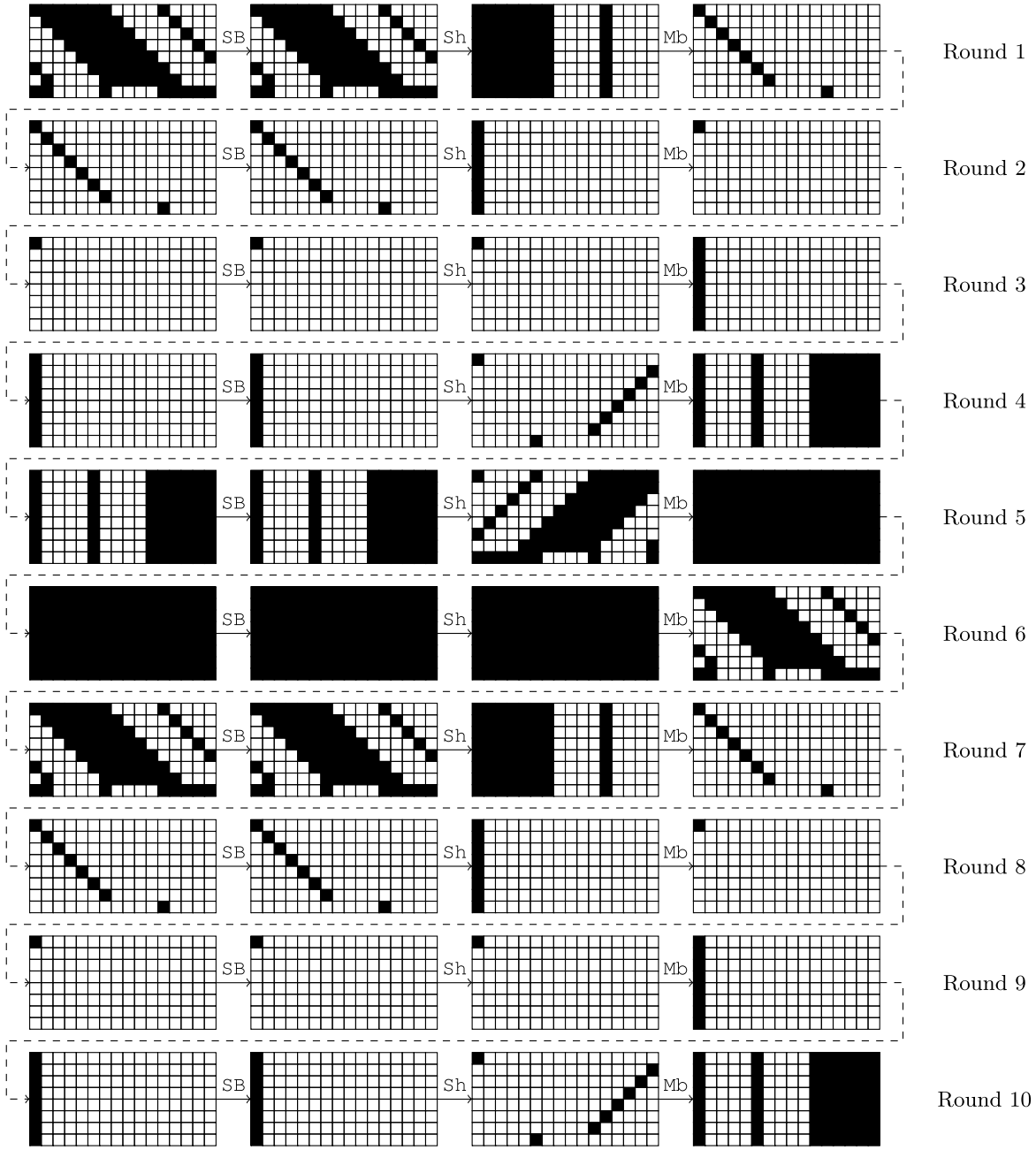


Fig. A.1. The 10-round truncated differential characteristic used to distinguish the permutation `P` of `Grøstl-512` from an ideal permutation.

represent this property with a 16×16 array on Fig. A.3: the 16 columns correspond to the 16 lists L'_i and the lines to the L_i, $1 \leq i \leq 16$. The cell (i, j) is white if and only if L_i has a non-null intersection with the list L'_j, otherwise it is gray.

Then, we note that the *MixCells* transition between the states `S8` and `S9` constraints the differences in the lists L'_i: in the first column of `S9` for example, only three bytes are active, so that the same column in `S8` can only have $2^{3\times 8}$ different differences, which means that knowing three out of the eight differences in an element of L'_1 is enough to deduce the other five. For a column-vector of differences lying in an n-dimensional subspace, we can divide the 2^{64} elements of the associated lists in 2^{8n} disjointed sets of 2^{64-8n} values each. So, whenever we know the n independent differences, the only

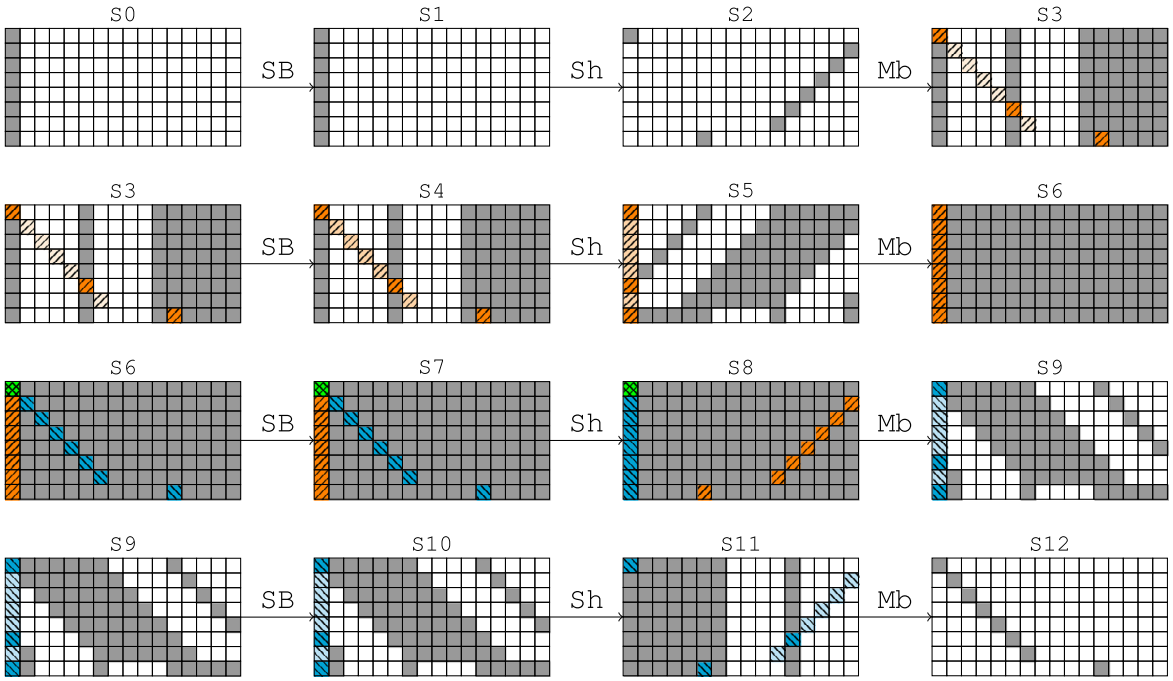
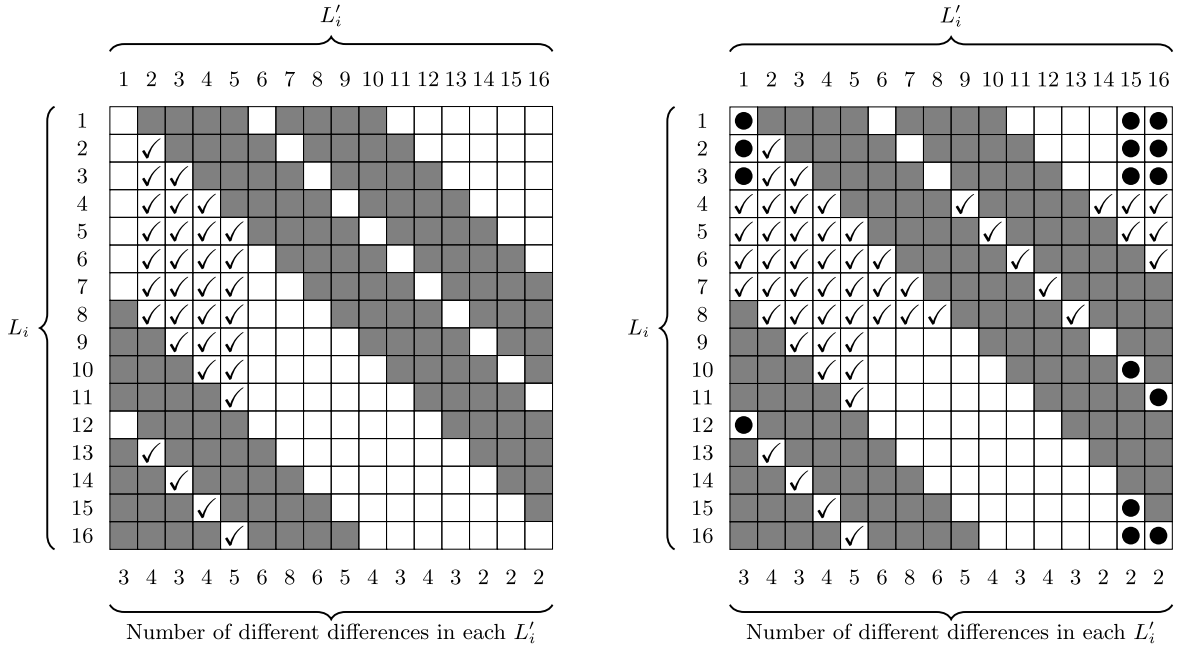
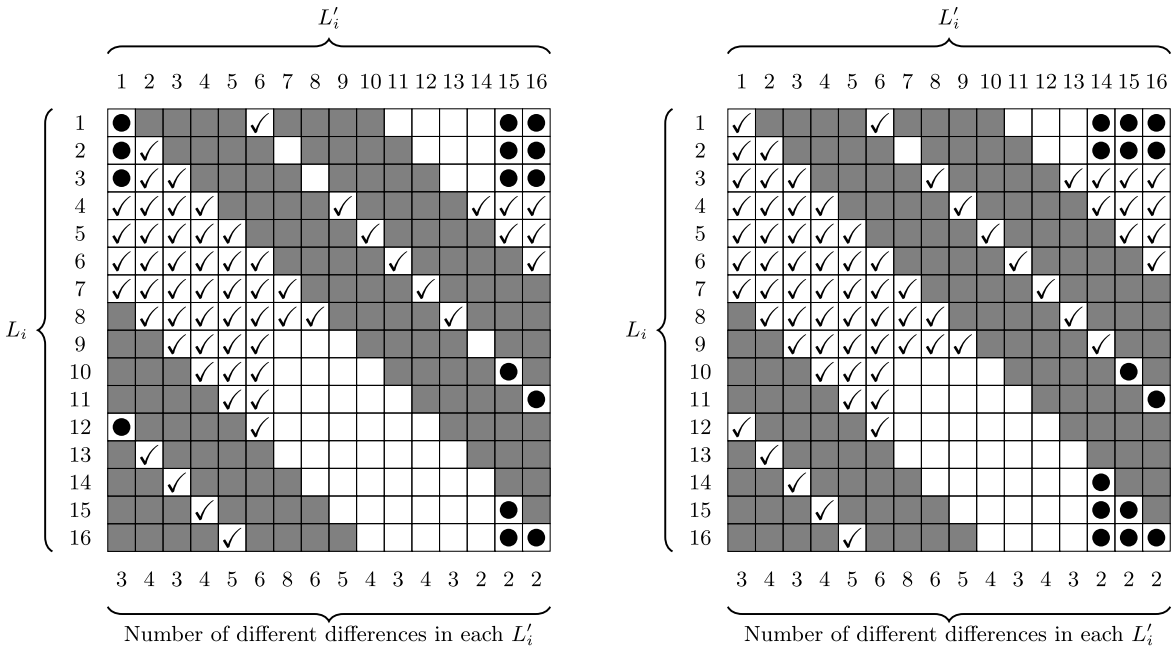


Fig. A.2. Inbound phase for the 10-round distinguisher attack on the Grøstl-512 permutation P_{512}. The four rounds represented are the rounds 4 to 7 from the whole truncated differential characteristic (Fig. A.1). A *gray byte* indicates an active byte; *hatched* and *coloured bytes* emphasize the *SuperSBoxes*. (Color figure online)



(a) Initial guesses. (b) End of first determine step.

Fig. A.3. First guess on the algorithm. A ✓ means we know both value and difference for that byte, a • means that we only determined the difference for that byte and *white bytes* are not constrained yet.



(a) Second guess: L'_6. (b) End of the second guess (L'_6).

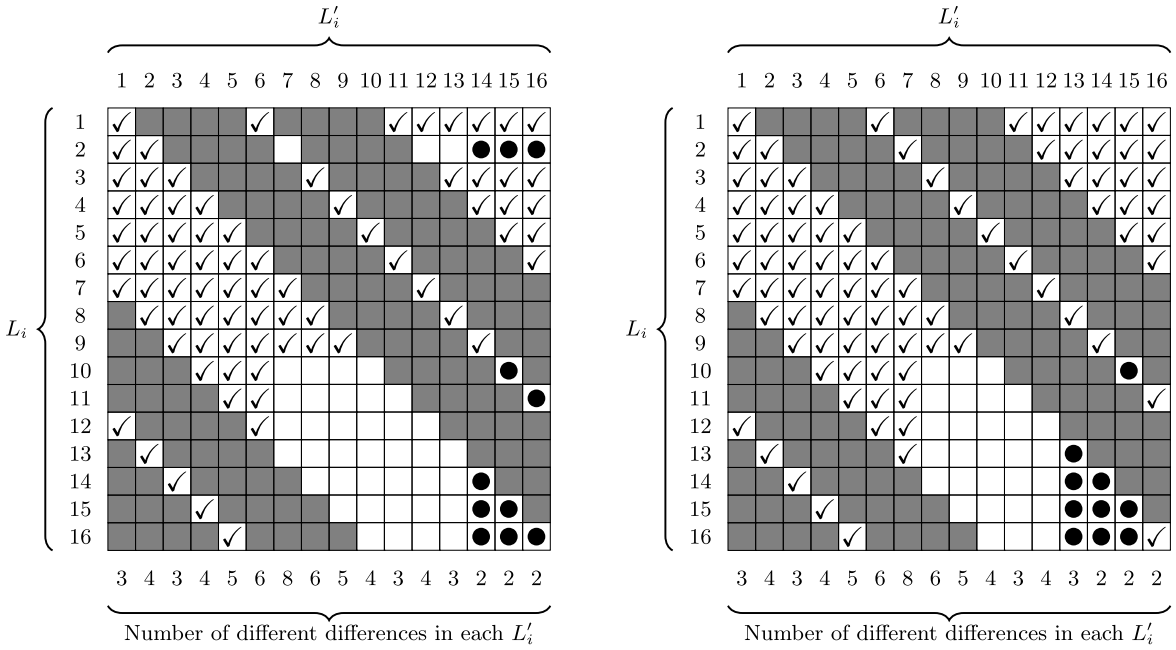
Fig. A.4. Second guess on the algorithm. A ✓ means we know both value and difference for that byte, a • means that we only determined the difference for that byte and *white bytes* are not constrained yet.

freedom that remains lie in the values. The bottom line of Fig. A.3 reports the subspace dimensions for each L'_i.

Using a guess-and-determine approach, we derive a way to use the previous facts to find the solution to the merge problem in time 2^{280}. As stated before, we expect only one solution; that is, we want to find a single element in each of the 32 lists. In the sequel, we describe a sequence of 4 guess-and-determine steps illustrated by pictures before and after each *determine* phase.

Step 1 We start by guessing the values and the differences of the elements associated with the lists L'_2, L'_3, L'_4 and L'_5. For this, we will try all the possible combinations of their elements, there are $2^{4\times 64} = 2^{256}$ in total. For each one of the 2^{256} tries, all the checked cells ✓ from Fig. A.3a now have known value and difference. From here, 8 bytes are known in each of the four lists L_5, L_6, L_7 and L_8: this imposes a 64-bit constraint on those lists, which filter out a single element in each. Thereby, we determined the value and difference in the other 16 bytes marked by ✓ in Fig. A.3b. In lists L'_1 and L'_{16}, we have reached the maximum number of independent differences (three and two, respectively), so we can determine the differences for the other bytes of those columns: we mark them by • . In L_4, the 8 constraints (three ✓ and two •) filter out one element; then, we deduce the correct element in L_4 and mark it by ✓. We can now determine the differences in L'_{15} since the corresponding subspace has a dimension equals to two. See Fig. A.3b for the current situation of the guess-and-determine algorithm.

Step 2 At this point, no more byte can be determined based on the information propagated so far. We continue by guessing the elements remaining in L'_6 (see Fig. A.4a).



(a) Third guess: L_1. (b) End of the third guess (L_1).

Fig. A.5. Third guess on the algorithm. A ✓ means we know both value and difference for that byte, a • means that we only determined the difference for that byte and *white bytes* are not constrained yet.

Since there are already six byte-constraints on that list (three ✓), only 2^{16} elements conform to the conditions. The time complexity until now is thus $2^{256+16} = 2^{272}$.

Guessing the list L'_6 implies a 64-bit constraint of the list L_9 so that we get a single element out of it and determine four yet-unknown other bytes. This enables to learn the independent differences in L'_{14} and therefore, we filter an element from L_3 (two ✓ and four •). At this stage, the list L'_1 is already fully constrained on its differences, so that we are left with a set of $2^{64-3\times 8} = 2^{40}$ values constrained on five bytes (five ✓). Hence, we are able to determine all the unset values in L'_1: see Fig. A.4b for the current situation.

Step 3 Again, the lack of constraints prevent us to determine more bytes. We continue by guessing the 2^8 elements left in L_1 (two ✓ and three •), which makes the time complexity increase to 2^{280} (see Fig. A.5a).

The list L_1 being totally known, we derive the vector of differences in L'_{13}, which adds an extra byte-constraint on L_2 where only one element was left, and so fully determines it. From here, L'_7 becomes fully determined as well (four ✓) and so is L'_{16}. In the latter, the differences being known, we were left with a set of $2^{64-2\times 8} = 2^{48}$ values, which are now constrained on six bytes (six ✓).

Step 4 We describe in Fig. A.5b the knowledge propagated so far, with time complexity 2^{280} and probability 1. In this step, no new guess is needed, and we show how to end the algorithm by probabilistic filterings on the remaining unset lists.

First, we observe that L_{10} is overdetermined (four ✓ and one •) by one byte. This means that we get the correct value with probability 2^{-8}, whereas L_{11} is filtered with probability 1 (four ✓). We assume the correct values are found, such that the element of

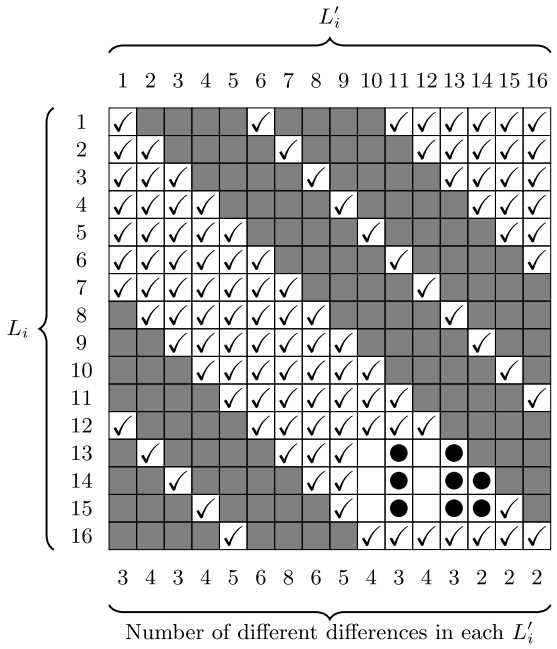


Fig. A.6. End of the guess-and-determine algorithm: after list L_{16} has been fully determined, we filter $L'_{10}, \ldots, L'_{14}$ with probability 1 and then $L_{13}, \ldots, L_{15}$ with probability 2^{-64}.

L'_8 happens to be correctly defined with probability 2^{-16} (five ✓), L'_9 with probability 1 (four ✓) and L'_{15} also with probability 1 since we get 6 ✓ that complete the knowledge of the 2-dimensional subspace of differences (six ✓ and two •). We continue in L'_{11} by learning the full vector of differences (three independent ✓ for a subspace of dimension 3), which constraints L_{12} on 11 bytes (five ✓ and one •) so that we get a valid element with probability 2^{-24}.

At this point, L_{16} is reduced to a single element with probability 2^{-8} (three ✓ and three •), which adds constraints on the three lists L'_{11}, L'_{13} and L'_{14}, where we already know all the differences (Fig. A.6). Consequently, we get respectively 5, 5 and 6 independent values (✓) on subspaces of respective dimensions 3, 3 and 2, which filter those three lists to a single element with probability 1. Finishing the guess-and-determine technique is done by filtering L'_{10} and L'_{12} with probability 1 (four ✓ in a subspace of dimension 4 for both lists), and then the three remaining lists L_{13}, L_{14} and L_{15} are all reduced to a single element which are the valid one with probability 2^{-64} for each (eight ✓). After this, if a solution is found, everything has been determined.

In total, for each guess, we successfully merge the 32 lists with probability

$$2^{-8-16-24-40-64-64-64} = 2^{-280}, \tag{A.2}$$

but the whole procedure is repeated $2^{64\times4+16+8} = 2^{280}$ times, so we expect to find the one existing solution. All in all, we described a way to do the merge with time complexity 2^{280} and memory complexity 2^{64}. The final complexity to find a valid candidate for the whole characteristic is then 2^{392} computations and 2^{64} memory.

A.3. *Comparison with Ideal Case*

In the ideal case, obtaining a pair whose input difference lies in a subset of size $IN = 2^{512}$ and whose output difference lies in a subset of size $OUT = 2^{64}$ for a 1024-bit permutation requires 2^{448} computations. We can directly conclude that this leads to a distinguishing attack on the 10-round reduced version of the Grøstl-512 permutation with 2^{392} computations and 2^{64} memory. Similarly, as explained in Sect. 2.3, this results also induces a nontrivial observation on the 10-round reduced version of the Grøstl-512 compression function with identical complexity.

One can also derive slightly cheaper distinguishers by aiming less rounds while keeping the same generic complexity: instead of using the 10-round truncated characteristic from Fig. A.1, it is possible to remove either round 3 or 9 and spare one $8 \rightarrow 1$ truncated differential transition. Overall, this gives a distinguishing attack on the 9-round reduced version of the Grøstl-512 permutation with 2^{336} computations and 2^{64} memory. By removing both rounds 3 and 9, we achieve 8 rounds with 2^{280} computations.

One can further gain another small factor for the 9-round case by using a $8 \rightarrow 2$ truncated differential transition instead of $8 \rightarrow 1$, for a final complexity of 2^{328} computations and 2^{64} memory. Indeed, the generic complexity drops to 2^{384} because we would now have $OUT = 2^{128}$.

Appendix B. Distinguishers for Reduced PHOTON Permutations

Using the same cryptanalysis technique, it is possible to study the recent lightweight hash function family PHOTON [8], which is based on five different versions of AES-like permutations. Using the notation previously described in this article, the five versions (c, t) for PHOTON are $(4, 5)$, $(4, 6)$, $(4, 7)$, $(4, 8)$ and $(8, 6)$ for increasing versions. All versions are defined to apply $N_r = 12$ rounds of an AES-like process.

Since the internal state is always square, by trivially adapting the method from Sect. 3 to the specific parameters of PHOTON, one can hope to obtain distinguishers for 9 rounds of the PHOTON internal permutations. However, we are able to do so only for the parameters $(4, 8)$ used in PHOTON-224/32/32 (see Table 1 with the comparison to previously known results). Indeed, the size t of the matrix plays an important role in the gap between the complexity of our algorithm and the generic one. The bigger is the matrix, the better will be the gap between the algorithm complexity and the generic one.

References

[1] P.S.L.M. Barreto, V. Rijmen, Whirlpool, in *Encyclopedia of Cryptography and Security*, ed. by H.C.A. van Tilborg, S. Jajodia, 2nd edn. (Springer, Berlin, 2011), pp. 1384–1385
[2] R. Benadjila, O. Billet, H. Gilbert, G. Macario-Rat, T. Peyrin, M. Robshaw, Y. Seurin, SHA-3, proposal: ECHO. Submission to NIST (updated) (2009)
[3] C. Boura, A. Canteaut, C.D. Cannière, Higher-order differential properties of Keccakand Luffa, in *FSE*. LNCS, vol. 6733 (Springer, Berlin, 2011), pp. 252–269
[4] J. Daemen, V. Rijmen, Rijndael for AES, in *AESCandidate Conference* (2000), pp. 343–348
[5] P. Gauravaram, L.R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schläffer, S.S. Thomsen, Grøstl—a SHA-3candidate. Submitted to the SHA-3competition, NIST (2008)
[6] P. Gauravaram, L.R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schläffer, S.S. Thomsen, Grøstl—a SHA-3candidate (Updated version). Submitted to the SHA-3competition (2011)

[7] H. Gilbert, T. Peyrin, Super-sbox cryptanalysis: improved attacks for AES-like permutations, in *Lecture Notes in Computer Science*, FSE, vol. 6147, ed. by S. Hong, T. Iwata (Springer, Berlin, 2010), pp. 365–383
[8] J. Guo, T. Peyrin, A. Poschmann, The PHOTONfamily of lightweight hash functions, in *Lecture Notes in Computer Science*, CRYPTO, vol. 6841, ed. by P. Rogaway (Springer, Berlin, 2011), pp. 222–239
[9] J. Guo, T. Peyrin, A. Poschmann, M.J.B. Robshaw, The LEDblock cipher, in *Lecture Notes in Computer Science.*, CHES, vol. 6917, ed. by B. Preneel, T. Takagi (Springer, Berlin, 2011), pp. 326–341
[10] J. Jean, P.A. Fouque, Practical near-collisions and collisions on round-reduced ECHO-256 compression function, in *Lecture Notes in Computer Science*, FSE, vol. 6733, ed. by A. Joux (Springer, Berlin, 2011), pp. 107–127
[11] J. Jean, M. Naya-Plasencia, M. Schläffer, Improved analysis of ECHO-256, in *Selected Areas in Cryptography*, ed. by A. Miri, S. Vaudenay. Lecture Notes in Computer Science, vol. 7118 (Springer, Berlin, 2011), pp. 19–36
[12] J. Jean, M. Naya-Plasencia, T. Peyrin, Improved rebound attack on the finalist Grøstl, in *Lecture Notes in Computer Science*, FSE, vol. 7549, ed. by A. Canteaut (Springer, Berlin, 2012), pp. 110–126
[13] L.R. Knudsen, Truncated and higher order differentials, in *Lecture Notes in Computer Science*, FSE, vol. 1008, ed. by B. Preneel (Springer, Berlin, 1994), pp. 196–211
[14] M. Lamberger, F. Mendel, C. Rechberger, V. Rijmen, M. Schläffer, Rebound Distinguishers: Results on the Full Whirlpool Compression Function. [15] 126–143
[15] M. Matsui (ed.), Advances in cryptology—ASIACRYPT 2009, 15th international conference on the theory and application of cryptology and information security, Tokyo, Japan, December 6–10 (2009). Proceedings, in *Lecture Notes in Computer Science*, ASIACRYPT, vol. 5912, ed. by M. Matsui (Springer, Berlin, 2009)
[16] K. Matusiewicz, M. Naya-Plasencia, I. Nikolic, Y. Sasaki, M. Schläffer, Rebound Attack on the Full LANECompression Function. [15] 106–125
[17] F. Mendel, T. Peyrin, C. Rechberger, M. Schläffer, Improved cryptanalysis of the reduced Grøstlcompression function, ECHOpermutation and AESblock cipher, in *Selected Areas in Cryptography*, ed. by M.J. Jacobson Jr., V. Rijmen, R. Safavi-Naini. Lecture Notes in Computer Science., vol. 5867 (Springer, Berlin, 2009), pp. 16–35
[18] F. Mendel, C. Rechberger, M. Schläffer, S.S. Thomsen, The rebound attack: cryptanalysis of reduced Whirlpooland Grøstl, in *Fast Software Encryption—FSE 2009*. Lecture Notes in Computer Science., vol. 5665 (Springer, Berlin, 2009)
[19] F. Mendel, C. Rechberger, M. Schläffer, S.S. Thomsen, Rebound attacks on the reduced Grøstlhash function, in *Lecture Notes in Computer Science*, CT-RSA vol. 5985, ed. by J. Pieprzyk (Springer, Berlin, 2010), pp. 350–365
[20] M. Naya-Plasencia, How to Improve Rebound Attacks. Cryptology ePrint Archive, Report 2010/607 (extended version) (2010)
[21] M. Naya-Plasencia, How to improve rebound attacks, in *Advances in Cryptology: CRYPTO 2011*. Lecture Notes in Computer Science, vol. 6841 (Springer, Berlin, 2011), pp. 188–205
[22] I. Nikolic, J. Pieprzyk, P. Sokolowski, R. Steinfeld, Known and chosen key differential distinguishers for block ciphers, in *Lecture Notes in Computer Science*, ICISC, vol. 6829, ed. by K.H. Rhee, D. Nyang (Springer, Berlin, 2010), pp. 29–48
[23] T. Peyrin, Cryptanalysis of Grindahl, in *Lecture Notes in Computer Science*, ASIACRYPT, vol. 4833, ed. by K. Kurosawa (Springer, Berlin, 2007), pp. 551–567
[24] T. Peyrin, Improved differential attacks for ECHOand Grøstl, in *Lecture Notes in Computer Science*, CRYPTO, vol. 6223, ed. by T. Rabin (Springer, Berlin, 2010), pp. 370–392
[25] Y. Sasaki, Y. Li, L. Wang, K. Sakiyama, K. Ohta, Non-full-active super-sbox analysis: applications to ECHOand Grøstl, in *Lecture Notes in Computer Science*, ASIACRYPT, vol. 6477, ed. by M. Abe (Springer, Berlin, 2010), pp. 38–55
[26] X. Wang, H. Yu, How to break MD5and other hash functions, in *Lecture Notes in Computer Science*, EUROCRYPT vol. 3494, ed. by R. Cramer (Springer, Berlin, 2005), pp. 19–35
[27] X. Wang, Y.L. Yin, H. Yu, Finding collisions in the full SHA-1, in *Lecture Notes in Computer Science*, CRYPTO vol. 3621, ed. by V. Shoup (Springer, Berlin, 2005), pp. 17–36

Conditional Differential Cryptanalysis of NLFSR-based Cryptosystems

Simon Knellwolf*, Willi Meier, and María Naya-Plasencia**

FHNW, Switzerland

Abstract. Non-linear feedback shift registers are widely used in lightweight cryptographic primitives. For such constructions we propose a general analysis technique based on differential cryptanalysis. The essential idea is to identify conditions on the internal state to obtain a deterministic differential characteristic for a large number of rounds. Depending on whether these conditions involve public variables only, or also key variables, we derive distinguishing and partial key recovery attacks. We apply these methods to analyse the security of the eSTREAM finalist Grain v1 as well as the block cipher family KATAN/KTANTAN. This allows us to distinguish Grain v1 reduced to 104 of its 160 rounds and to recover some information on the key. The technique naturally extends to higher order differentials and enables us to distinguish Grain-128 up to 215 of its 256 rounds and to recover parts of the key up to 213 rounds. All results are the best known thus far and are achieved by experiments in practical time.

Keywords: differential cryptanalysis, NLFSR, distinguishing attack, key recovery, Grain, KATAN/KTANTAN

1 Introduction

For constrained environments like RFID tags or sensor networks a number of cryptographic primitives, such as stream ciphers and lightweight block ciphers have been developed, to provide security and privacy. Well known such cryptographic algorithms are the stream ciphers Trivium [5] and Grain [12, 13] that have been selected in the eSTREAM portfolio of promising stream ciphers for small hardware [9], and the block cipher family KATAN/KTANTAN [6]. All these constructions build essentially on non-linear feedback shift registers (NLFSRs). These facilitate an efficient hardware implementation and at the same time enable to counter algebraic attacks.

Stream ciphers and block ciphers both mix a secret key a and public parameter (the initial value for stream ciphers and the plaintext for block ciphers) in an involved way to produce the keystream or the ciphertext, respectively.

* Supported by the Hasler Foundation `www.haslerfoundation.ch` under project number 08065.

** Supported by an ERCIM "Alain Bensoussan" Fellowship Programme.

In cryptanalysis, such systems are often analysed in terms of boolean functions that to each key k and public parameter x assign an output bit $f(k, x)$. Several cryptanalytic methods analyse derived functions from f. They can be roughly divided into algebraic and statistical methods. The cube attack presented in [8] is an algebraic method. It consists in finding many derivatives of f that are linear in the key bits such that the key can be found by solving a system of linear equations. The d-monomial test introduced in [10] provides a statistical framework to analyse the distribution of degree d monomials in the algebraic normal form of f. Another statistical approach is presented in [11, 14], where the concept of probabilistc neutral key bits is applied to derivatives of f. The notion of cube testers introduced in [2] covers many of these methods. All of them have in common that they interact with f mainly in a black box manner, exploiting the structure of the underlying primitive only indirectly.

In this paper we propose a general analysis principle that we call *conditional differential cryptanalysis*. It consists in analysing the output frequency of derivatives of f on specifically chosen plaintexts (or initial values). Differential cryptanalyis, introduced in [4] for the analysis of block ciphers, studies the propagation of an input difference through an iterated construction and has become a common tool in the analysis of initialization mechanisms of stream ciphers, see [3, 7, 18]. In the case of NLFSR-based constructions, only few state bits are updated at each iteration, and the remaining bits are merely shifted. This results in a relatively slow diffusion. Inspired by message modification techniques introduced in [17] for hash function cryptanalysis, we trace the differences round by round and identify conditions on the internal state bits that control the propagation of the difference through the initial iterations. From these conditions we derive plaintexts (or initial values) that follow the same characteristic at the initial rounds and allow us to detect a bias in the output difference. In some cases the conditions also involve specific key bits which enables us to recover these bits in a key recovery attack.

The general idea of conditional differential cryptanalysis has to be elaborated and adapted with respect to each specific primitive. This is effected for the block cipher family KATAN and its hardware optimized variant KTANTAN as well as for the stream ciphers Grain v1 and Grain-128. The analysis of the block cipher family KATAN/KTANTAN is based on first order derivatives and nicely illustrates our analysis principle. For a variant of KATAN32 reduced to 78 of the 254 rounds we can recover at least two key bits with probability almost one and complexity 2^{22}. Comparable results are obtained for the other members of the family. We are not aware of previous cryptanalytic results on the KATAN/KTANTAN family. The analysis of Grain v1 is similar to that of KATAN, however the involved conditions are more sophisticated. We obtain a practical distinguisher for up to 104 of the 160 rounds. The same attack can be used to recover one key bit and four linear relations in key bits with high probability. Grain v1 was previously analysed in [7], where a sliding property is used to speed up exhaustive search by a factor two, and in [1], where a non-randomness property for 81 rounds could be detected.

Conditional differential cryptanalysis naturally extends to higher order derivatives. This is demonstrated by our analysis of Grain-128, which, compared to Grain v1, is surprisingly more vulnerable to higher order derivatives. We get a practical distinguisher for up to 215 of the 256 rounds and various partial key recovery attacks for only slightly less rounds. For a 197 round variant we recover eight key bits with probability up to 0.87, for a 213 round variant two key bits with probability up to 0.59. The previously best known cryptanalytic result was a theoretical key recovery attack on 180 rounds, and was able to speed up exhaustive key search by a factor 2^4, but without the feasibility to predict the value of single key bits, see [11]. Moreover, a result in [7] mentions key recovery for up to 192 rounds and in [1] a non-randomness property was detected in a chosen key scenario.

The paper is organised as follows. Section 2 recalls the definition of higher order derivatives of boolean functions and discusses the application of frequency tests to such derivatives. Section 3 provides the general idea of conditional differential cryptanalysis of NLFSR-based cryptosystems. In the Sections 4, 5 and 6 this idea is refined and adapted to a specific analysis of the KATAN/KTANTAN family, Grain v1 and Grain-128.

2 Notation and Preliminaries

In this paper $\mathbb{F}_2$ denotes the binary field and $\mathbb{F}_2^n$ the n-dimensional vector space over $\mathbb{F}_2$. Addition in $\mathbb{F}_2$ is denoted by $+$, whereas addition in $\mathbb{F}_2^n$ is denoted by $\oplus$ to avoid ambiguity. For $0 \leq i \leq n-1$ we denote $e_i \in \mathbb{F}_2^n$ the vector with a one at position i and zero otherwise.

We now recall the definition of the i-th derivative of a boolean function introduced in [15, 16] and we discuss the application of a frequency test to such derivatives.

2.1 Derivatives of Boolean Functions

Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a boolean function. The *derivative of f with respect to* $a \in \mathbb{F}_2^n$ is defined as

$$\Delta_a f(x) = f(x \oplus a) + f(x).$$

The derivative of f is itself a boolean function. If $\sigma = \{a_1, \ldots, a_i\}$ is a set of vectors in $\mathbb{F}_2^n$, let $L(\sigma)$ denote the set of all 2^i linear combinations of elements in σ. The *i-th derivative of f with respect to σ* is defined as

$$\Delta_\sigma^{(i)} f(x) = \sum_{c \in L(\sigma)} f(x \oplus c).$$

We note that the i-th derivative of f can be evaluated by summing up 2^i evaluations of f. We always assume that $a_1, \ldots, a_i$ are linearly independent, since otherwise $\Delta_\sigma^{(i)} f(x) = 0$ trivially holds. If we consider a keyed boolean function $f(k, \cdot)$ we always assume that the differences are applied to the second argument and not to the key.

2.2 Random Boolean Functions and Frequency Test

Let D be a non-empty subgroup of $\mathbb{F}_2^n$. A *random boolean function on* D is a function $D \to \mathbb{F}_2$ whose output is an independent uniformly distributed random variable. If f is a random boolean function on D, the law of large numbers says that for sufficiently many inputs $x_1, \ldots, x_s \in D$ the value

$$t = \frac{\sum_{k=1}^{s} f(x_k) - s/2}{\sqrt{s/4}}$$

approximately follows a standard normal distribution. Denoting

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}u^2} du$$

the standard normal distribution function, a boolean function is said to *pass the frequency test* on $x_1, \ldots, x_s$ at a significance level α if

$$\Phi(t) < 1 - \frac{\alpha}{2}$$

A random boolean function passes the frequency test with probability $1 - \alpha$. If the frequency test is used to distinguish a keyed boolean function $f(k, \cdot)$ from a random boolean function, we denote by β the probability that $f(k, \cdot)$ passes the frequency test for a random key k. The distinguishing advantage is then given by $1 - \alpha - \beta$.

2.3 Frequency Test on Derivatives

If $\sigma = \{a_1, \ldots, a_i\}$ is a set of linearly independent differences, the i-th derivative of a boolean random function is again a boolean random function. Its output is the sum of 2^i independent uniformly distributed random variables. But for any two inputs x, x' with $x \oplus x' \in L(\sigma)$ the output values are computed by the same sum and thus $\Delta_\sigma^{(i)} f(x) = \Delta_\sigma^{(i)} f(x')$. Hence, the i-th derivative is not a random function on D, but on the quotient group $D/L(\sigma)$. A frequency test of $\Delta_\sigma^{(i)} f$ on s inputs needs $s2^i$ queries to f.

3 Conditional Differential Cryptanalysis of NLFSR

This section provides the general idea of our analysis. It is inspired by message modification techniques as they were introduced in [17] to speed up the collision search for hash functions. We trace differences through NLFSR-based cryptosystems and exploit the non-linear update to prevent their propagation whenever possible. This is achieved by identifying conditions on the internal state variables of the NLFSR. Depending on whether these conditions involve the public parameter or also the secret key, they have to be treated differently in a chosen

plaintext attack scenario. The goal is to obtain many inputs that satisfy the conditions, i.e. that follow the same differential characteristic at the initial rounds. In more abstract terms, we analyse derivatives of keyed boolean functions and exploit that their output values are iteratively computed.

We briefly explain NLFSR-based cryptosystems and why our analysis principle applies to them. Then we define three types of conditions that control the difference propagation in NLFSR-based cryptosystems and we explain how to deal with each of these types in a chosen plaintext (chosen initial value) attack scenario. The basic strategy is refined and adapted in the later sections to derive specific attacks on KATAN/KTANTAN, Grain v1 and Grain-128.

3.1 NLFSR-based Cryptosystems

An NLFSR of length l consists of an initial state $s_0, \ldots, s_{l-1} \in \mathbb{F}_2$ and a recursive update formula $s_{l+i} = g(s_i, \ldots, s_{l+i-1})$ for $i \geq 0$, where g is a non-linear boolean function. The bit s_{l+i} is called the *bit generated at round* i and $s_i, \ldots, s_{l+i-1}$ is called the *state of round* $i-1$. Our analysis principle applies to any cryptographic construction that uses an NLFSR as a main building block. These constructions perform a certain number of rounds, generating at each round one or more bits that non-linearly depend on the state of the previous round. It is this non-linear dependency that we exploit in conditional differential cryptanalysis.

Let $f : \mathbb{F}_2^m \times \mathbb{F}_2^n \to \mathbb{F}_2$ denote the keyed boolean function that to every key k and public parameter x assigns one output bit $f(k, x)$ of an NLFSR-based construction. If we consider a first order derivative of the function f, we apply a difference $a \in \mathbb{F}_2^n$ to the public parameter. The value $\Delta_a f(k, x)$ then denotes the output difference $f(k, x) + f(k, x \oplus a)$. If s_i is a state bit of our construction, we denote $\Delta_a s_i(k, x)$ the difference in this state bit for the key k, the public parameter x and the difference a.

3.2 Conditions and Classification

We now introduce the concepts of our analysis principle. In general, the difference of a newly generated state bit depends on the differences and the values of previously generated state bits. Each time that $\Delta_a s_i(k, x)$ non-linearly depends on a bit that contains a difference, we can identify conditions on previously generated state bits that control the value of $\Delta_a s_i(k, x)$. In most cases, the conditions are imposed to prevent the propagation of the difference to the newly generated state bits. In particular it is important to prevent the propagation at the initial rounds. Since we want to statistically test the frequency of $\Delta_a f(k, \cdot)$ on inputs that satisfy the conditions, there is an important tradeoff between the number of imposed conditions and the number of inputs that we can derive. The conditions can not only involve bits of x, but also bits of k. We classify them into three types:

- Type 0 conditions only involve bits of x.
- Type 1 conditions involve bits of x and bits of k.

– Type 2 conditions only involve bits of k.

In a chosen plaintext (chosen initial value) scenario, type 0 conditions can easily be satisfied by the attacker, whereas he cannot control type 2 conditions at all. In most cases, type 2 conditions consist of simple equations and the probability that they are satisfied for a uniformly random key can easily be determined. Since we do not assume that our attacks can be repeated for more than one key, type 2 conditions generally decrease the advantage of distinguishing attacks and define classes of weak keys for this kind of attacks. On the other hand we specifically exploit type 2 conditions to derive key recovery attacks based on hypothesis tests. This is explained in Section 6 where we analyse Grain-128.

In a different way, also type 1 conditions can be used to recover parts of the key. To deal with the type 1 conditions, we introduce the concept of free bits. Suppose that the state bit s_i depends on x as well as on some bits of k, and suppose that we want to satisfy the type 1 condition $s_i = 0$. In a chosen plaintext scenario, we cannot control this condition in a simple way. We call those bits of x that do not influence the value of s_i for any key k, the *free bits* for the condition. The remaining bits of x are called *non-free*. Together with k the non-free bits determine whether the condition is satisfied or not. We call x a *valid input* if, for a given key k, it satisfies the imposed condition. If we define the set φ as $\varphi = \{e_i \in \mathbb{F}_2^n | x_i \text{ is a free bit}\}$ then we can generate $2^{|\varphi|}$ valid inputs from a single valid input x: these are the elements of the coset $x \oplus L(\varphi)$. In general, more than one type 1 condition are imposed. In that case, the free bits are those that are free for all of these conditions. In some cases it may be possible to give a finite number of configurations for the non-free bits such that at least one configuration determines a valid input. Otherwise, if t type 1 conditions are imposed, we expect that about one of 2^t different inputs is valid and we just repeat the attack several times with different random inputs.

In some cases we can not obtain enough inputs only by the method of free bits. We then try to find non-free bits that only must satisfy a given equation but otherwise can be freely chosen. This provides us with more degrees of freedom to generate a sample of valid inputs. We refer to the analysis of KATAN and Grain v1 for concrete examples of this method.

3.3 Choosing the Differences

The choice of a suitable difference for conditional differential cryptanalysis is not easy and strongly depends on the specific construction. In particular this holds for higher order derivatives, but also for first order ones. In general, the difference propagation should be controllable for as many rounds as possible with a small number of conditions. In particular, there should not be too many type 1 and type 2 conditions at the initial rounds. Differences which can be controlled by isolated conditions of type 1 or type 2 are favorable for key recovery attacks.

The set of differences for higher order derivatives can be determined by combining first order differences whose characteristics do not influence each other

at the initial rounds. In a non-conditional setting, [1] describes a genetic algorithm for finding good sets of differences. This black-box approach did not yield particularly good sets for our conditional analysis.

4 Analysis of KATAN/KTANTAN

KATAN/KTANTAN is a family of lightweight block ciphers proposed in [6]. The family consists of six ciphers denoted by KATANn and KTANTANn for $n = 32, 48, 64$ indicating the block size of the cipher. All instances accept an 80-bit key and use the same building blocks, namely two NLFSRs and a small LFSR acting as a counter. The only difference between KATANn and KTANTANn is the key scheduling.

In the following we describe KATAN32 and provide the details of our analysis for this particular instance of the family. Our analysis of the other instances is very similar. We only sketch the differences and provide the empirical results.

We emphasize that our analysis does not reveal a weakness of any of the original KATAN/KTANTAN ciphers. In contrary, with respect to our method, it seems that the number of rounds is sufficiently large to provide a confident security margin.

4.1 Description of KATAN32

The two NLFSRs of KATAN32 have length 13 and 19 and we denote their states by $l_i, \ldots, l_{i+12}$ and $r_i, \ldots, r_{i+18}$, respectively. A 32-bit plaintext block x is loaded to the registers by $l_i = x_{31-i}$ for $0 \leq i \leq 12$ and $r_i = x_{18-i}$ for $0 \leq i \leq 18$. The LFSR has length 8 and we denote its state by $c_i, \ldots, c_{i+7}$. Initialization is done by $c_i = 1$ for $0 \leq i \leq 6$ and $c_7 = 0$. The full encryption process takes 254 rounds defined by

$$\begin{aligned} c_{i+8} &= c_i + c_{i+1} + c_{i+3} + c_{i+8}, \\ l_{i+13} &= r_i + r_{i+11} + r_{i+6}r_{i+8} + r_{i+10}r_{i+15} + k_{2i+1}, \\ r_{i+19} &= l_i + l_{i+5} + l_{i+4}l_{i+7} + l_{i+9}c_i + k_{2i}, \end{aligned}$$

where $k_0, \ldots, k_{79}$ are the bits of the key and k_i is recursively computed by

$$k_{j+80} = k_j + k_{j+19} + k_{j+30} + k_{j+67}$$

for $i \geq 80$. Finally, the states of the two NLFSRs are output as the ciphertext. If we consider a round-reduced variant of KATAN32 with r rounds, the bits l_{r+i} for $0 \leq i \leq 12$ and r_{r+i} for $0 \leq i \leq 18$ will be the ciphertext.

4.2 Key Recovery for KATAN32 Reduced to 78 Rounds

Our analysis is based on a first order derivative and uses the concept of free bits to satisfy type 1 conditions. Here, to obtain enough inputs, we will identify

non-free bits that only must satisfy an underdefined system of linear equations, which gives us more freedom degrees generate the samples.

We consider a difference of weight five at the positions 1,7,12,22 and 27 of the plaintext block. Let $a = e_1 \oplus e_7 \oplus e_{12} \oplus e_{22} \oplus e_{27}$ denote the initial difference. At round 0 we have

$$\begin{aligned} \Delta_a l_{13}(k, x) &= 1 + x_{10}, \\ \Delta_a r_{19}(k, x) &= x_{24} + 1 \end{aligned}$$

and impose the conditions $x_{10} = 1$ and $x_{24} = 1$ to prevent the difference propagation. Similarly at the rounds 1, 2, 3 and 5, we impose the bits $x_2, x_6, x_5, x_9, x_{19}, x_{25}$ to be zero. At round 7 we have

$$\Delta_a l_{20}(k, x) = r_{22}$$

and we impose the first type 1 condition

$$r_{22} = x_{28} + x_{23} + x_{21} + k_6 = 0. \tag{1}$$

At round 9 we impose $x_3 = 0$. Then three additional type 1 conditions

$$r_{19} = x_{31} + x_{26} + x_{27} + x_{22} + k_0 = 1, \tag{2}$$

$$r_{23} = x_{27} + x_{22} + x_{23}x_{20} + x_{18} + x_7 + x_{12} + k_1 + k_8 = 0, \tag{3}$$

$$r_{26} = 1 + x_{20}(x_{17} + k_3) + k_{14} = 0 \tag{4}$$

are imposed at the rounds 11, 13 and 20.

The free bits for these conditions can be directly read from the equations. They are:

$$x_0, x_4, x_8, x_{11}, x_{13}, x_{14}, x_{15}, x_{16}, x_{29} \text{ and } x_{30}.$$

So far, for any valid plaintext we can derive a sample of 2^{10} valid plaintexts. Since, in this case, this is not enough to perform a significant frequency test, we try to obtain larger samples by better analysing the non-free bits. Looking at the equations (1) to (4), we note that the non-free bits $x_7, x_{12}, x_{18}, x_{21}, x_{22}, x_{26}, x_{27}, x_{28}$ and x_{31} only occur linearly. They can be freely chosen as long as they satisfy the system of linear equations

$$\begin{cases} x_{28} + x_{21} = A \\ x_{31} + x_{26} + x_{27} + x_{22} = B \\ x_{27} + x_{22} + x_{18} + x_7 + x_{12} = C \end{cases}$$

for constants A, B, C. This system has 2^6 different solutions that can be added to each valid plaintext. In total this gives a sample of size 2^{16} that we can generate from a valid plaintext. Since we imposed 9 type 0 conditions we are left with 2^5 different samples of plaintexts for a given key. The conditions are satisfied for at least one of these samples. On this sample the difference in bit 18 of

the ciphertext after 78 rounds (this is bit r_{78}) is strongly biased. We perfom a frequency test of $\Delta_a r_{78}(k, \cdot)$ on each of the 2^5 generated samples. At significance level $\alpha = 10^{-4}$ the frequency test fails on at least one of them with probability almost one, and if it fails, all four type 1 conditions are satisfied with probability almost one. This allows us to recover k_0, k_6, the relation $k_1 + k_8$ and either k_{14} (if $x_{20} = 0$) or the relation $k_3 + k_{14}$ with high probability. The complexity of this attack is 2^{22}.

4.3 Analysis of KATAN48 and KATAN64

All the three members of the KATAN family perform 254 rounds, they use the same LFSR and the algebraic structure of the non-linear update functions is the same. The differences between the KATANn ciphers are the block size n, the length of the NLFSRs, the tap positions for the non-linear update and the number of times the NLFSRs are updated per round.

For KATAN48 the NLFSRs have length 19 and 29 and each register is updated twice per round. We obtained our best result with a difference of weight four at the positions 1, 10, 19 and 28 in the plaintext block. Imposing four type 0 conditions and two type 1 conditions we are able to derive a sample of size 2^{31} from a valid plaintext. This allows us to recover the key bit k_{12} and the relation $k_1 + k_{14}$ after 70 rounds (this corresponds to 140 updates of the NLFSRs) with a complexity of 2^{34}.

For KATAN64 the NFLSRs have length 25 and 39 and each register is updated three times per round. We obtained our best result with a difference of weight three at the positions 0, 13 and 26. Imposing six type 0 conditions and two type 1 conditions we are able to derive a sample of size at least 2^{32} from a valid plaintext. This allows us to recover k_2 and $k_1 + k_6$ after 68 rounds (204 updates of the NLFSRs) with a complexity of 2^{35}

4.4 Analysis of the KTANTAN family

KTANTANn is very similar to KATANn. They only differ in the key scheduling part. In KATAN the key is loaded into a register and linearly expanded to the round keys after round 40. Until round 40 the original key bits are used as the round keys. In KTANTAN, from the first round, the round keys are a linear combination of key bits (depending on the state of the counter LFSR, which is entirely known). Hence, our analysis of KATANn directly translates to KTANTANn, but instead of recovering a single key bit, we recover a linear relation of key bits. For instance in KATAN32 we recover the relation $k_7 + k_{71}$ instead of bit k_0.

5 Analysis of Grain v1

Grain v1 is a stream cipher proposed in [13] and has been selected for the final eSTREAM portfolio [9]. It accepts an 80-bit key k and a 64-bit initial value x.

The cipher consists of three building blocks, namely an 80-bit LFSR, an 80-bit NLFSR and a non-linear output function. The state of the LFSR is denoted by $s_i, \ldots, s_{i+79}$ and the state of the NLFSR by $b_i, \ldots, b_{i+79}$. The registers are initialized by $b_i = k_i$ for $0 \leq i \leq 79$, $s_i = x_i$ for $0 \leq i \leq 63$ and $s_i = 1$ for $64 \leq i \leq 79$ and updated according to

$$\begin{aligned} s_{i+80} &= f(s_i, \ldots, s_{i+79}), \\ b_{i+80} &= g(b_i, \ldots, b_{i+79}) + s_i, \end{aligned}$$

where f is linear and g has degree 6. The output function is taken as

$$z_i = \sum_{k \in \mathcal{A}} b_{i+k} + h(s_{i+3}, s_{i+25}, s_{i+46}, s_{i+64}, b_{i+63}),$$

where $\mathcal{A} = \{1, 2, 4, 10, 31, 43, 56\}$ and h is defined as

$$\begin{aligned} h(s_{i+3}, s_{i+25}, s_{i+46}, s_{i+64}, b_{i+63}) &= s_{i+25} + b_{i+63} \\ &+ s_{i+3}s_{i+64} + s_{i+46}s_{i+64} + s_{i+64}b_{i+63} \\ &+ s_{i+3}s_{i+25}s_{i+46} + s_{i+3}s_{i+46}s_{i+64} + s_{i+3}s_{i+46}b_{i+63} \\ &+ s_{i+25}s_{i+46}b_{i+63} + s_{i+46}s_{i+64}b_{i+63} \end{aligned}$$

The cipher is clocked 160 times without producing any keystream. Instead the output function is fed back to the LFSR and to the NLFSR.

If we consider round-reduced variants of Grain v1 with r initialization rounds, the feedback of the output stops after r rounds and the first keystream bit is z_r.

Our analysis is similar to the one of KATAN32, but the equations for the conditions are more complex. We first present an attack on 97 rounds and then extend it to 104 rounds.

5.1 Distinguishing Attack and Key Recovery for 97 Rounds

Our analysis is based on the first order derivative with respect to a single difference in bit 37 of the initial value. Let $a = e_{37}$ denote the difference. The first conditions are defined at round 12, where the difference in s_{37} eventually propagates to the state bits s_{92} and b_{92} via the feedback of z_{12}. We have

$$\Delta_a z_{12}(k, x) = 1 + x_{15}x_{58} + x_{58}k_{75}.$$

We impose the type 0 condition $x_{58} = 1$ and we define the type 1 condition $x_{15} + k_{75} = 0$ to prevent the propagation. The next conditions are determined at round 34, where we have

$$\Delta_a z_{34}(k, x) = s_{98} + x_{59}s_{80} + s_{80}s_{98} + s_{80}b_{97}.$$

We define the conditions $s_{80} = 0$ and $s_{98} = 0$. Similarly we determine $s_{86} = 0$ and $s_{92} = 0$ at the rounds 40 and 46, respectively. So far, we imposed one type 0

condition at round 12 and we have five type 1 conditions at the rounds 12, 34, 40 and 46. The type 1 conditions jointly have 25 free bits:

$$x_7, x_8, x_{10}, x_{11}, x_{14}, x_{16}, x_{17}, x_{20}, x_{22}, x_{24}, x_{28}, x_{30}, x_{32}, x_{33},$$
$$x_{34}, x_{36}, x_{39}, x_{42}, x_{45}, x_{49}, x_{54}, x_{55}, x_{59}, x_{60} \text{ and } x_{61}.$$

In average we expect that one out of 2^5 randomly chosen initial values satisfies the conditions. We define a distinguisher that chooses 2^5 random initial values and for each performs a frequency test of $\Delta_a z_{97}(k, \cdot)$ on the sample of 2^{25} inputs generated by the free bits. Instead of randomly choosing 2^5 initial values we can choose 2^4 and test each of them for $x_{15} = 0$ and $x_{15} = 1$. This guarantees that the condition from round 12 is satisfied for at least one of them. Experiments with 2^{10} keys at a significance level $\alpha = 0.005$ show that at least one of the 2^5 tests fails with probability 0.99. This gives a distinguisher with complexity 2^{31} and advantage of about 0.83 for Grain v1 reduced to 97 rounds.

The two conditions $x_{15}+k_{75} = 0$ and $s_{86} = 0$ are crucial to obtain a significant bias after 97 rounds. In a key recovery scenario this reveals information about the key. Experiments show that both conditions hold with probability almost one if the frequency test fails. This recovers the key bit k_{75} and the value of $k_7 + k_8 + k_{10} + k_{37} + k_{49} + k_{62} + k_{69}$ (coming from $s_{86} = 0$).

5.2 Extension to 104 Rounds

Using the same conditions as before, we extend the attack to 104 rounds. We use the same idea as for KATAN32 to increase the size of the sample that can be generated from one initial value. We gain four additional degrees of freedom by noting that the non-free bits x_6, x_{19}, x_{29}, x_{44} and x_{57} influence only the condition imposed at round 40 and must only satisfy the linear equation

$$x_6 + x_{19} + x_{29} + x_{44} + x_{57} = A$$

for a constant A. In total, we can now derive a sample of size 2^{29} from one initial value.

The distinguisher defined above has now a complexity of 2^{35} and advantage of about 0.45. When the frequency test fails, the conditions $x_{15} + k_{75} = 0$ and $s_{92} = 0$ are satisfied with a probability almost one, which gives us k_{75} and the value of $k_{13} + k_{14} + k_{16} + k_{22} + k_{43} + k_{55} + k_{68}$ (coming from $s_{92} = 0$). The remaining three conditions are satisfied with a probability about 0.70 and give us similar relations in the key bits.

The sample size can be further increased, because also the non-free bits $x_{13}, x_{23}, x_{38}, x_{51}$ and x_{62} only must satisfy a linear equation. This gives a distinguisher with complexity 2^{39} and advantage of about 0.58.

6 Analysis of Grain-128

Grain-128 was proposed in [12] as a bigger version of Grain v1. It accepts a 128-bit key k and a 96-bit initial value x. The general construction of the cipher

is the same as for Grain v1, but the LFSR and the NLFSR both contain 128-bits. The content of the LFSR is denoted by $s_i, \ldots, s_{i+127}$ and the content of the NLFSR is denoted by $b_i, \ldots, b_{i+127}$. The initialization with the key and the initial value is analogous to Grain v1 and the update is performed according to

$$\begin{aligned} s_{i+128} &= f(s_i, \ldots, s_{i+127}), \\ b_{i+128} &= g(b_i, \ldots, b_{i+127}) + s_i, \end{aligned}$$

where f is linear and g has degree 2. The output function is taken as

$$z_i = \sum_{k \in \mathcal{A}} b_{i+k} + h(b_{i+12}, s_{i+8}, s_{i+13}, s_{i+20}, b_{i+95}, s_{i+42}, s_{i+60}, s_{i+79}, s_{i+95}),$$

where $\mathcal{A} = \{2, 15, 36, 45, 64, 73, 89\}$ and h is defined as

$$h(x) = b_{i+12}s_{i+8} + s_{i+13}s_{i+20} + b_{i+95}s_{i+42} + s_{i+60}s_{i+79} + b_{i+12}b_{i+95}s_{i+95}$$

The cipher is clocked 256 times without producing any keystream. Instead the output function is fed back to the LFSR and to the NLFSR.

If we consider round-reduced variants of Grain-128 with r initialization rounds, the feedback of the output stops after r rounds and the first keystream bit is z_r.

For the analysis of Grain-128 we use higher order derivatives. The general idea of conditional differential cryptanalysis naturally extends. As in the case of first order derivatives we always assume that the differences are applied to the initial value and not to the key.

6.1 Distinguishing Attack up to 215 Rounds

Our attack is based on a derivative of order thirteen with respect to the set of differences

$$\sigma = \{e_0, e_1, e_2, e_{34}, e_{35}, e_{36}, e_{37}, e_{65}, e_{66}, e_{67}, e_{68}, e_{69}, e_{95}\}.$$

These differences are chosen because they do not influence each other in the initial rounds. As a consequence the corresponding differential characteristic (of order thirteen) is zero for as many as 170 rounds. This can be extended to 190 rounds by imposing simple type 0 conditions that control the propagation of each single difference. As an example we derive the conditions for the difference e_{65}. The first condition is derived from round 5, where we have

$$\Delta_{e_{65}} z_5(k, x) = x_{84}.$$

We impose $x_{84} = 0$. In the same way the conditions $x_{58} = 0$ and $x_{72} = 0$ prevent difference propagation at rounds 45 and 52. At round 23 we have

$$\Delta_{e_{65}} z_{23}(k, x) = k_{118}.$$

As we will see below, the type 2 condition $k_{118} = 0$ determines a class of weak keys for the distinguishing attack.

Proceeding the same way for the other differences we derive 24 type 0 conditions that consist in setting the following bits to zero: x_{27}, x_{28}, x_{29}, x_{30}, x_{41}, x_{42}, x_{43}, x_{44}, x_{58}, x_{59}, x_{60}, x_{61}, x_{62}, x_{72}, x_{73}, x_{74}, x_{75}, x_{76}, x_{77}, x_{84}, x_{85}, x_{86}, x_{87}, x_{88}. In addition to k_{118} the key bits k_{39}, k_{119}, k_{120} and k_{122} can be identified to define classes of weak keys.

There are $2^{96-13-24} = 2^{59}$ initial values that are different in $\mathbb{F}_2^n/L(\sigma)$ and satisfy all type 0 conditions. We define a distinguisher that performs a frequency test of $\Delta_\sigma^{(13)} z_r(k, \cdot)$ on 2^{12} of these inputs. Table 1 summarizes the empirical results obtained for 2^{12} different keys tested at a significance level $\alpha = 0.005$. The indicated values denote the probabilty $1-\beta$, where β denotes the probability that $\Delta_\sigma^{(13)} z_r(k, \cdot)$ passes the frequency test. Our distinguisher has complexity 2^{25} and advantage $1-\alpha-\beta$. The values in the first row are obtained without any condition on the key. They show that we can distinguish Grain-128 reduced to 215 rounds with an advantage of about 0.008. The other rows indicate the probabilities for the classes of weak keys defined by the indicated type 2 conditions.

Table 1. Distinguishing attack on Grain-128 reduced to r rounds: Probability $1 - \beta$ for $\alpha = 0.005$ and complexity 2^{25}. Type 2 conditions define classes of weak keys.

type 2 condition	$r = 203$	$r = 207$	$r = 211$	$r = 213$	$r = 215$
–	1.000	0.587	0.117	0.173	0.013
$k_{39} = 0$	1.000	0.630	0.128	0.275	0.017
$k_{118} = 0$	1.000	0.653	0.177	0.231	0.024
$k_{119} = 0$	1.000	0.732	0.151	0.267	0.025
$k_{120} = 0$	1.000	0.876	0.234	0.249	0.026
$k_{122} = 0$	1.000	0.668	0.160	0.285	0.015

6.2 Key Recovery up to 213 Rounds

In this section we specifically exploit type 2 conditions to recover single key bits with high probability. The attack is explained by a prototypical example that recovers three bits of Grain-128 reduced to 197 rounds with a probability up to 0.87. It is based on a derivative of order five and can easily be extended to recover more bits by using slightly other derivatives. This is demonstrated by an attack that recovers eight bits using two additional derivatives (both of order five). A second attack uses the derivative of order thirteen from the previous section and recovers three bits for Grain-128 reduced to 213 rounds with a probability up to 0.59.

Prototypical Example. We use a derivative of order five with respect to the differences $\sigma = \{e_1, e_{36}, e_{66}, e_{67}, e_{68}\}$. In the same way as in the distinguishing attack, we impose conditions on the initial value to control the propagation of each difference. Altogether we impose 12 type 0 conditions and denote by W

the set of initial values satisfying all of them. The crucial observation is the following. The key bit k_{121} controls the characteristic of e_{68} in the very early phase of initialization, namely at round 26. If $k_{121} = 1$ the difference propagates, otherwise it does not. This strongly influences the frequency of $\Delta_\sigma^{(5)} z_r(k, \cdot)$ after $r = 197$ rounds. Similar strong influences can be found for k_{40} after $r = 199$ rounds and for k_{119} after $r = 200$ rounds. This allows to recover these bits by a binary hypothesis tests.

Key Recovery by Hypothesis Test. Let X be a uniformly distributed random variable taking values in $W/L(\sigma)$ and define

$$p_r(k) = \Pr[\Delta_\sigma^{(5)} z_r(k, X) = 1].$$

If the key is considered as a uniformly distributed random variable K, $p_r(K)$ is a random variable in the interval $[0, 1]$. Our attack is based on the observation that the conditional distributions of $p_r(K)$ conditioned on $K_i = 0$ and $K_i = 1$, for well chosen i, strongly differ even for a large number of rounds. This can be exploited to perform a binary hypothesis test on the value of K_i. An attacker can estimate a single observation $\hat{p}_r$ of $p_r(K)$ to take her decision. Since in all our attacks the expectation of $p_r(K)$ conditioned on $K_i = 0$ is significantly smaller than the conditional expectation conditioned on $K_i = 1$, we determine a parameter $\pi \in [0, 1]$ and take our decision according to the rule defined as

$$K_i = \begin{cases} 0 & \text{if } \hat{p}_r < \pi \\ 1 & \text{otherwise.} \end{cases}$$

The success probability of the attack essentially depends on the choice of π. If we denote $\alpha = \Pr[p_r(K) \geq \pi | K_i = 0]$ the probability that we falsely guess $K_i = 1$ and $\beta = \Pr[p_r(K) < \pi | K_i = 1]$ the corresponding probability that we falsely guess $K_i = 0$, then the *probability of a correct decision*, denoted P_c, is given as

$$\mathrm{P}_c = 1 - (\alpha + \beta)/2.$$

An optimal π maximizes P_c. Since the conditional distributions of $p_r(K)$ are not known explicitely, we empirically determine π in a precomputation phase of the attack.

Back to the Example. The first row of Table 2 shows the precomputed parameters π and the resulting probability P_c for our prototypical example. The precomputation of each π was done for 2^{14} key pairs and 2^{14} initial values for each key. This gives an overall precomputation complexity of $6 \cdot 2^{33}$ since we have to compute two histograms for each key bit. The attack itself consists in estimating $\hat{p}_r$ for $r = 197, 199$ and 200. Note that all three estimates can be obtained by the same computation which has complexity 2^{19} when estimating over 2^{14} initial values. The probabilities P_c are not completely independent and the probability of correctly guessing all three bits together is about 0.463.

Table 2. Key recovery for reduced Grain-128: P_c is the probability of correctly guessing key bit k_i. The attack complexity is 2^{19} for $|\sigma| = 5$ and 2^{25} for $|\sigma| = 13$.

Difference set	k_i	r	π	P_c
$\sigma = \{e_1, e_{36}, e_{66}, e_{67}, e_{68}\}$	k_{40}	199	0.494	0.801
	k_{119}	200	0.492	0.682
	k_{121}	197	0.486	0.867
$\sigma = \{e_0, e_1, e_2, e_{34}, e_{35}, e_{36}, e_{37},$	k_{39}	213	0.490	0.591
$e_{65}, e_{66}, e_{67}, e_{68}, e_{69}, e_{95}\}$	k_{72}	213	0.488	0.566
	k_{119}	206	0.356	0.830
	k_{120}	207	0.486	0.807
	k_{120}	211	0.484	0.592
	k_{122}	213	0.478	0.581

Recovering 8 Bits after 197 Rounds. The prototypical example can be extended by using two other sets of differences which are obtained by shifting all differences by one position to the left and to the right, respectively. This allows to recover five additional bits of the key, namely $k_{39}, k_{40}, k_{118}, k_{120}$ and k_{122}. The complexities of this extended attack are $9 \cdot 2^{34}$ for the precomputation and $3 \cdot 2^{19}$ for the attack itself. We recover all eight bits correctly with a probability of 0.123. This can be improved up to 0.236 by first determining k_{121} and k_{122} and then recovering the remaining bits conditioned on the values of k_{121} and k_{122}.

Recovering Bits up to 213 Rounds. If we use the derivative of order thirteen that we already used in the distinguishing attack, after 213 rounds we can recover two key bits with probability of almost 0.6. The last row of Table 2 summarizes the results. Here, the precomputation was done for 2^{12} key pairs and 2^{12} initial values for each key which gives a precomputation complexity of 2^{38}. The complexity of the attack itself is 2^{25}.

7 Conclusion

We presented a first analysis of the KATAN/KTANTAN family as well as the best known cryptanalytic results on Grain v1 and Grain-128. This was obtained by conditional differential cryptanalysis which also applies to other NLFSR-based contructions and provides further hints for choosing an appropriate number of rounds with regard to the security/efficiency tradeoff in future designs of such constructions.

Acknowledgements

This work was partially supported by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II.

References

1. Aumasson, J.P., Dinur, I., Henzen, L., Meier, W., Shamir, A.: Efficient FPGA Implementations of High-Dimensional Cube Testers on the Stream Cipher Grain-128. In: SHARCS (2009)
2. Aumasson, J.P., Dinur, I., Meier, W., Shamir, A.: Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. In: Dunkelman, O. (ed.) FSE. LNCS, vol. 5665, pp. 1–22. Springer (2009)
3. Biham, E., Dunkelman, O.: Differential Cryptanalysis in Stream Ciphers. Cryptology ePrint Archive, Report 2007/218 (2007), http://eprint.iacr.org/
4. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO. LNCS, vol. 537, pp. 2–21. Springer (1990)
5. Cannière, C.D.: Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In: Katsikas, S.K., Lopez, J., Backes, M., Gritzalis, S., Preneel, B. (eds.) ISC. LNCS, vol. 4176, pp. 171–186. Springer (2006)
6. Cannière, C.D., Dunkelman, O., Knezevic, M.: KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: Clavier, C., Gaj, K. (eds.) CHES. LNCS, vol. 5747, pp. 272–288. Springer (2009)
7. Cannière, C.D., Küçük, Ö., Preneel, B.: Analysis of Grain's Initialization Algorithm. In: Vaudenay, S. (ed.) AFRICACRYPT. LNCS, vol. 5023, pp. 276–289. Springer (2008)
8. Dinur, I., Shamir, A.: Cube Attacks on Tweakable Black Box Polynomials. In: Joux, A. (ed.) EUROCRYPT. LNCS, vol. 5479, pp. 278–299. Springer (2009)
9. ECRYPT: The eSTREAM project, http://www.ecrypt.eu.org/stream/
10. Englund, H., Johansson, T., Turan, M.S.: A Framework for Chosen IV Statistical Analysis of Stream Ciphers. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT. LNCS, vol. 4859, pp. 268–281. Springer (2007)
11. Fischer, S., Khazaei, S., Meier, W.: Chosen IV Statistical Analysis for Key Recovery Attacks on Stream Ciphers. In: Vaudenay, S. (ed.) AFRICACRYPT. LNCS, vol. 5023, pp. 236–245. Springer (2008)
12. Hell, M., Johansson, T., Maximov, A., Meier, W.: A Stream Cipher Proposal: Grain-128. In: ISIT. pp. 1614–1618 (2006)
13. Hell, M., Johansson, T., Meier, W.: Grain: A Stream Cipher for Constrained Environments. IJWMC 2(1), 86–93 (2007)
14. Khazaei, S., Meier, W.: New Directions in Cryptanalysis of Self-Synchronizing Stream Ciphers. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) INDOCRYPT. LNCS, vol. 5365, pp. 15–26. Springer (2008)
15. Knudsen, L.R.: Truncated and Higher Order Differentials. In: Preneel, B. (ed.) FSE. LNCS, vol. 1008, pp. 196–211. Springer (1994)
16. Lai, X.: Higher order derivatives and differential cryptanalysis. In: Blahut, R.E., Costello, D.J., Maurer, U., Mittelholzer, T. (eds.) Communicationis and Cryptography: Two Sides of one Tapestry. pp. 227–233. Kluwer Academic Publishers (1994)
17. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer, R. (ed.) EUROCRYPT. LNCS, vol. 3494, pp. 19–35. Springer (2005)
18. Wu, H., Preneel, B.: Resynchronization Attacks on WG and LEX. In: Robshaw, M.J.B. (ed.) FSE. LNCS, vol. 4047, pp. 422–432. Springer (2006)

Scrutinizing and Improving Impossible Differential Attacks: Applications to CLEFIA, Camellia, LBlock and Simon (Full Version)*

Christina Boura[1], María Naya-Plasencia[2], Valentin Suder[2]

[1] Versailles Saint-Quentin-en-Yvelines University, France
christina.boura@prism.uvsq.fr
[2] Inria, France
Maria.Naya_Plasencia@inria.fr, Valentin.Suder@inria.fr

Abstract. Impossible differential cryptanalysis has shown to be a very powerful form of cryptanalysis against block ciphers. These attacks, even if extensively used, remain not fully understood because of their high technicality. Indeed, numerous are the applications where mistakes have been discovered or where the attacks lack optimality. This paper aims in a first step at formalizing and improving this type of attacks and in a second step at applying our work to block ciphers based on the Feistel construction. In this context, we derive generic complexity analysis formulas for mounting such attacks and develop new ideas for optimizing impossible differential cryptanalysis. These ideas include for example the testing of parts of the internal state for reducing the number of involved key bits. We also develop in a more general way the concept of using multiple differential paths, an idea introduced before in a more restrained context. These advances lead to the improvement of previous attacks against well known ciphers such as CLEFIA-128 and Camellia, while also to new attacks against 23-round LBlock and all members of the Simon family.
Keywords. block ciphers, impossible differential attacks, CLEFIA, Camellia, LBlock, Simon.

1 Introduction

Impossible differential attacks were independently introduced by Knudsen [22] and Biham et al. [7]. Unlike differential attacks [8] that exploit differential paths of high probability, the aim of impossible differential cryptanalysis is to use differentials that have a probability of zero to occur in order to eliminate the key candidates leading to such impossible differentials.

The first step in an impossible differential attack is to find an impossible differential covering the maximum number of rounds. This is a procedure that has been extensively studied and there exist algorithms for finding such impossible differentials efficiently [21, 20, 12]. Once such a maximum-length impossible differential has been found and placed, one extends it by some rounds to both directions. After this, if a candidate key partially encrypts/decrypts a given pair to the impossible differential, then this key certainly cannot be the right one and is thus rejected. This technique provides a sieving of the key space and the remaining candidates can be tested by exhaustive search.

Despite the fact that impossible differential cryptanalysis has been extensively employed, the key sieving step of the attack does not seem yet fully understood. Indeed, this part of the procedure is highly technical and many parameters have to be taken into consideration. Questions that naturally arise concern the way to choose the plaintext/ciphertext pairs, the way to calculate the necessary data to mount the attack, the time complexity of the overall procedure as well as which are the parameters that optimize the attack. However, no simple and generalized way for answering these questions has been provided until now and the generality of most of the published attacks is lost within the tedious details of each application. The problems that arise from this approach is that mistakes become very common and attacks become difficult to verify. Errors in the analysis are often discovered and as we demonstrate in the next paragraph, many papers in the literature present flaws. These flaws include errors in the computation of the time or the data complexity, in the analysis of the memory requirements or of the complexity of some intermediate steps of the attacks. We can cite many such cases for different algorithms, as shown in Table 1. Note however, that the list of flaws presented in this table is not exhaustive.

*Partially supported by the French Agence Nationale de la Recherche through the BLOC project under Contract ANR-11-INS-011.

Algorithm	# rounds	Reference	Type of error	Gravity of error	Where discovered
CLEFIA-128 (without whit. layers)	14	[40]	data complexity higher than codebook	attack does not work	[32]
CLEFIA-128	13	[33]	cannot be verified without implementation	-	[10]
Camellia (without FL/FL^{-1} layers)	12	[38]	big flaw in computation as in [37]	attack does not work	this paper
Camellia-128	12	[37]	big flaw in computation	attack does not work	[26]
Camellia-128/192/256 (without FL/FL^{-1} layers)	11/13/14	[24]	small complexity flaws	corrected attacks work	[38]
LBlock	22	[27]	small complexity flaw	corrected attack works	[28]
SIMON (all versions)	14/15/15/16/16/ 19/19/22/22/22	[4]	data complexity higher than codebook	attacks do not work	Table 1 of [4]
SIMON (all versions)	13/15/17/20/25/	[1, 2]	big flaw in computation	attacks do not work	Appendix A.2

Table 1. Summary of flaws in previous impossible differential attacks on CLEFIA-128, Camellia, LBlock and SIMON.

Instances of such flaws can for example be found in analyses of the cipher CLEFIA. CLEFIA is a lightweight 128-bit block cipher developed by SONY in 2007 [29] and adopted as an international ISO/IEC 29192 standard in lightweight cryptography. This cipher has attracted the attention of many researchers and numerous attacks have been published so far on reduced round versions [34, 35, 33, 25, 31, 11]. Most of these attacks rely on impossible differential cryptanalysis. However, as pointed out by the designers of CLEFIA [30], some of these attacks seem to have flaws, especially in the key filtering phase. We can cite here a recent paper by Blondeau [10] that challenges the validity of the results in [33], or a claimed attack on 14 rounds of CLEFIA-128 [40], for which the designers of CLEFIA showed that the necessary data exceeds the whole codebook [32]. Another extensively analyzed cipher is the ISO/IEC 18033 standard Camellia, designed by Mitsubishi and NTT [5]. Among the numerous attacks presented against this cipher, some of the more successful ones rely on impossible differential cryptanalysis [38, 37, 23, 26, 24]. In the same way as for CLEFIA, some of these attacks were detected to have flaws. For instance, the attack from [37] was shown in [26] to be invalid. We discovered a similar error in the computation that invalidated the attack of [38]. Also, [38] reveals small flaws in [24]. Errors in impossible differential attacks were also detected for other ciphers. For example, in a cryptanalysis against the lightweight block cipher LBlock [27], the time complexity revealed to be incorrectly computed [28]. Another problem can be found in [4], where the data complexity is higher than the amount of data available in the block cipher SIMON, or in [1, 2], where some parameters are not correctly computed. During our analysis, we equally discovered problems in some attacks that do not seem to have been pointed out before. In addition to all this, the more the procedure becomes complicated, the more the approach lacks optimality. To illustrate this lack of optimality presented in many attacks we can mention a cryptanalysis against 22-round LBlock [19], that could easily be extended to 23 rounds if a more optimal approach had been used to evaluate the data and time complexities, as well as an analysis of Camellia [23] which we improve in Section 4.

The above examples clearly show that impossible differential attacks suffer from the lack of a unified and optimized approach. For this reason, the first aim of our paper is to provide a general framework for dealing with impossible differential attacks. In this direction, we provide new generic formulas for computing the data, time and memory complexities. These formulas take into account the different parameters that intervene into the attacks and provide a highly optimized way for mounting them. Furthermore, we present some new techniques that can be applied in order to reduce the data needed or to reduce the number of key bits that need to be guessed. In particular we present a new method that helps reducing the number of key bits to be guessed by testing instead some bits of the internal state during the sieving phase. This technique has some similarities with the methods introduced in [15, 17], however important differences exist as both techniques are applied in a completely different context. In addition to this, we apply and develop the idea of multiple impossible differentials, introduced in [35], to obtain more data for mounting our attacks. To illustrate the strength of our new approach we consider Feistel constructions and we apply the above ideas to a number of lightweight block ciphers, namely CLEFIA, Camellia, LBlock and SIMON.

More precisely, we present an attack as well as different time/data trade-offs on 13-round CLEFIA-128 that improve the time and data complexity of the previous best known attack [26] and improvements in the complexity of the best known attacks against all versions of Camellia [23]. In addition, in order to demonstrate the generality of our method, we provide the results of our attacks against 23-round LBlock and all versions of the SIMON block cipher. The attack on LBlock is the best attack so far in the single-key setting [3], while our attacks on SIMON are the best known impossible differential attacks for this family of ciphers and the best attacks in general for the three smaller versions of SIMON.

Summary of our attacks. We present here a summary of our results on the block ciphers CLEFIA-128, Camellia, LBlock and SIMON and compare them to the best impossible differential attacks known for the four analyzed algorithms. This summary is given in Table 2, where we point out with a '*' if the mentioned attack is the best cryptanalysis result on the target cipher or not, i.e. by the best known attack we consider any attack reaching the highest number of rounds, and with the best complexities among them.

Algorithm	# Rounds	Time	Data (CP)	Memory (Blocks)	Reference
CLEFIA-128	13	$2^{121.2}$	$2^{117.8}$	$2^{86.8}$	[25]
using state-test technique	13	$2^{116.90}$	$2^{116.33}$	$2^{83.33}$	Section 3
using multiple impossible differentials	13	$2^{122.26}$	$\mathbf{2^{111.02}}$	$\mathbf{2^{82.60}}$	Section 3*
combining with state-test technique	13	$\mathbf{2^{116.16}}$	$2^{114.58}$	$2^{83.16}$	Section 3*
Camellia-128	11	2^{122}	2^{122}	2^{98}	[23]
	11	$\mathbf{2^{118.43}}$	$\mathbf{2^{118.4}}$	$\mathbf{2^{92.4}}$	Section 4*
Camellia-192	12	$2^{187.2}$	2^{123}	$2^{155.41}$	[23]
	12	$\mathbf{2^{161.06}}$	$\mathbf{2^{119.7}}$	$\mathbf{2^{150.7}}$	Section 4*
Camellia-256	13	$2^{251.1}$	2^{123}	2^{203}	[23]
	13	$\mathbf{2^{225.06}}$	$\mathbf{2^{119.71}}$	$\mathbf{2^{198.71}}$	Section 4*
Camellia-256†	14	$2^{250.5}$	2^{120}	2^{120}	[23]
	14	$\mathbf{2^{220}}$	$\mathbf{2^{118}}$	2^{173}	Section 4
LBlock	22	$2^{79.28}$	2^{58}	$2^{72.67}$	[19]
	22	$\mathbf{2^{71.53}}$	2^{60}	$\mathbf{2^{59}}$	Appendix B,[13]
	23	$2^{75.36}$	2^{59}	2^{74}	Appendix B,[13]*
SIMON32/64	**19**	$2^{62.56}$	2^{32}	2^{44}	Appendix A*
SIMON48/72	**20**	$2^{70.69}$	2^{48}	2^{58}	Appendix A*
SIMON48/96	**21**	$2^{94.73}$	2^{48}	2^{70}	Appendix A*
SIMON64/96	**21**	$2^{94.56}$	2^{64}	2^{60}	Appendix A
SIMON64/128	**22**	$2^{126.56}$	2^{64}	2^{75}	Appendix A
SIMON96/96	**24**	$2^{94.62}$	2^{94}	2^{61}	Appendix A
SIMON96/144	**25**	$2^{190.56}$	2^{128}	2^{77}	Appendix A
SIMON128/128	**27**	$2^{126.6}$	2^{94}	2^{61}	Appendix A
SIMON128/192	**28**	$2^{190.56}$	2^{128}	2^{77}	Appendix A
SIMON128/256	**30**	$2^{254.68}$	2^{128}	2^{111}	Appendix A

Table 2. Summary of the best impossible differential attacks on CLEFIA-128, Camellia, LBlock and SIMON and presentation of our results. The presence of a '*' mentions if the current attack is the best known attack against the target cipher. Note here that we provide only the best of our results with respect to the time complexity. Other trade-offs can be found in the following sections. † see Section 4.1 for details.

The rest of the paper is organized as follows. In Section 2 we present a generic methodology for mounting impossible differential attacks, provide our complexity formulas and show new techniques and improvements for attacking a Feistel-like block cipher using impossible differential cryptanalysis. Section 3 is dedicated to the details of our attacks on CLEFIA and Section 4 presents our applications to all versions of Camellia. Finally, our attacks on the other ciphers can be found in Appendix A and B.

[3] In [14], an independent and simultaneous result on 23-round LBlock with worse time complexity was proposed.

2 Complexity analysis

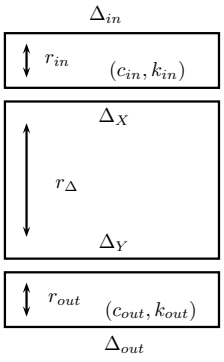
We provide in this section a comprehensive complexity analysis of impossible differential attacks against block ciphers as well as some new ideas that help improving the time and data complexities. We derive in this direction new generic formulas for the complexity evaluation of such attacks. The role of these formulas is twofold; on the one hand we aim at clarifying the attack procedure by rendering it as general as possible and on the other hand help at optimizing the time and data requirements. Establishing generic formulas should help mounting as well as verifying such attacks by avoiding the use of complicated procedures often leading to mistakes.

An impossible differential attack consists mainly of two general steps. The first one deals with the discovery of a maximum-length impossible differential, that is an input difference Δ_X and an output difference Δ_Y such that the probability that Δ_X propagates after a certain number of rounds, r_Δ, to Δ_Y is zero. The second step, called the key sieving phase, consists in the addition of some rounds to potentially both directions. These extra added rounds serve to verify which key candidates partially encrypt (resp. decrypt) data to the impossible differential. As this impossible differential is of probability zero, keys showing such behavior are clearly not the right encryption key and are thus removed from the candidate keys space.

We start by introducing the notation that will be used in the rest of the paper. As in this work we are principally interested in the key sieving phase, we start our attack after a maximum impossible differential has been found for the target cipher.

The differential $(\Delta_X \rightarrow \Delta_{in})$ (resp. $(\Delta_Y \rightarrow \Delta_{out})$) occurs with probability 1 while the differential $(\Delta_X \leftarrow \Delta_{in})$ (resp. $(\Delta_Y \leftarrow \Delta_{out})$) is verified with probability $\frac{1}{2^{c_{in}}}$ (resp. $\frac{1}{2^{c_{out}}}$), where c_{in} (resp. c_{out}) is the number of bit-conditions that have to be verified to obtain Δ_X from Δ_{in} (resp. Δ_Y from Δ_{out}).

It is important to correctly determine the number of key bits intervening during an attack. We call this quantity *information key bits*. In an impossible differential attack, one starts by determining all the subkey bits that are involved in the attack. We denote by k_{in} the subset of subkey bits involved in the attack during the first r_{in} rounds, and k_{out} during the last r_{out} ones. However, some of these subkey bits can be related between them. For example, two different subkey bits can actually be the same bit of the master key. Alternatively, a bit in the set can be some combination, or can be easily determined by some other bits of the set. The way that the different key bits in the target set are related is determined by the key schedule. The actual parameter that we need to determine for computing the complexity of the attacks is the information key bits intervening in total, that is from an information theoretical point of view, the log of the entropy of the involved key bits, that we denote by $|k_{in} \cup k_{out}|$.



- Δ_X, Δ_Y: input (resp. output) differences of the impossible differential.
- r_Δ: number of rounds of the impossible differential.
- Δ_{in}, Δ_{out}: set of all possible input (resp. output) differences of the cipher.
- r_{in}: number of rounds of the differential path(Δ_X, Δ_{in}).
- r_{out}: number of rounds of the differential path(Δ_Y, Δ_{out}).

We continue now by describing our attack scenario on $(r_{in} + r_\Delta + r_{out})$ rounds of a given cipher.

2.1 Attack scenario

Suppose that we are dealing with a block cipher of block size n parametrized by a key K of size $|K|$. Let the impossible differential be placed between the rounds $(r_{in} + 1)$ and $(r_{in} + r_\Delta)$. As already said, the impossible differential implies that it is not feasible that an input difference Δ_X at round $(r_{in} + 1)$

propagates to an output difference Δ_Y at the end of round $(r_{in} + r_\Delta)$. Thus, the goal is, for each given pair of inputs (and their corresponding outputs), to discard the keys that generate a difference Δ_X at the beginning of round $(r_{in} + 1)$ and at the same time, a difference Δ_Y at the output of round $(r_{in} + r_\Delta)$. We need then enough pairs so that the number of non-discarded keys is significantly lower than the a priori total number of key candidates.

Suppose that the first r_{in} rounds have an input truncated difference in Δ_{in} and an output difference Δ_X, which is the input of the impossible differential. Suppose that there are c_{in} bit-conditions that need to be verified so that Δ_{in} propagates to Δ_X and $|k_{in}|$ information key bits involved.

In a similar way, suppose that the last r_{out} rounds have a truncated output difference in Δ_{out} and an input difference Δ_Y, which is the output of the impossible differential. Suppose that there are c_{out} bit-conditions that need to be verified so that Δ_{out} propagates to Δ_Y in the backward direction and $|k_{out}|$ information key bits involved.

We show next how to determine the amount of data needed for an attack.

2.2 Data complexity

The probability that for a given key, a pair of inputs already satisfying the differences Δ_{in} and Δ_{out} verifies all the $(c_{in} + c_{out})$ bit-conditions is $2^{-(c_{in}+c_{out})}$. In other words, this is the probability that for a pair of inputs having a difference in Δ_{in} and an output difference in Δ_{out}, a key from the possible key set is discarded. Therefore, by repeating the procedure with N different input (or output) pairs, the probability that a trial key is kept in the candidate keys set is $P = (1 - 2^{-(c_{in}+c_{out})})^N$.

There is not a unique strategy for choosing the amount of input (or output) pairs N. This choice principally depends on the overall time complexity, which is influenced by N, and the induced data complexity. Different trade-offs are therefore possible. A popular strategy, generally used by default is to choose N such that only the right key is left after the sieving procedure. This amounts to choose P as

$$P = (1 - 2^{-(c_{in}+c_{out})})^N < \frac{1}{2^{|k_{in} \cup k_{out}|}}.$$

In this paper we adopt a different approach that can help reducing the number of pairs needed for the attack and offers better trade-offs between the data and time complexity. More precisely, we permit smaller values of N. By proceeding like this, we will be probably left with more than one key in our candidate keys set and we will need to proceed to an exhaustive search among the remaining candidates, but the total time complexity of the attack will probably be much lower. In practice, we will start considering values of N such that P is slightly smaller than $\frac{1}{2}$ so to reduce the exhaustive search by at least one bit. The smallest value of N, denoted by $N_{\min}$, verifying

$$P = (1 - 2^{-(c_{in}+c_{out})})^{N_{\min}} \simeq e^{-N_{\min} \times 2^{-(c_{in}+c_{out})}} < \frac{1}{2}$$

is approximately $N_{\min} = 2^{c_{in}+c_{out}}$. Then we have to choose $N \geq N_{\min}$.

We provide now a solution for determining the cost of obtaining N pairs such that their input difference belongs to Δ_{in} and their output difference belongs to Δ_{out}. To the best of our knowledge, this is the first generic solution to this problem.

Finding N pairs verifying a given truncated differential. Gilbert and Peyrin gave in [16] a solution to the so-called limited birthday problem that searches for a pair of inputs whose difference lies in an input space Δ_{in} and whose output (ciphertext) difference lies in an output space Δ_{out} (see Figure 1). According to this solution, prooved in [18], the cost for finding one such pair is given by

$$C_1 = \max \left\{ \min_{\Delta \in \{\Delta_{in}, \Delta_{out}\}} \left\{ \sqrt{2^{n+1-|\Delta|}} \right\}, 2^{n+1-(|\Delta_{in}|+|\Delta_{out}|)} \right\}.$$

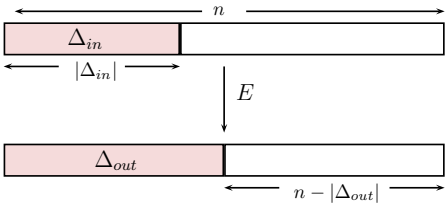


Fig. 1. A pair of inputs to the encryption function can differ in the subspace Δ_{in} and the difference of the ciphertext values can lie in the subspace Δ_{out}.

In our attack we search for a solution to a generalization of this problem, i.e. we want to determine the cost of finding not just one but N pairs lying in the given input and output spaces Δ_{in} and Δ_{out}. A direct way to treat this problem would be to estimate this cost by $N \times C_1$. However, this solution is not always optimal. In particular, as we will explain in a while, when the input space Δ_{in} is relatively large, the number of inputs that we will need, which determines the data complexity of the attack and at the same time the cost for constructing N pairs, can be lower than $N \times C_1$. We denote, the number of necessary inputs as C_N, as this quantity corresponds equally to the cost of constructing N pairs.

We can distinguish between two cases, depending on the dimension of the input space, $|\Delta_{in}|$, the dimension of the output space, $|\Delta_{out}|$ and the value of N. More precisely, the cost for constructing N pairs will depend on the value of

$$\frac{2^{|\Delta_{in}|}2^{|\Delta_{in}|-1}}{2^{n-|\Delta_{out}|}}$$

compared to N. The quantity $2^{|\Delta_{in}|}2^{|\Delta_{in}|-1}$ corresponds to the number of pairs that can be constructed if the values in Δ_{in} can take all possible values. On the other hand, the quantity $n - |\Delta_{out}|$ stands for the size of the partial collision, as we permit the output of these pairs to vary only in the subspace Δ_{out}.

- If $N \leq \frac{2^{|\Delta_{in}|}2^{|\Delta_{in}|-1}}{2^{n-|\Delta_{out}|}}$, this means that $|\Delta_{in}|$ is large enough to allow us to build C_N inputs belonging to the same structure ($C_N \leq 2^{|\Delta_{in}|}$). For the sake of clarity, we define a structure, as the set of inputs that can take all possible values in the subspace Δ_{in} and whose remaining $n - |\Delta_{in}|$ bits are fixed to a constant value. Therefore $N = \frac{C_N \cdot C_N/2}{2^{n-|\Delta_{out}|}}$, which means that we need $C_N = \sqrt{N2^{n-|\Delta_{out}|+1}}$ inputs.
- Otherwise, if $N > \frac{2^{|\Delta_{in}|}2^{|\Delta_{in}|-1}}{2^{n-|\Delta_{out}|}}$ which means that $|\Delta_{in}|$ is not large enough, we will need to consider several structures of size $2^{|\Delta_{in}|}$. Let 2^y be the number of these extra structures chosen in a way that $N = 2^y \frac{2^{|\Delta_{in}|}2^{|\Delta_{in}|-1}}{2^{n-|\Delta_{out}|}}$. The number of inputs is in this case given by:

$$C_N = 2^y 2^{|\Delta_{in}|} = N2^{n-|\Delta_{out}|-|\Delta_{in}|+1}.$$

By taking all of this into account together with the fact that we are dealing with a permutation (having thus a symmetry in both directions) and by considering the attacker to be able to choose the ciphertexts as well as the plaintexts, we can conclude that the cost of obtaining the N pairs will be:

$$C_N = \max\left\{\min_{\Delta\in\{\Delta_{in},\Delta_{out}\}}\left\{\sqrt{N2^{n+1-|\Delta|}}\right\}, N2^{n+1-|\Delta_{in}|-|\Delta_{out}|}\right\}. \quad (1)$$

We can observe thus that we gain a factor of $\sqrt{N}$ in the first of the two above cases compared to the trivial solution of taking $C_N = N \times C_1$. As we've already mentioned the cost C_N represents the amount of data needed for the attack. Obviously, as the size of the state is equal to n, the following inequality, should hold:

$$C_N \leq 2^n.$$

This inequality simply states that the total amount of data used for the attack cannot exceed the codebook. These conditions are not verified in several cases from [4], as well as in the corrected version of [40] which invalidates the corresponding attacks.

2.3 Time and memory complexity

We are going to detail now the computation of the time complexity of the attack. Note that the formulas that we are presenting in this section are the first generic formulas given for estimating the complexity of impossible differential attacks.

By following the early abort technique [24], the attack consists in storing the N pairs and testing out step by step the key candidates, by reducing at each time the size of the remaining possible pairs. The time complexity is then determined by three quantities. The first term is the cost C_N, that is the amount of needed data (see Formula (1)) for obtaining the N pairs, where N is such that $P < 1/2$. The second term corresponds to the number of candidate keys $2^{|k_{in} \cup k_{out}|}$, multiplied by the average cost of testing the remaining pairs. For all the applications that we have studied, this cost can be very closely approximated by $\left(N + 2^{|k_{in} \cup k_{out}|} \frac{N}{2^{c_{in}+c_{out}}}\right) C'_E$, where C'_E is the ratio of the cost of partial encryption to the full encryption. Finally, the third term is the cost of the exhaustive search for the key candidates still in the candidate keys set after the sieving. By taking into account the cost of one encryption C_E, we conclude that the time complexity of the attack is

$$T_{comp} = \left(C_N + \left(N + 2^{|k_{in} \cup k_{out}|} \frac{N}{2^{c_{in}+c_{out}}}\right) C'_E + 2^{|K|} P\right) C_E, \tag{2}$$

where $C_N = \max\left\{\min_{\Delta \in \{\Delta_{in}, \Delta_{out}\}} \left\{\sqrt{N 2^{n+1-|\Delta|}}\right\}, N 2^{n+1-|\Delta_{in}|-|\Delta_{out}|}\right\}$, with N such that $P = (1 - 1/(2^{c_{in}+c_{out}}))^N < 1/2$ and where the last term corresponds to $2^{|K|-|k_{in} \cup k_{out}|} P 2^{|k_{in} \cup k_{out}|}$. Obviously, as we want the attack complexity to be smaller than the exhaustive search complexity, the above quantity should be smaller than $2^{|K|} C_E$.

It must be noted here that this is a minimum estimation of the complexity, that, in practice, and thanks to the idea of Section 2.4, it approximates really well the actual time complexity, as it can be seen in the applications, and in particular, in the tight correspondance shown between the LBlock estimation that we detail in Appendix B and the exact calculation from [13]. The precise evaluation of C'_E (that is always smaller than 1) can only be done once the attack parameters are known. However, C'_E can be estimated quite easily by calculating the ratio between the active SBoxes during a partial encryption and the total number of SBoxes (thought it is not always the best approximation, it is a common practice).

Memory complexity. By using the early abort technique [24], the only elements that need to be stored are the N pairs. Therefore, the memory complexity [4] of the attack is determined by N.

2.4 Choosing $\Delta_{in}, \Delta_{out}, c_{in}$ and c_{out}.

We explain now, the two possible ways for choosing $\Delta_{in}, \Delta_{out}, c_{in}$ and c_{out}. For this, we introduce the following example that can be visualized in Figure 2 and where we consider an Sbox-based cipher. In this example, we will only talk about Δ_{in} and c_{in}, however the approach for Δ_{out} and c_{out} is identical.

Suppose that the state is composed of two branches of four nibbles each. The round function is composed of a non-linear layer S, seen as a concatenation of four Sboxes S_0, S_1, S_2 and S_3, followed by a linear layer M. There exist two different ways for choosing $|\Delta_{in}|$ and c_{in}:

1. The most intuitive way is to consider $|\Delta_{in}| = 4 + 4$ and $c_{in} = 4$, as the size of α and of β is 4 bits, and in the first round we want 4 bits to collide. In this case, for a certain key, the average probability that a pair taken out of the $2^{4+4}2^{4+4-1}$ pairs belonging to Δ_{in} leads to Δ_X is 2^{-4}.

[4] If $N > 2^{|k_{in} \cup k_{out}|}$ we could store the discarded key candidates instead, but in practice this is rarely the case. We can therefore consider a memory complexity of $\min\{N, 2^{|k_{in} \cup k_{out}|}\}$.

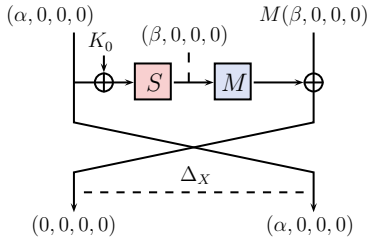


Fig. 2. Choosing Δ_{in} and c_{in}

2. In general, the difference α can take $2^4 - 1$ different values. However, each value can be associated by the differential distribution table of the Sbox S_0 to 2^3 output differences on average[5], so the possibilities for the difference β are limited to 2^3. Therefore, we can consider that $|\Delta_{in}| \approx 4 + 3$. But, in this case $c_{in} = 3$, as for each input pair belonging to the $2^{4+3}2^{4+3-1}$ possible ones, there exist on average 2 values that make the differential transition $\alpha \to \beta$ possible (instead of 1 in the previous case).

We can see, by using the generic formulas provided in Section 2.3, that both cases induce practically the same time complexity, as the difference in N compensates with the difference in $c_{in}+c_{out}$. However, the memory complexity, given by N, is slightly better in case 2. Furthermore, case 2, in which a preliminary filtering of the pairs is done, allows to reduce the average cost of using the early abort technique [24].

In several papers, for example in [37] and [24], the second case is followed. However, its application is partial (either for the input or the output part) and this with no apparent reason. Note however, that in these papers, the associated c_{out} was not always correctly computed and sometimes, 8-bit conditions were considered when 7-bit conditions should have been accounted for. For reasons of simplicity, we will consider case 1 in our applications and check afterwards the actual memory needed.

2.5 Using multiple impossible differentials to reduce the data complexity

We explain in this section a method to reduce the data complexity of an attack. This method is inspired by the notion of multiple impossible differentials that was introduced by Tsunoo et al. [35] and applied to 12-round CLEFIA-128. The idea in this technique is to consider simultaneously several impossible differentials, instead of taking just one. We assume, as done in [16], that the differences in Δ_{in} (and in Δout) lie in a closed set. We can mention two ways in which this can be a priori done:

1. Take rotated versions of a certain impossible differential. We call n_{in} the number of different input pattern differences generated by the rotated versions of the chosen impossible differential.
2. When the middle conditions have several impossible combinations, we can consider the same first half of the differential path together with a rotated version of the second one, in a way to get a different impossible differential. We call n_{out} the number of different output pattern differences generated by the rotated versions of the second part of the path that we will consider. For the sake of simplicity and without loss of generality we will only consider the case of rotating the second half of the path.

It is important to point out that for our analysis to be valid, in both cases the number of conditions associated to the impossible differential attack should stay the same. Both cases can be translated into a higher amount of available data by redefining two quantities, $|\Delta'_{in}|$ and $|\Delta'_{out}|$, that will take the previous roles of $|\Delta_{in}|$ and $|\Delta_{out}|$,

$$|\Delta'_{in}| = |\Delta_{in}| + \log_2(n_{in}) \text{ and } |\Delta'_{out}| = |\Delta_{out}| + \log_2(n_{out}).$$

$|\Delta'_{in}|$ is the log of the total size of the set of possible input differences, and $|\Delta'_{out}|$ is the log of the total size of the set of possible output differences.

[5] This quantity depends on the Sbox. In this example, we consider that all four Sboxes have good cryptographic properties.

In this case, the data complexity C_N is computed with the corrected values for the input sizes and is, as can be easily seen, smaller than if only one path had been used. The time complexity remains the same, except for the C_N term. Indeed, the middle term of Formula (2) remains the same, as for a given pair, the number of key bits involved stays $2^{|k_{in} \cup k_{out}|}$. Equally, as the number of partial possible keys involved in the attack is $n_{in} n_{out} 2^{|k_{in} \cup k_{out}|}$, the last term of Formula (2) is now

$$\frac{2^{|K|}}{n_{in} \cdot n_{out} 2^{|k_{in} \cup k_{out}|}} (P \cdot n_{in} \cdot n_{out} \cdot 2^{|k_{in} \cup k_{out}|}) = 2^{|K|} P$$

and so also stays the same.

In Section 3 we present our attacks on CLEFIA. In part of these attacks, we use multiple impossible differentials to reduce the data complexity. Besides, this technique shows particularly useful for mounting attacks on some versions of the SIMON family for which there is not enough available data to mount a valid attack with the traditional method.

2.6 Introducing the *state-test* technique

We introduce now a new method that consists in making a test for some part of the internal state instead of guessing the necessary key bits for computing it. This somewhat reminds the techniques presented in [15, 17] in the context of meet-in-the-middle attacks. However, the technique that we present in this section, and that we call the *state-test* technique is different since it consists in checking the values of the internal state to verify if we can discard all the involved candidates.

Very often during the key filtering phase of impossible differential attacks, the size of the internal state that needs to be known is smaller than the number of key bits on which it depends. As we will see, focusing on the values that a part of the state can take permits to eliminate some key candidates without considering all the values for the involved key bits. The state-test technique works by fixing s bits of the plaintexts, something which allows us to reduce the number of information key bits by s. We will explain how this method works by a small example.

Consider a 32-bit Feistel construction, where each branch can be seen as a concatenation of four nibbles (see Figure 3). Suppose that the round function is composed of a non-linear layer S, seen as a concatenation of four 4-bit invertible Sboxes (S_0, S_1, S_2, S_3) and of a linear layer M on $\mathbb{F}_{2^4}$. We suppose for this example that the branch number of M, that is the minimal number of active Sboxes in any two consecutive rounds, is less than 5. Let $\Delta_X = (\alpha, 0, 0, 0)|(0, 0, 0, 0)$ be the input difference of the impossible differential, placed at the end of the second round and let $\Delta_{in} = (*, *, *, 0)|(*, *, *, *)$ be the difference at the input of the block cipher. Note however that in reality, the leftmost side of Δ_{in} only depends on a 4-bit non-zero difference δ, i.e. $\Delta_{in} = M(\delta, 0, 0, 0)|(*, *, *, *)$.

As can be seen in Figure 3, there are in total 4 active Sboxes and thus there are $c_{in} = 16$ conditions that have to be verified in order to have a transition from Δ_{in} to Δ_X. Therefore, the first step is to collect N pairs such that $P = (1 - 2^{-(c_{in}+c_{out})})^N = (1 - 2^{-c_{in}})^N = (1 - 2^{-16})^N < \frac{1}{2}$. The exact value of N will be chosen in a way to obtain the best trade-off for the complexities. Before describing the new method, we start by explaining how this attack would have worked in the classical way. As we can see in Figure 3, there are 3×4 bits that have to be guessed ($K_{0,0}$, $K_{0,1}$ and $K_{0,2}$) in order to verify the conditions on the first round and there are 2×4 bits that have to be guessed ($K_{0,3}$ and $K_{1,0}$) in order to verify the conditions on the second round. Therefore, for all N pairs, one starts by testing all the 2^4 possible values for the first nibble of K_0. After this first guess, $N \times 2^{-4}$ pairs remain in average, as there are 4-bit conditions that need to be verified by the guess through the first round. Then one continues by testing the second and the third nibble of K_0 and finally the last nibble of K_0 and the first nibble of K_1. At each step, the amount of data remaining is divided by 2^4. To summarize, we have $|k_{in} \cup k_{out}| = |k_{in}| = 20$ and $2^{c_{in}+c_{out}} = 2^{c_{in}} = 2^4 2^4 2^4 2^4$. Then Formula (2) can be used to evaluate the time complexity of the attack as

$$\left(C_N + \left(N + 2^{20} \frac{N}{2^{16}} \right) C'_E + 2^{20} P 2^{|K|-20} \right) C_E. \tag{3}$$

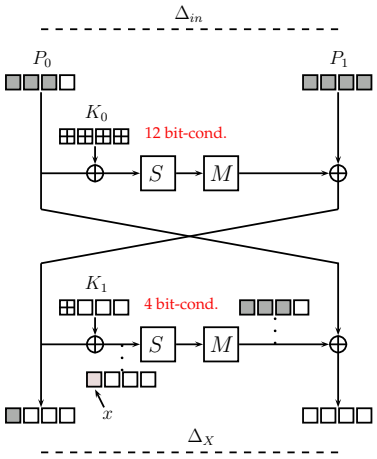


Fig. 3. Grey color stands for nibbles with non-zero difference. Hatched key nibbles correspond to the part of the subkeys that have to be guessed. The nibble x is the part of the state on which we apply the state-test technique.

We will see now how the state-test technique applies to this example and how it permits to decrease the time complexity. Consider the first nibble of the left part of the state after the addition of the subkey K_1. We denote this nibble by x. Note that mathematically, x can be expressed as

$$\begin{aligned} x &= K_{1,0} \oplus P_{1,0} \oplus M(S(K_0 \oplus P_0))_0 \\ x \oplus P_{1,0} &= K_{1,0} \oplus m_0 S_0(K_{0,0} \oplus P_{0,0}) \oplus m_1 S_1(K_{0,1} \oplus P_{0,1}) \oplus m_2 S_2(K_{0,2} \oplus P_{0,2}) \oplus m_3 S_3(K_{0,3} \oplus P_{0,3}), \end{aligned} \tag{4}$$

where the m_i's are coefficients in $\mathbb{F}_2^4$.

Suppose now that for all pairs, we fix the last $s = 4$ bits of P_0 to the same constant value. One can verify that this is a reasonable assumption, as by fixing this part of the inputs we still have enough data to mount the attack. Then one starts as before, by guessing the first three nibbles of K_0. After this 12-bit guess, approximately $N \times 2^{-12}$ pairs remain. We know for each pair the input and output differences of the Sbox of the second round as the needed part of K_0 has been guessed. Therefore, by a simple lookup at the differential distribution table of the involved Sbox, we obtain one value for x that verifies the second round conditions in average per pair (about half of the time the transition is not possible, whereas for the other half we find two values). Equation (4) becomes

$$x \oplus P_{1,0} \oplus m_0 S_0(K_{0,0} \oplus P_{0,0}) \oplus m_1 S_1(K_{0,1} \oplus P_{0,1}) \oplus m_2 S_2(K_{0,2} \oplus P_{0,2}) = K_{1,0} \oplus m_3 S_3(K_{0,3} \oplus P_{0,3}), \tag{5}$$

where the left side of Equation (5), that we denote by x', is known for each pair.

Thus, for each guess of $(K_{0,0}, K_{0,1}, K_{0,2})$, we construct a table of size $N \times 2^{-12}$, where we store these values of x'. The last and more important step consists now in looking if all 2^4 possible values of x' appear in the table. Note here, that as $N \geq 2^{16}$, the size of the table is necessarily greater than or equal to 2^4.

Since $P_{0,3}$ is fixed, the only unknown values in Equation (5) are $K_{1,0}$ and $K_{0,3}$. If all values for x' are in the table and since S_3 is a permutation, for any choice of $K_{1,0}$ and any choice of $K_{0,3}$, there will always exist (at least) one pair such that $K_{1,0} \oplus m_3 S_3(K_{0,3} \oplus P_{0,3})$ is in the table, leading thus to the impossible differential. As a conclusion, we know that if x' takes all the possible values in the table, we can remove the keys composed by the guessed value $(K_{0,0}, K_{0,1}, K_{0,3})$ from the candidate keys set, as for all the values of $(K_{1,0}, K_{0,3})$, they would imply the impossible differential. If instead, x' does not take all the possible values for a certain value of $(K_{0,0}, K_{0,1}, K_{0,3})$, we can test this partial key combined to all the possibilities of the remaining key bits that verify Equation (5) for the missing x', as they belong to the remaining key candidates.

The main gain of the state-test technique is that it decreases the number of information key bits and therefore the time complexity. For instance, in this example, the variable x' can be seen as 4 information

key bits [6] instead of 2×4 key bits we had to guess in the classic approach (the bits of $K_{0,3}$ and of $K_{1,0}$). We have $s = 4$ less bits to guess thanks to the $s = 4$ bits of the plaintext that we have fixed. Thus the time complexity in this case becomes

$$\left(C_N + \left(N + 2^{20-4}\frac{N}{2^{16}}\right)C'_E + 2^{20-4}P2^{|K|-(20-4)}\right)C_E. \tag{6}$$

One can see now by comparing Equations (6) and (3) that the time complexity is lower with the state-test technique, than with the trivial method. Indeed, the first and the third term of the Equations (6) and (3) remain the same, while the second term is lower in Equation (6). Finally, note that the probability P for a key to be still in the candidate keys set remains the same as before. Indeed, during the attack we detect all and the same candidate keys for which none of the N pairs implies the impossible differential, which are the same candidate keys that we would have detected in a classic attack.

We would like to note here that we have implemented the state-test technique on a toy cipher, having a structure similar to the one that we introduced in this section, and we have verified its correcteness.

Application of the state-test technique in parallel for decreasing the probability P. An issue that could appear with this technique is that as we have to fix a part of the plaintexts, s bits, the amount of data available for computing the N pairs is reduced. The probability P associated to an attack is the probability for a key to remain in the candidate keys set. When the amount of available data is small, the number of pairs N that we can construct is equally small and thus the probability P is high. In such a situation, the dominant term of the time complexity (Formula (2)), is in general the third one, i.e. $2^{|K|}P$.

More precisely, we need the sum of $\log_2(C_N)$ and s, the number of plaintext bits that we fix, to be less than or equal to the block size. This limits the size of N that we can consider, leading to higher probabilities P, and could lead, sometimes, to higher time complexities. To avoid this, one can repeat the attack in parallel for several different values, say Y, of the fixed part of the plaintext. In this case, the data and memory needed are multiplied by Y. On the other hand, repeating the attack in parallel permits to detect more efficiently if a guessed key could be the right one. Indeed, for a guessed key, only if none of the tables constructed as described above contains all the values for x', one can test if this guessed key is the correct one.

To summarize, by repeating the state-test technique in parallel, we multiply the available data by Y, as well as the available pairs, and since the attack is done Y times in parallel, the probability P becomes P^Y. The probability decreases much faster than the data or the other terms of the time complexity increase. Therefore, the Formula (2) becomes in this case:

$$\left(C_N \times Y + \left(N \times Y + 2^{|k_{in} \cup k_{out}|-s}\frac{N \times Y}{2^{c_{in}+c_{out}}}\right)C'_E + 2^{|K|}P^Y\right)C_E. \tag{7}$$

In Section 3, we are going to see an application of this technique to 13-round CLEFIA-128, and at the end of Section 4.1 we show an application on Camellia-256.

3 Application to CLEFIA

CLEFIA is a lightweight 128-bit block cipher designed by Shirai et al. in 2007 [29] and based on a 4-branch generalized Feistel network. We provide here a short description of the algorithm specifications. See [29] for a more complete description.

[6] Note that we could, equivalently, consider all possible values of x' in the last step, and consider the associated remaining pairs table, that would have a size of $N2^{-16}$ (empty if the key is a good candidate, not empty otherwise), obtaining the same key candidates of 16 bits, 12 from $(K_{0,0}, K_{0,1}, K_{0,3})$ and 4 information key bits from x', with the same complexity as in the previously described method.

3.1 Description of CLEFIA

Encryption algorithm. Denote by $P = P_0|P_1|P_2|P_3$ a 128-bit plaintext, where each P_i, $i = 0, 1, 2, 3$, is a 32-bit vector. Denote by C be the corresponding ciphertext. CLEFIA supports keys of size 128, 192 or 256 bits and the total number of iterations, say R, depends on the key size. More precisely, $R = 18$ for the 128-bit version, while $R = 22$ and $R = 26$ for the two following variants. A key-scheduling algorithm, whose description we omit, is used to generate $2R$ round keys $RK_0, \ldots, RK_{2R-1}$ and 4 whitening keys $WK_0, \ldots, WK_3$. The encryption is performed as follows:

- $P_0^0|P_1^0|P_2^0|P_3^0 = P_0|P_1 \oplus WK_0|P_2|P_3 \oplus WK_1$
- For $i = 1, 2, \ldots, R$ do
  - $P_0^i = F_0(P_0^{i-1}, RK_{2i-2}) \oplus P_1^{i-1}$
  - $P_1^i = P_2^{i-1}$
  - $P_2^i = F_1(P_2^{i-1}, RK_{2i-1}) \oplus P_3^{i-1}$
  - $P_3^i = P_0^{i-1}$
- $C = P_0^R|P_1^R \oplus WK_2|P_2^R|P_3^R \oplus WK_3$.

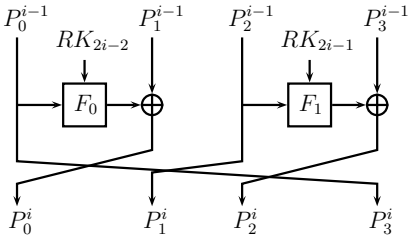


Fig. 4. A round of CLEFIA.

Round functions F_0, F_1. Each round of CLEFIA is composed of two 32-bit round functions F_0 and F_1 (see Figure 4) that have the same structure. The first step in the function F_0 (resp. F_1) is an XOR between the 32-bit subkey RK_{2i-2} (resp. RK_{2i-1}) and P_0^{i-1} (resp. P_2^{i-1}). Then, two 8×8-bit SBoxes S_0 and S_1 compose a layer which is applied to the result. Finally, the four obtained bytes are mixed by a 4×4-byte matrix, M_0 (resp. M_1) that has a maximal *branch number*, i.e. 5. A detailed description of the SBoxes S_0, S_1 while also the matrices M_0, M_1 can be found in [29].

We are going to describe now an impossible differential cryptanalysis of 13-round CLEFIA-128.

3.2 Impossible Differential Cryptanalysis of 13-round CLEFIA-128

The authors of [34] noticed that a difference on the internal state of CLEFIA of the form $P^i = 0_{32}|0_{32}|0_{32}|A$ cannot lead to a difference $P^{i+9} = 0_{32}|0_{32}|B|0_{32}$ after 9 rounds, where A and B are 4-byte vectors for which only one byte in a different position is active (e.g. $A = (\alpha, 0_8, 0_8, 0_8)$ and $B = (0_8, \beta, 0_8, 0_8)$). We use this same 9-round impossible differential and place it between rounds 3 and 11. Therefore, for our attack, $r_{in} = r_{out} = 2$ and $r_\Delta = 9$, as in [25].

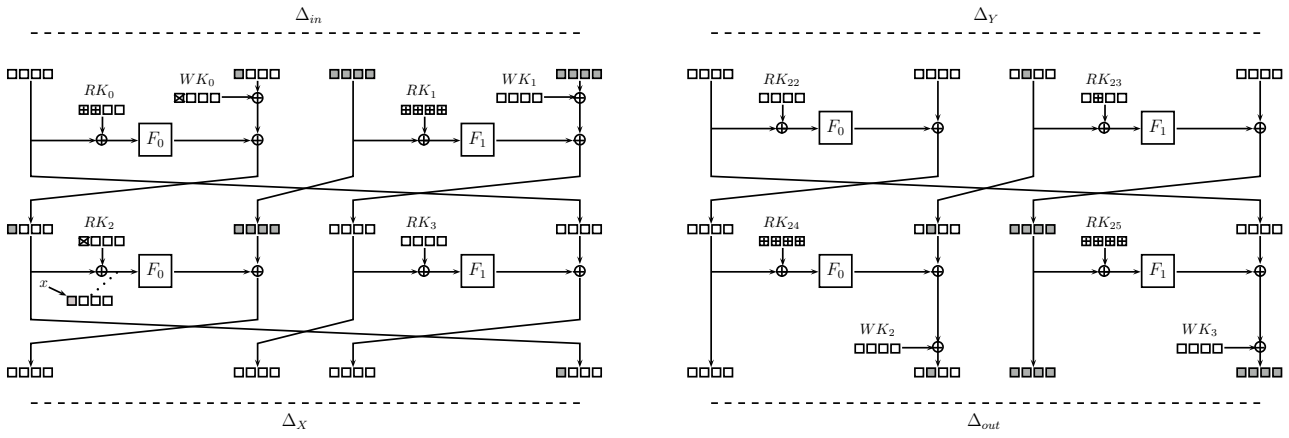


Fig. 5. The attack on CLEFIA-128. Grey color stands for bytes with a non-zero difference, while hatched bytes are the subkey bytes that have to be guessed.

The differential placed on the top and at the bottom of the impossible differential are depicted in Figure 5. We describe now the parameters for our cryptanalysis of 13-round CLEFIA-128. As can be seen

in Figure 5 there are $c_{in} + c_{out} = 40 + 40$ bit-conditions that need to be verified so that the difference in the plaintexts $\Delta_{in} = 0_{32}|(*_8, 0_8, 0_8, 0_8)|M_0(*_8, 0_8, 0_8, 0_8)|*_{32}$ propagates to $\Delta_X = 0_{32}|0_{32}|0_{32}|(\alpha, 0_8, 0_8, 0_8)$ and the difference in the ciphertexts $\Delta_{out} = 0_{32}|(0_8, *_8, 0_8, 0_8)|M_1(0_8, *_8, 0_8, 0_8)|*_{32}$ propagates to $\Delta_Y = 0_{32}|0_{32}|(0_8, \beta, 0_8, 0_8)|0_{32}$. In this way, $|\Delta_{in}| = |\Delta_{out}| = 48$.

Following the complexity analysis of Section 2, we need to construct at least $N_{\min} = 2^{80}$ pairs whose input difference is in Δ_{in} and whose output difference is in Δ_{out}. The cost to construct these pairs is

$$C_{N_{\min}} = \max\left\{\sqrt{2^{80}2^{129-48}}, 2^{80}2^{129-48-48}\right\} = 2^{113}.$$

Using the state-test technique. We use now the state-test technique, described in Section 2.6 to test the 8 bits of the internal state denoted by x in Figure 5, instead of guessing the whole subkey RK_0 and the XOR of the leftmost byte of RK_2 and WK_0. For doing this, we need to fix part of the 32 leftmost bits of the plaintexts. As the number of needed data is $C_{N_{\min}} = 2^{113}$, we can fix at most $128 - 113 = 15$ bits. However, as each Sbox is applied to 8 bits, we will only fix one byte of this part of the plaintexts. We will guess then 24 bits of the subkey RK_0 which are situated on the other bytes.

During a classical attack procedure, we would need to guess 32 bits of RK_1, 32 bits of RK_0 and 8 bits of $RK_2 \oplus WK_0$, thus $k_{in} = 72$. We would also need to guess 8 bits of $RK_{23} \oplus WK_2$, 32 bits of RK_{24} and 32 bits of RK_{25}, therefore $k_{out} = 72$. However, the subkeys RK_1 and RK_{24} share 22 bits in common. As a consequence, the number of information key bits would be $|k_{in} \cup k_{out}| = 72 + 72 - 22 = 122$. As we will fix 8 bits of the plaintexts, with respect to Section 2 on the state-test technique, it is the same to say that there will be $|k_{in} \cup k_{out}| - 8 = 122 - 8 = 114$ bits to test. The time complexity of our attack, computed using Formula (2) is then

$$\left(C_N + \left(N + 2^{114}\frac{N}{2^{80}}\right)\frac{18}{104} + 2^{128}P\right)C_E,$$

where the fraction 18/104 is the ratio of the cost of partial encryption to the full encryption, that we noted C'_E. Since our attack needs at least 2^{113} plaintexts and since we fixed 8 bits out of them, we have $128 - 113 - 8 = 7$ bits of freedom for building structures. Among all the possible trade-offs with respect to the amount of data, the best time complexity we obtained is $2^{116.90}C_E$ with $2^{83.33}$ pairs built from $2^{116.33}$ plaintexts.

Using multiple impossible differentials. The authors of [34] noticed that there exist several different 9-round impossible differentials, see [34, Table 1]. In [35], the authors used these multiple impossible differentials to attack 12 rounds of CLEFIA-128. Here, we will apply our formalized approach of this idea presented in Section 2.5, to reduce the data complexity of the attack on 13 rounds of CLEFIA-128.

We use the $n_{in} = 2 \times 4$ different inputs to the impossible differentials, that is $P^i = 0_{32}|A|0_{32}|0_{32}$ and $P^i = 0_{32}|0_{32}|0_{32}|A$, where A can take a difference on only one of the four possible bytes. For each one of them, there are $n_{out} = 3$ different output impossible differences $P^{i+9} = 0_{32}|0_{32}|B|0_{32}$ after 9 rounds, where B has only one byte active in a different position than the active byte in A. We have now $|\Delta'_{in}| = |\Delta_{in}| + \log_2(8) = 48 + 3$ and $|\Delta'_{out}| = |\Delta_{out}| + \log_2(3) = 48 + 1.58$. Since the bit-conditions remain unchanged, $c_{in} + c_{out} = 80$, the minimal number of pairs needed for the attack to work is $N_{\min} = 2^{80}$. For this number of pairs, we need $C_{N_{\min}} = 2^{113-4.58} = 2^{108.42}$ plaintexts. The number of information key bits is $|k_{in} \cup k_{out}| = 122$. We have then $\left(C_N + \left(N + 2^{122}\frac{N}{2^{80}}\right)\frac{18}{104} + P2^{128}\right)C_E$. Among all the possible trade-offs with respect to the amount of data, the best time complexity we obtained is $2^{122.26}C_E$ with $2^{82.6}$ pairs built from $2^{111.02}$ plaintexts. Recall here that the aim of this approach was to reduce data complexity. Thus, in this attack the gain on the data complexity is the important part[7].

Combining the state-test technique with multiple impossible differentials. We can combine now both previous approaches in order to reduce at the same time the time and the data complexity.

We consider here only 2 out of the 3 different n_{out} presented in the previous paragraph for one fixed first half of the impossible differential. We have now $|\Delta'_{out}| = |\Delta_{out}| + \log_2(2) = 48 + 1$ while $|\Delta_{in}|$ remains

[7] In [25], the authors used a loose approximation for a partial encryption as $C'_E = 1/104$.

48. Since the bit-conditions remain unchanged, i.e. $c_{in} + c_{out} = 80$, the minimal number of pairs needed for the attack to work is $N_{\min} = 2^{80}$. For this number of pairs, we need $C_{N_{\min}} = 2^{113-1} = 2^{112}$ plaintexts which allow us to fix 16 bits on the plaintexts in order to use the state-test technique. Fixing 16 bits on the plaintexts means that we only have to guess 16 bits of the subkey RK_0.

As we are fixing $s = 16$ bits of the plaintexts, the number of information key bits is then $|k_{in} \cup k_{out}| = 122 - 16 = 106$. Then, by combining multiple impossible differentials and the state-test technique we have

$$\left(C_N + \left(N \times 2^7 + 2^{106}\frac{N \times 2^7}{2^{80}}\right)\frac{18}{104} + P2^{128}\right)C_E,$$

where the second term is multiplied by 2^7, which is the cost for checking the table combinations of the different output impossible differentials. If we consider $N = 2^{80+3.16}$ pairs, we need $C_N = 2^{115}$ plaintexts to construct them and thus the time complexity is $2^{116.16}C_E$.

4 Applications to Camellia

Camellia is a 128-bit block cipher designed by Aoki et. al. in 2000 [5]. It is a Feistel-like construction where two key-dependent layers FL and FL^{-1} are applied every 6 rounds to each branch. There exist three different versions of the cipher, that we note Camellia-128, Camellia-192 and Camellia-256, depending on the key size used. The number of iterations, say R, is $R = 18$ for the 128-bit version and $R = 24$ rounds for the other two versions. We give here a brief description of the algorithm specifications. For more details, one can refer to [5].

4.1 Description of Camellia.

We briefly describe the encryption process for Camellia. A 128-bit plaintext P is first XORed with the prewhitening key $kw_1|kw_2$. The encryption process, is as follows:

- $L_0|R_0 = P \oplus (kw_1|kw_2)$
- For $i = 1, 2, \ldots, R$ and $i \neq 6, 12, 18$ do
  - $L_r = R_{r-1} \oplus F(L_{r-1}, k_r) \quad , R_r = L_{R-1}$
- For $i = 6, 12$ and 18 do
  - $L'_r = R_{r-1} \oplus F(L_{R-1}, k_r), \quad R'_r = L_{r-1};$
  - $L_r = FL(L'_r, kl_{r/3-1}), R_r = FL^{-1}(L'_r, kl_{r/3}).$
- $C = (R_{24}|L_{24}) \oplus kw_3|kw_4,$

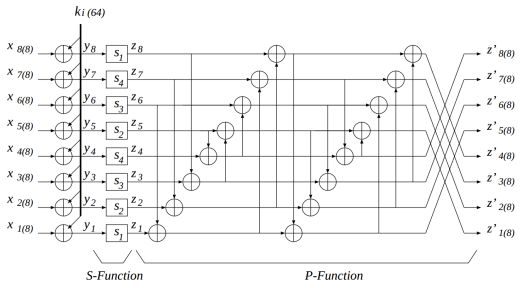


Fig. 6. Round function of Camellia.

where kw_3 and kw_4 are the two postwhitening keys, kl_r are the keys parametrizing the FL-layers and k_r are the round subkeys. The round function F can be visualized in Figure 6.

Previous Cryptanalysis. Camellia is since 2005 an international ISO/IEC standard and has therefore attracted a lot of attention from the cryptographic community. Since Camellia has a particular design, involving the so-called FL/FL^{-1} layers, its cryptanalysis can be classified in several categories. Some attacks consider the FL/FL^{-1} functions, while others do not take them into consideration. Equally, some attacks take into account the whitening keys, whereas others don't and finally all attacks do not start from the same round. The best attacks on Camellia in terms of the number of rounds and the complexities are those presented in [23, Section 4.2]. In this section we first present improvements of the best attacks that include the FL/FL^{-1} layers and the whitening keys. Next we build an attack using the state-test technique on 14-round Camellia-256 starting from the first round but without the FL/FL^{-1} layers and the whitening keys.

Improvements. We improve here the complexities of the previous attacks that take into account the FL/FL^{-1} layers and the whitening keys on all three versions of Camellia. By using the complexity analysis introduced in Section 2, we can optimize the complexities of the corresponding attacks from [23]. Note that we use for this the same parameters as in [23]. The parameters of our attacks on 11-round Camellia-128, 12-round Camellia-192 and 13-round Camellia-256 are depicted in Table 3. As can be seen in Table 2, the time complexity of our improved attack on Camellia-128 is $2^{118.43}C_E$, with data complexity $2^{118.4}$ and memory complexity $2^{92.4}$. For Camellia-192, the time, data and memory complexities are $2^{161.06}C_E$, $2^{119.7}$ and $2^{150.7}$ respectively, while for Camellia-256 the corresponding complexities are $2^{225.06}C_E$, $2^{119.71}$ and $2^{198.71}$.

Algorithm	$\|\Delta_{in}\|$	$\|\Delta_{out}\|$	r_{in}	r_{out}	r_{Δ}	c_{in}	c_{out}	$\|k_{in} \cup k_{out}\|$
Camellia-128	23	80	1	2	8	32	57	96
Camellia-192	80	80	2	2	8	73	73	160
Camellia-256	80	128	2	3	8	73	121	224

Table 3. Attack parameters against all versions of Camellia

Using the state-test technique on Camellia-256. We provide here an impossible differential attack on Camellia-256 without FL/FL^{-1} layers and without whitening keys by using the state-test technique. Note here, that unlike all previous attacks of this kind that do not start from the first round in order to take advantage of the key schedule asymmetry, our attack starts from the first round of the cipher. This attack covers 14 rounds of Camellia-256 which is, based on our knowledge, the highest number of rounds attacked for this version of the cipher. In [23] another attack on 14 rounds of Camellia-256 without FL/FL^{-1} and whitening keys is presented, however, as said before, it does not start from the first round, and it equally uses a specific property of the key schedule in the rounds where it is applied.

In this attack, we consider the same 8-round impossible differential as in [26] and we add 4+2 rounds such that $r_{in} = 4$, $r_{out} = 2$ and $r_{\Delta} = 8$. We have $|\Delta_{in}| = 128$, $|\Delta_{out}| = 56$, $c_{in} = 120$ and $c_{out} = 48$. Then we need at least $N_{\min} = 2^{168}$ plaintext pairs for our attack to work. The amount of data needed to construct these pairs is $C_{N_{\min}} = \max\left\{\sqrt{2^{168}2^{129-128}}, 2^{168}2^{129-184}\right\} = 2^{113}$. There remain then $128 - 113 = 15$ bits of freedom. Thus, we can fix $s = 8$ bits on the ciphertexts to apply the state-test technique on the 8 bits of the internal state at the penultimate round. The number of information key bits is $|k_{in} \cup k_{out}| = 227 - 8 = 219$ since there are 45 bits shared between the subkeys with respect to the key schedule of Camellia-256. The best attack is obtained with $N = 2^{118}$ pairs. In this case, the time complexity is $2^{220}C_E$, the data complexity is 2^{118} plaintexts and the memory is 2^{118}.

5 Conclusion

To start with, we have proposed in this paper a generic vision of impossible differential attacks with the aim of simplifying and helping the construction and verification of this type of cryptanalysis. Until now, these attacks were very tedious to mount and even more to verify, and so, very often flaws appeared in the computations. We believe that our objective has been succesfully reached, as it can be seen by the high amount of new improved attacks that we have been able to propose, as well as by all the different possible trade-offs for each one of them, something that would be near to unthinkable prior to our work.

Next, the generic and clear vision of impossible differental attacks has allowed us to discover and propose new ideas for improving these attacks. In particular, we have proposed the *state-test* technique, that allows to reduce the number of key bits involved in the attack, and so to reduce the time complexity. We have also formalized and adapted to our generic scenario the notion introduced in [35] of multiple impossible differentials. This option allows reducing the data complexity. Finally, we have proposed several applications to different variants of the Feistel block ciphers CLEFIA, Camellia, LBlock and Simon, providing in most of the cases, the best known attack on reduced-round versions of these ciphers.

We hope that these results will simplify and improve future impossible attacks on Feistel ciphers, as well as their possible combination with other attacks. For instance, in [39] a combination of impossible differential with linear attacks is proposed. We have not been able to verify these results, but this direction could be promising.

References

1. F. Abed, E. List, S. Lucks, and J. Wenzel. Differential and linear cryptanalysis of reduced-round SIMON. Cryptology ePrint Archive, Report 2013/526, 2013.
2. F. Abed, E. List, J. Wenzel, and S. Lucks. Differential Cryptanalysis of round-reduced Simon and Speck. In *FSE 2014*, LNCS. Springer, 2014. To appear.
3. J. Alizadeh, N. Bagheri, P. Gauravaram, A. Kumar, and S. K. Sanadhya. Linear Cryptanalysis of Round Reduced Variants of SIMON. Cryptology ePrint Archive, Report 2013/663, 2013.
4. H. A. Alkhzaimi and M. M. Lauridsen. Cryptanalysis of the SIMON Family of Block Ciphers. Cryptology ePrint Archive, Report 2013/543, 2013.
5. K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita. Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis. In *Selected Areas in Cryptography - SAC 2000*, volume 2012 of *LNCS*, pages 39–56. Springer, 2000.
6. R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. The SIMON and SPECK Families of Lightweight Block Ciphers. Cryptology ePrint Archive, Report 2013/404, 2013.
7. E. Biham, A. Biryukov, and A. Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In *EUROCRYPT 1999*, volume 1592 of *LNCS*, pages 12–23. Springer, 1999.
8. E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In *CRYPTO 1990*, volume 537 of *LNCS*, pages 2–21. Springer, 1990.
9. A. Biryukov, A. Roy, and V. Velichkov. Differential Analysis of Block Ciphers SIMON and SPECK. In *FSE 2014*, LNCS. Springer, 2014. To appear.
10. C. Blondeau. Improbable Differential from Impossible Differential: On the Validity of the Model. In *INDOCRYPT*, volume 8250 of *LNCS*, pages 149–160. Springer, 2013.
11. A. Bogdanov, H. Geng, M. Wang, L. Wen, and B. Collard. Zero-Correlation Linear Cryptanalysis with FFT and Improved Attacks on ISO Standards Camellia and CLEFIA. In *Selected Areas in Cryptography - SAC 2013*, volume 8282 of *LNCS*, pages 306–323. Springer, 2013.
12. C. Bouillaguet, O. Dunkelman, P-A. Fouque, and G. Leurent. New Insights on Impossible Differential Cryptanalysis. In *Selected Areas in Cryptography-SAC 2011*, volume 7118 of *LNCS*, pages 243–259. Springer, 2011.
13. C. Boura, M. Minier, M. Naya-Plasencia, and V. Suder. Improved Impossible Differential Attacks against Round-Reduced LBlock. Cryptology ePrint Archive, Report 2014/279, 2014.
14. J. Chen, Y. Futa, A. Miyaji, and C. Su. Impossible differential cryptanalysis of LBlock with concrete investigation of key scheduling algorithm. Cryptology ePrint Archive, Report 2014/272, 2014.
15. O. Dunkelman, G. Sekar, and B. Preneel. Improved Meet-in-the-Middle Attacks on Reduced-Round DES. In *INDOCRYPT*, volume 4859 of *LNCS*, pages 86–100. Springer, 2007.
16. H. Gilbert and T. Peyrin. Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations. In *FSE 2010*, volume 6147 of *LNCS*, pages 365–383. Springer, 2010.
17. T. Isobe and K. Shibutani. Generic Key Recovery Attack on Feistel Scheme. In *ASIACRYPT (1)*, volume 8269 of *LNCS*, pages 464–485. Springer, 2013.
18. M. Iwamoto, T. Peyrin, and Y. Sasaki. Limited-Birthday Distinguishers for Hash Functions - Collisions beyond the Birthday Bound Can Be Meaningful. In *ASIACRYPT (2)*, volume 8269 of *LNCS*, pages 504–523. Springer, 2013.
19. F. Karakoç, H. Demirci, and A. E. Harmanci. Impossible Differential Cryptanalysis of Reduced-Round LBlock. In *WISTP 2012*, volume 7322 of *LNCS*, pages 179–188. Springer, 2012.
20. J. Kim, S. Hong, and J. Lim. Impossible differential cryptanalysis using matrix method. *Discrete Mathematics*, 310(5):988–1002, 2010.
21. J. Kim, S. Hong, J. Sung, C. Lee, and S. Lee. Impossible Differential Cryptanalysis for Block Cipher Structures. In *INDOCRYPT 2003*, volume 2904 of *LNCS*, pages 82–96. Springer, 2003.
22. L. R. Knudsen. DEAL – A 128-bit cipher. Technical Report, Department of Informatics, University of Bergen, Norway, 1998.
23. Y. Liu, L. Li, D. Gu, X. Wang, Z. Liu, J. Chen, and W. Li. New Observations on Impossible Differential Cryptanalysis of Reduced-Round Camellia. In *FSE 2012*, volume 7549 of *LNCS*, pages 90–109. Springer, 2012.
24. J. Lu, J. Kim, N. Keller, and O. Dunkelman. Improving the Efficiency of Impossible Differential Cryptanalysis of Reduced Camellia and MISTY1. In *CT-RSA*, volume 4964 of *LNCS*, pages 370–386. Springer, 2008.
25. H. Mala, M. Dakhilalian, and M. Shakiba. Impossible Differential Attacks on 13-Round CLEFIA-128. *J. Comput. Sci. Technol.*, 26(4):744–750, 2011.
26. H. Mala, M. Shakiba, M. Dakhilalian, and G. Bagherikaram. New Results on Impossible Differential Cryptanalysis of Reduced-Round Camellia-128. In *Selected Areas in Cryptography-SAC 2009*, volume 5867 of *LNCS*, pages 281–294. Springer, 2009.

27. M. Minier and M. Naya-Plasencia. A Related Key Impossible Differential Attack Against 22 Rounds of the Lightweight Block Cipher LBlock. *Inf. Process. Lett.*, 112(16):624–629, 2012.
28. M. Minier and M. Naya-Plasencia. Private communication, May 2013.
29. T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata. The 128-Bit Blockcipher CLEFIA (Extended Abstract). In *Fast Software Encryption - FSE 2007*, volume 4593 of *LNCS*, pages 181–195. Springer, 2007.
30. T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata. Private communication, May 2014.
31. X. Tang, B. Sun, R. Li, and C. Li. Impossible differential cryptanalysis of 13-round CLEFIA-128. *Journal of Systems and Software*, 84(7):1191–1196, 2011.
32. CLEFIA Design Team. Comments on the impossible differential analysis of reduced round CLEFIA presented at Inscrypt 2008, Jan. 8, 2009.
33. C. Tezcan. The Improbable Differential Attack: Cryptanalysis of Reduced Round CLEFIA. In *INDOCRYPT*, volume 6498 of *LNCS*, pages 197–209. Springer, 2010.
34. Y. Tsunoo, E. Tsujihara, M. Shigeri, T. Saito, T. Suzaki, and H. Kubo. Impossible Differential Cryptanalysis of CLEFIA. In *FSE*, volume 5086 of *LNCS*, pages 398–411. Springer, 2008.
35. Y. Tsunoo, E. Tsujihara, M. Shigeri, T. Suzaki, and T. Kawabata. Cryptanalysis of CLEFIA using multiple impossible differentials. In *Information Theory and Its Applications. ISITA 2008*, pages 1–6, 2008.
36. W. Wu and L. Zhang. LBlock: A Lightweight Block Cipher. In *ACNS 2011*, volume 6715 of *LNCS*, pages 327–344. Springer, 2011.
37. W. Wu, L. Zhang, and W. Zhang. Improved Impossible Differential Cryptanalysis of Reduced-Round Camellia. In *Selected Areas in Cryptography-SAC 2008*, volume 5381 of *LNCS*, pages 442–456. Springer, 2008.
38. W. Wu, W. Zhang, and D. Feng. Impossible Differential Cryptanalysis of Reduced-Round ARIA and Camellia. *J. Comput. Sci. Technol.*, 22(3):449–456, 2007.
39. Z. Yuan, X. Li, and H. Liu. Impossible Differential-Linear Cryptanalysis of Reduced-Round CLEFIA-128. Cryptology ePrint Archive, Report 2013/301, 2013.
40. W. Zhang and J. Han. Impossible Differential Analysis of Reduced Round CLEFIA. In *Inscrypt*, volume 5487 of *LNCS*, pages 181–191. Springer, 2008.

A Application to the SIMON family of block ciphers

SIMON is a family of lightweight block ciphers, optimized for performance on hardware devices, recently proposed by the NSA [6]. Though its very recent appearance and the fact that nothing is said about its resistance against cryptanalysis in the description document, several results on reduced versions have already appeared [4, 2, 3, 9]. We apply here our method on SIMON and come out with the best impossible differential attacks for all versions of the algorithm, as well as with the best known attacks on the smaller variants.

A.1 Brief description of the SIMON family

The SIMON family of block ciphers is based on a classical Feistel construction operating on two n-bit branches. Therefore, the total block size is equal to $2n$ bits. The round function is composed of very simple operations consisting of rotations, XORs and the AND operation. More precisely, at each round, a nonlinear function $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ transforms the left branch in the following way:

$$F(L_{i-1}) = ((L_{i-1} \lll 8)\&(L_{i-1} \lll 1)) \oplus (L_{i-1} \lll 2).$$

The output of F is then XORed with the round subkey and with the right branch R_{i-1} to form the left input of the next round. The round function is iterated r times, where the exact number of iterations r depends on the cipher's version. There exist in total ten members of the SIMON family, each one characterized by different block and key sizes. We denote a member of the SIMON family by SIMONx/x', where x denotes the block size and x' the key size. All versions can be seen in column 1 of Table 2. For the key schedule description, we refer to the description document [6].

A.2 Previous cryptanalysis and comparison to our results

Since its recent publication, SIMON has received a remarkable amount of analysis from the community. Most of these works [4, 2, 3, 9] analyze the resistance of the cipher against differential, impossible differential, linear and rectangle attacks. The best current results are due to differential cryptanalysis [2, 9], while

the number of attacked rounds with impossible differential attacks is much lower for all versions. Also, proposed impossible differential attacks on SIMON present flaws. In [4], the data complexity is too high for the attack to work, while in [1, 2] the computed c_{in} is not correct, as we can check from our Figure 7, where the input rounds are the same as in their case, and c_{in} should be 22, instead of 10.

We provide here impossible differential attacks for all members of the SIMON family. With these results we attack, for all versions, much more rounds than the previous best impossible differential attacks. Furthermore, our attacks constitute the best known cryptanalysis results for the three smaller versions.

A.3 Impossible differential attacks on SIMON

We provide here our attacks on the SIMON family. As the approach used in our analysis is the same for all the versions except of SIMON96/96 and 128/128, we will only present the details of the attack on SIMON32/64 and give briefly the attack parameters for the other versions.

Without using the improvements of Section 2.5, it would not have been possible to mount impossible differential attacks on any version of SIMON, as the data available from the obtained patterns would not have been enough. Indeed, for all versions of SIMON, it holds that $c_{in} = |\Delta_{in}|$ and $c_{out} = |\Delta_{out}|$ implying that $C_N \geq 2^{n+1}$ for all $N \geq 2^{c_{in}+c_{out}}$. This is why we mount our attacks by rotating the second part of the impossible differential while keeping the same first part (see Section 2.5) so that another impossibility in the middle is produced. If we rotate the output pattern by 2 bits (see Table 4), it is possible to generate a second impossible differential with the same first part of the differential. More precisely, for each version of SIMON, there exist at least two output patterns that give the longest impossible differential for a given input pattern. Therefore, for all versions $|\Delta'_{out}| = |\Delta_{out}| + 1$. This method ensures enough data to attack all versions of the SIMON family except for SIMON96/96 and 128/128.

In Section A.5 we present an example of how to simultaneously apply both ideas from Section 2.5. This approach does not change the number of attacked rounds but it permits to improve the data complexity for all versions and to provide valid attacks against SIMON96/96 and 128/128.

Attack on SIMON32/64. By following the previous approach we found that the longest impossible differentials (32 in total) are covering 11 rounds. The impossible differentials that we used for our attack can be visualized in Table 4. However, we note here, that any other maximum-length differential would have led to an equivalent attack. This differential was placed between rounds 5 and 16 and extended by $r_{in} = 4$ rounds and $r_{out} = 4$ rounds to both directions. In such a way, the first 19 ($= 4 + 11 + 4$) rounds of the cipher were attacked. It can be seen in Figure 7 that the number of bit-conditions is $c_{in} + c_{out} = 44$, with $c_{in} = c_{out} = 22$. We can equally see that $|\Delta_{in}| = 22$ and $|\Delta'_{out}| = 23$. Determining the number of information key bits $|k_{in} \cup k_{out}|$ in the case of SIMON is straightforward. Indeed, in this case, the number of information key bits is simply the sum of the different subkey bits intervening in the attack. These bits are marked in blue in Figure 7, from where we can easily verify that $|k_{in} \cup k_{out}| = 2 * 27 = 54$. The complexity of our attack can be seen in Table 5. Notice that for this attack on SIMON32/64, $C'_E = \frac{54}{16 \cdot 19} = 2^{-2.49}$.

r	Left branch L_r	Right branch R_r
4	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
5	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
6	0 0 0 0 0 0 0 ∗ 0 0 0 0 0 1 ∗ 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
7	0 0 0 0 0 ∗ ∗ 0 0 0 0 1 ∗ ∗ 0 ∗	0 0 0 0 0 0 0 ∗ 0 0 0 0 0 1 ∗ 0
8	0 0 0 ∗ ∗ ∗ 0 ∗ 0 1 ∗ ∗ ∗ ∗ ∗ 0	0 0 0 0 0 ∗ ∗ 0 0 0 0 1 ∗ ∗ 0 ∗
9	0 ∗ ∗ ∗ ∗ ∗ ∗ 1 ∗ ∗ ∗ ∗ ∗ ∗ 0 ∗	0 0 0 ∗ ∗ ∗ 0 ∗ 0 1 ∗ ∗ ∗ ∗ ∗ 0
10	∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗	0 ∗ ∗ ∗ ∗ ∗ ∗ **1** ∗ ∗ ∗ ∗ ∗ ∗ 0 ∗
10	∗ 0 1 ∗ ∗ ∗ ∗ ∗ 0 0 0 0 ∗ ∗ ∗ 0	1 ∗ ∗ ∗ ∗ ∗ ∗ **0** ∗ **0** ∗ ∗ ∗ ∗ ∗ ∗
11	0 0 0 0 1 ∗ ∗ 0 ∗ 0 0 0 0 0 ∗ ∗	∗ 0 1 ∗ ∗ ∗ ∗ ∗ 0 0 0 0 ∗ ∗ ∗ 0
12	∗ 0 0 0 0 0 1 ∗ 0 0 0 0 0 0 0 0	0 0 0 0 1 ∗ ∗ 0 ∗ 0 0 0 0 0 ∗ ∗
13	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0	∗ 0 0 0 0 0 1 ∗ 0 0 0 0 0 0 0 0
14	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
15	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Table 4. Impossible differential for the attack on SIMON32/64. 0 denotes a bit without difference, 1 denotes a bit with a difference, and ∗ denotes a bit for which we do not know whether there is a difference or not.

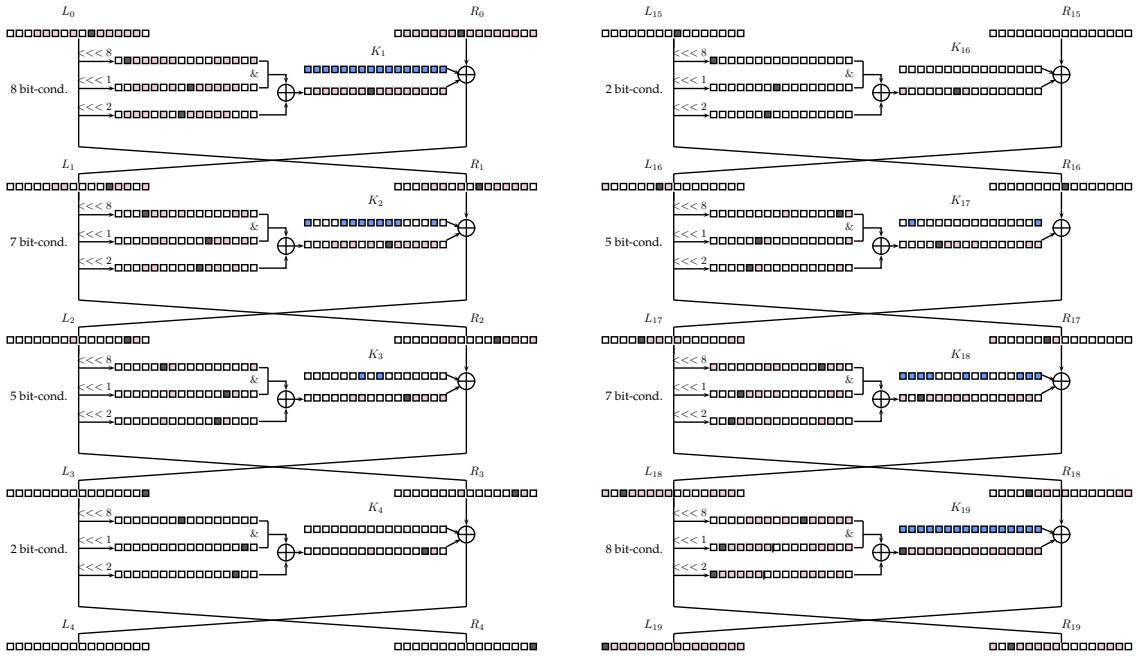


Fig. 7. The initial rounds (on the left) and the final rounds (on the right) of the attack on SIMON32/64.

A.4 Attacks on the other versions

We used the same strategy to attack the other versions of the SIMON family. The results for all versions except for SIMON96/96 and 128/128 are summarized in Table 5.

	SIMON 32/64	SIMON 48/72	SIMON 48/96	SIMON 64/96	SIMON 64/128	SIMON 96/144	SIMON 128/192	SIMON 128/256
Rounds	19	20	21	21	22	25	28	30
Time	$2^{62.56}$	$2^{70.69}$	$2^{94.73}$	$2^{94.56}$	$2^{126.56}$	$2^{142.59}$	$2^{190.56}$	$2^{254.68}$
Data	2^{32}	2^{48}	2^{48}	2^{64}	2^{64}	2^{96}	2^{128}	2^{128}
Memory	2^{44}	2^{58}	2^{70}	2^{60}	2^{75}	2^{77}	2^{77}	2^{111}

Table 5. Complexity summary of our attacks on the majority of SIMON versions.

A.5 Using multiple impossible differentials for attacking SIMON32/64-96/96 and 128/128

Example on SIMON32/64. When considering both possibilities from Section 2.5, we can easily find 4 rotated input patterns (by 0,7,8,31 to the left), independent one from another. We can then define $|\Delta'_{in}| = 22 + \log 4 = 24$ and $|\Delta'_{out}| = 22 + \log 2 = 23$. All the remaining parameters stay the same as in the attack described in Section A.3. The best complexities for the attack are given by considering $N = 2^{45}$, with $C_N = 2^{31}$ and a time complexity of $2^{61.12}C_E$, or by using the whole codebook giving a time complexity of $2^{58.28}C_E$.

Example on SIMON96/96 *and 128/128.* In the 96/96 case, we consider an attack on 24 rounds (4+16+4), where we can find 8 independent input patterns generated by rotations of an original one (by 0,7,8,16,19,25,31,37 to the left). With $|\Delta'_{in}| = 33$ and $|\Delta'_{out}| = 31$, if we consider $N = 2^{61}$, we

have $C_N = 2^{94}$ and a time complexity of $2^{94.62}C_E$. The same can be done with 128/128 and 27 rounds (4+19+4), obtaining a time complexity of $2^{126.6}C_E$.

B Application to LBlock

LBlock is a lightweight block cipher designed by Wu and Zhang in 2011 [36]. It is an iterated construction that can be seen as a variant of a Feistel network. The total number of iterations is equal to 32. The cipher operates on a 64-bit state and encrypts messages by using a 80-bit key.

We start by providing a short description of the algorithm specifications. For a more complete description one can refer to [36].

B.1 Description of LBlock

Encryption algorithm. Denote by $P = L_0|R_0$ a 64-bit plaintext, where L_0 and R_0 are 32-bit vectors. The encryption procedure is as follows.

- For $i = 1, 2, \ldots, 31$ do
  - $R_i = L_{i-1}$
  - $L_i = F(L_{i-1}, K_i) \oplus (R_{i-1} \lll 8)$.
- $L_{32} = L_{31}$
- $R_{32} = F(L_{31}, K_{32}) \oplus (R_{31} \lll 8)$.
- Output the ciphertext $C = L_{32}|R_{32}$.

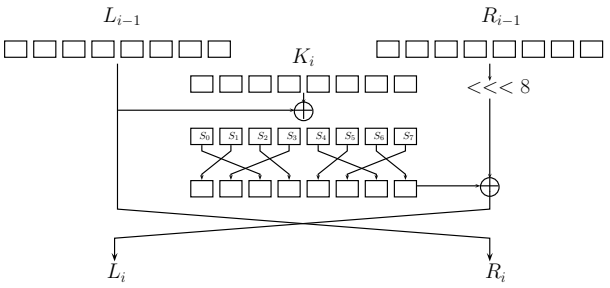


Fig. 8. A round of LBlock

Round function F. A round of LBlock is depicted in Figure 8. The round function F can be divided into three steps. First, the 32-bit subkey K_i is added to L_{i-1} by a simple XOR. Then, a nonlinear layer applies to the result. This nonlinear layer consists of the application nibble by nibble of eight different 4-bit Sboxes $S_0, \ldots, S_7$. The description of these Sboxes can be found in [36]. Finally, the resulting nibbles are permuted as seen in Figure 8.

We are going now to describe our attack against 23 rounds of LBlock. To our knowledge, this is so far the best known attack against this cipher. We will equally present an improved impossible differential attack of the one published so far on 22 rounds. Here, we show how to build the attack using the generic attack strategy and the time complexity estimation, and it can be seen in [13] how this generic estimation perfectly corresponds to the detailed and careful time complexity computation.

B.2 Impossible differential attack on 23 rounds of LBlock

First, we notice that a difference $P^i = 0_{32}|(0_4, 0_4, 0_4, 0_4, \alpha, 0_4, 0_4, 0_4)$ cannot lead to a difference $P^{i+14} = (0_4, 0_4, 0_4, 0_4, 0_4, \beta, 0_4, 0_4)|0_{32}$ after 14 rounds. We set this impossible differential between the rounds 5 and 19 of the cipher. Note that this impossible differential is different from both impossible differentials used to attack 21 and 22 rounds of LBlock in [19].

We extend this impossible differential by adding $r_{in} = 5$ rounds to the input and $r_{out} = 4$ rounds to the output and attack therefore the 23 first rounds of the cipher. As can be seen in Figure 9, the number of input bit-conditions is $c_{in} = 4 \times 11 = 44$ and the number of output bit-conditions $c_{out} = 4 \times 7 = 28$. In the same way, $|\Delta_{in}| = 12 \times 4 = 48$ and $|\Delta_{out}| = 8 \times 4 = 32$.

As $c_{in} + c_{out} = 72$, we see that we need at least $N_{\min} = 2^{72}$ pairs for the attack to work. By trying different values for N, we found out that the best time complexity is given when choosing $N = 2^{74}$. The amount of data we need to construct these pairs is

$$C_N = \max\left\{\sqrt{2^{74}2^{65-48}}, 2^{74}2^{65-48-32}\right\} = 2^{59}.$$

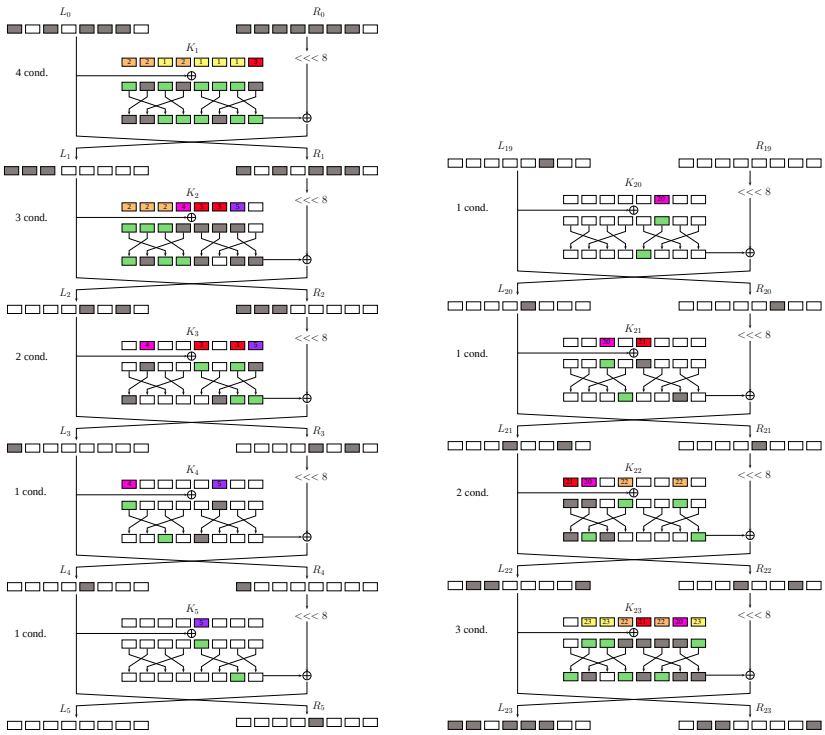


Fig. 9. The initial rounds (on the left) and the final rounds (on the right). Different colors stand for the round that these key bits intervene.

When analyzing the key schedule of LBlock, whose detailed description can be found in [36], we noticed that the amount of information key bits in our attack is $|k_{in} \cup k_{out}| = 73$ bits.

We briefly recall in Figure 10 the parameters of our attack on 23 rounds.

Δ_{in}	$(*0*0***0, *******0)$
Δ_{out}	$(**0***00, 0**0000*)$

Δ_X	$(00000000, 0000\alpha000)$
Δ_Y	$(00000\beta00, 00000000)$

$\lvert\Delta_{in}\rvert$	$\lvert\Delta_{out}\rvert$	r_{in}	r_{out}	r_Δ	c_{in}	c_{out}	$\lvert k_{in} \cup k_{out}\rvert$
48	32	5	4	14	44	28	73

Fig. 10. Parameters of our impossible differential cryptanalysis of 23-round LBlock.

By using the Formula (2) we compute the time complexity of our 23-round attack on LBlock with 2^{59} chosen plaintexts, to be $2^{75.36}C_E$. The memory complexity, determined by N, is 2^{74}. It is important to point out that this estimated complexity perfectly correspond to the carefully computed one that is detailed in [13], showing that our time complexity estimation is indeed very tight.

B.3 Improvement of the Impossible Differential Cryptanalysis of 22-round LBlock

The previous best known attack against LBlock was an impossible differential attack on 22 rounds [19], of time complexity $2^{79.28}C_E$. We show here that it is possible to mount an 22-round attack with an improved time complexity. More precisely, by applying the formulas of Section 2, we obtain an attack of data complexity 2^{60}, time complexity $2^{71.53}C_E$ and memory complexity 2^{59}. The parameters of our attack are depicted in Figure 11.

Δ_{in}	$(***00000, *0*0***0)$
Δ_{out}	$(**0***00, 0**0000*)$

Δ_X	$(00000000, 0000\alpha000)$
Δ_Y	$(00000\beta00, 00000000)$

$\lvert\Delta_{in}\rvert$	$\lvert\Delta_{out}\rvert$	r_{in}	r_{out}	r_Δ	c_{in}	c_{out}	$\#(k_{in} \cup k_{out})$
32	32	4	4	14	28	28	71

Fig. 11. Parameters of our impossible differential cryptanalysis of 22 rounds of LBlock.

Cryptogr. Commun.
DOI 10.1007/s12095-012-0069-3

Correlation attacks on combination generators

Anne Canteaut · María Naya-Plasencia

Received: 22 January 2012 / Accepted: 1 August 2012

Abstract The combination generator is a popular stream cipher construction. It consists of several independent devices working in parallel whose outputs are combined by a Boolean function. The output of this function is the keystream. The security of this generator has been extensively studied in the case where the devices are LFSRs. Some particular cases where the devices are nonlinear have also been studied, most notably the different versions of the eSTREAM proposal named Achterbahn. Several cryptanalysis techniques against these ciphers have been published, extending the classical correlation attack. But each of these attacks has been presented mainly in a very particular scenario. Therefore, this paper aims at generalising these methods to any combination generator in order to be able to compare their respective advantages and to determine the optimal attack for each particular generator. Generic formulas for the data-time-space complexities are then provided, which only depend on the number of devices, their periods and the number of their internal states and of the Boolean combining function. Some of the considered improvements can also be used in a much more general context, which includes linear attacks against some block ciphers.

Keywords Correlation attacks · Stream cipher · NLFSR · Parity-check

Mathematics Subject Classifications (2010) 68P25 · 94A60

A. Canteaut ( )
INRIA project-team SECRET, B.P. 105, 78153 Le Chesnay cedex, France
e-mail: Anne.Canteaut@inria.fr

M. Naya-Plasencia
Laboratoire PRISM, Université de Versailles St-Quentin-en-Yvelines,
45 avenue des Etats-Unis, 78035 Versailles Cedex, France
e-mail: maria.naya.plasencia@gmail.com

Published online: 21 August 2012

1 Introduction

One of the most popular constructions of stream ciphers is the combination generator. In this model, several independent devices (e.g. feedback shift registers) work in parallel. Their outputs are taken as inputs by a Boolean combining function, and the output of this function provides the keystream bits. The case where the combination generator uses shift registers with a linear feedback function is a very old and well-studied model which has been shown to be vulnerable to several attacks, including correlation attacks [5–7, 25–27, 29, 33, 34, 38], algebraic attacks [9, 10] and distinguishing attacks [3, 22]. Meanwhile, the combination generators composed of LFSRs with unknown feedback polynomials or of shift registers with nonlinear feedbacks appear to be less vulnerable to this type of attacks.

Such a combination generator using nonlinear feedback shift registers (NLFSRs) was submitted in 2005 to the eSTREAM public competition [11], launched by the ECRYPT European network in order to recommend some secure stream ciphers. This keystream generator named Achterbahn was designed by Gammel et al. [13–16]. A version of this algorithm was selected for the second phase of the competition. It was afterwards eliminated due to several attacks presented on its successive versions [20, 21, 28, 35, 36].

However, since each of these attacks has been applied to a different scenario and described in a different paper, it is hard to compare their respective advantages, to determine precisely the scenarios where each variant applies and to decide which is the optimal attack in a given context. This paper then aims at reviewing and generalizing these attacks against combination generators with nonlinear constituent devices in order to include all these variants in a well-defined family and to provide generic formulas for the complexities in data, time and memory, depending on the parameters of the generator. Here, we also want to determine the parameters of the combination generator which make it resistant to the whole family of attacks. Indeed, these attacks only require the knowledge of the periods of the sequences produced by the constituent devices and of the Boolean combining function. The result is that, once we are given such a generator, we will be able to determine in an automatic way, the different trade-offs and then the complexities of the different attacks that can be applied. Therefore, it makes it possible to design such a cipher, with an a priori knowledge of its security level regarding correlation attacks (which are the best known applicable attacks up to date). We also show that some of the techniques used for attacking the combination generator apply to a more general problem which includes for instance the problem to be solved in linear attacks against iterated block ciphers with Matsui's Algorithm 2.

The paper is organised as follows. Section 2 presents the combination generator which will be analysed in detail, as well as a the basic principle of correlation attacks against this generator and the more general problem which must be solved for breaking this type of generator. Then, Section 3 describes a general method for speeding-up most correlation attacks as soon as the considered biased sequence can be decomposed as the sum of two sequences with independent initial states. Section 4 presents another technique introduced by Hell and Johansson [20] which leads to a better trade-off between time and data complexities for solving the general correlation problem. Section 5 then shows that, in a correlation attack against the combination generator, both previous improvements can be applied to detect the

correlation between parity-check relations derived from the generator. It finally compares the different trade-offs achieved by these attacks and discusses which are the best ones to be applied depending on the situation.

2 General model and notation

2.1 The general combination generator

The general keystream generator which is studied throughout the paper is a pseudo-random generator composed of independent binary devices, i.e., each of these devices is a finite-state automaton producing one bit at each time instant. A combination generator based on n such devices consists in combining the outputs of the devices by a Boolean function f of n variables. Then, the output of this Boolean function at each time instant provides the corresponding bit of keystream (Fig. 1).

In the paper, the keystream is denoted by $\boldsymbol{S} = (S(t))_{t\geq 0}$. Moreover, $\mathcal{R}_i$ denotes device i and L_i denotes the number of bits of its internal state. The sequence produced by $\mathcal{R}_i$ is denoted by $\boldsymbol{x}_i = (x_i(t))_{t\geq 0}$. The important and only fact that the attacker needs to known about this sequence is that it is a periodic sequence with period $T_i \leq 2^{L_i}$. In Section 5, for the sake of simplicity, we consider the periods of the n sequences to be coprime. The attacks are also valid if it is not the case, though their complexities could be easily reduced.

The internal states of the devices are usually initialised from the secret key and a public initialisation vector by an initialisation algorithm. In this paper, we will focus on state-recovery attacks only, i.e., we will aim at recovering the initial states of the devices just before starting generating the keystream sequence, leaving the key-recovery problem which highly depends on the properties of each initialisation algorithm. The fact that an attacker is able to recover the initial states of the devices is obviously considered as a major weakness of the cipher on its own, as it implies for instance that the keystream can be reproduced. Also, in all practical cases that we have studied, the state-recovery attack could always be turned into a key-recovery attack.

2.2 Principle of correlation attacks

As for all attacks against synchronous stream ciphers, we will consider the known-plaintext scenario which equivalently means that we assume that some keystream bits are known to the attacker.

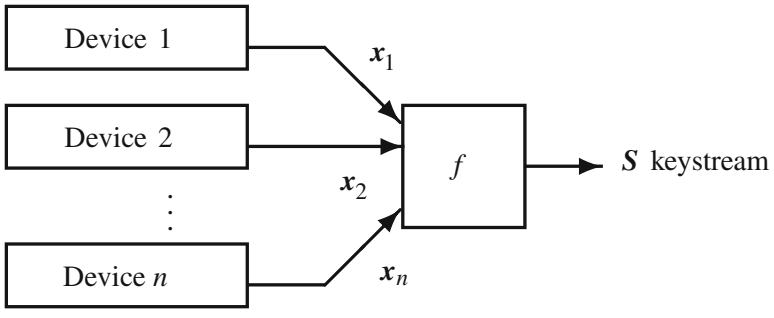


Fig. 1 Keystream generator composed of several independent devices combined by a Boolean function

A generic attack which always applies to this type of cipher is the exhaustive search for the initial states of the n devices. For each possible initial configuration, we can compute the generated keystream and deduce the correct initial state when the sequence produced by the combining function is the same as the observed keystream. Such a generic attack requires at least $2^{\sum_{i=1}^{n} L_i}$ trials. In a well-conceived stream cipher, this time complexity is too large and makes the attack infeasible.

In this context, correlation attacks introduced by Siegenthaler [38] are divide-and-conquer attacks in the sense that they aim at recovering the initial states of some of constituent devices only, independently from the other ones. In other words, they exploit the existence of a smaller generator, composed of a subset of the constituent devices combined by a Boolean function having fewer input variables than f, whose output $\boldsymbol{\sigma} = (\sigma(t))_{t\geq 0}$ is correlated to the keystream produced by the original generator.

If the n-bit vector corresponding to the outputs of the n devices at each time instant is uniformly distributed, the existence of such a small generator is equivalent to the existence of a biased approximation of f depending on fewer variables, in the sense of the following definition.

Definition 1 Let h be a Boolean function with n variables. Then, the bias of h is

$$\mathcal{E}(h) = 2\Pr[h(x) = 0] - 1 = \frac{1}{2^n}\big[\sharp\{x \in \mathbf{F}_2^n, h(x) = 0\} - \sharp\{x \in \mathbf{F}_2^n, h(x) = 1\}\big].$$

Sometimes, this quantity is called the imbalance of h, since the function is said to be balanced if $\mathcal{E}(h) = 0$. It also corresponds to the correlation between h and the all-zero function.

If f and g are two Boolean functions, the correlation between f and g, also named the bias of the approximation of f by g, equals $\mathcal{E}(f + g)$.

In a correlation attack, the lowest number of devices which must be considered simultaneously in the small generator is the lowest integer m such that there exists g, a biased approximation of the Boolean function f on n input variables, which depends on m input variables only. By definition, the smallest m is equal to $R + 1$ where R is the so-called *resiliency order* (aka correlation-immunity order when f is balanced) of the combining function [37]. It is worth noticing that R is upper-bounded by $(n - 1 - \deg f)$ [37], and that the algebraic degree of f cannot be too low for avoiding algebraic attacks for instance. Then, a precomputational step in the attack consists in finding an appropriate biased approximation g of f, which depends on m variables. For $m = R + 1$, which is usually the best choice for the attacker, it is known that the approximation of f with $m = R + 1$ variables which has the highest bias is the linear function corresponding to the sum of all involved variables (possibly with a nonzero constant term) [5]. In the case where it is suitable to consider more than $R + 1$ devices together, then the approximation with the highest bias may be nonlinear, but it can be easily computed by the technique described in [39, Theorem 1]. It is worth noticing that there is no need for maximizing the magnitude of $\mathcal{E}(f + g)$ instead of maximizing $\mathcal{E}(f + g)$ when there is no restriction on the value of $g(0)$. Indeed, for any approximation g, we have $\mathcal{E}(f + (g \oplus 1)) = -\mathcal{E}(f + g)$. It follows that we can assume that $\mathcal{E}(f + g)$ is always positive when g is a good approximation of f.

Once an appropriate approximation has been found, the attack consists in recovering the correct initialisations of the m targeted devices. The simplest method is the one originally proposed by Siegenthaler [38]: it performs an exhaustive search for the initial state of the small generator, i.e., for the initial states of the m targeted devices. For each initialisation, the attacker produces N bits of the output $\boldsymbol{\sigma}$ and computes the correlation between these N bits and the corresponding keystream bits, i.e.,

$$\sum_{t=0}^{N-1}(-1)^{S(t)+\sigma(t)} .$$

The correct initialisation is then expected to be the one maximizing the correlation, or to lead to a correlation higher than some appropriate threshold.

Therefore, this attack is a particular instance of the following more general problem, which will be referred to as the *general correlation problem*. Let $\boldsymbol{z} = (z(t))_{t\geq 0}$ be a binary sequence depending on a secret parameter K (e.g. a part of the initial state) which can take 2^k different values. Assume that, for any t,

$$\Pr[z(t) = 0] = \begin{cases} \frac{1}{2}(1+\varepsilon) & \text{if } K = K^\star \\ \frac{1}{2} & \text{otherwise} \end{cases}$$

where $K^\star$ is a given unknown value and $\varepsilon > 0$. The problem is then to recover $K^\star$, i.e., to find the initial state which maximizes

$$\sum_{t=0}^{N-1}(-1)^{z(t)} .$$

It is worth noticing that linear attacks against iterated block ciphers with Matsui's Algorithm 2 [32] are faced with the same problem where $\boldsymbol{z}$ consists of several evaluations of a linear relation between the plaintext and the ciphertext.

In the following, we present two techniques for solving the general correlation problem when the sequence $\boldsymbol{z}$ can be decomposed as a sum of several sequences with independent initial states and with periods less than N. Section 3 describes a general algorithm which reduces the time complexity of the attack with a similar data complexity. This algorithm has been used in the particular case of the attack against Achterbahn 80/128 proposed by Naya-Plasencia [35]. The second improvement described in Section 4 has been first proposed by Hell and Johansson [20]. It consists in using decimated sequences in order to achieve a better trade-off between time and data complexity. Finally, Section 5 shows that both improvements can be used for computing the correlation between so-called parity-check relations in the particular context of a correlation attack against a combination generator. The obtained attacks apply to any such generator once the periods of the n constituent sequences are known, as well as the (possibly strong) Boolean combining function f. In general, we will see that the main weaknesses of this generator might originate from two possible facts: the fact that the periods of the sequences produced by the devices are too small (even though the number of devices, n, is big), and a weak combining function.

3 Speeding-up the general correlation attack

3.1 Basic algorithm

The basic technique for solving the general correlation problem consists in performing an exhaustive search for the secret initial state of the sequence $\boldsymbol{z}$ and in applying a hypothesis test to recover the correct initialisation. The optimal hypothesis test is defined by the Neyman–Pearson lemma which compares the value of the correlation to an appropriate threshold. All initial state candidates can also be sorted as specified by the Neyman–Pearson lemma, and then they can be tested in order of probability [30]. It is known, for instance from [22, Section 4.1], that the number of samples needed to determine the correct initialisation out of 2^k possible values is then

$$N \simeq \frac{2k \ln 2}{\varepsilon^2},$$

where ε denotes the bias of the sequence $\boldsymbol{z}$. The time complexity is then $N2^k$.

In the particular case where $\boldsymbol{z}$ depends affinely on its k-bit initial state K, i.e., if there exists a sequence $(y(t))_{t\geq 0}$ independent from K such that

$$z(t) = \alpha_t \cdot K \oplus y(t), \ \forall t \geq 0 \ ,$$

the time complexity can be reduced by using a FFT technique as proposed for instance in [7, 31]. Indeed, for any possible initial state K, we have

$$\mathcal{Z}(K) = \sum_{t=0}^{N-1} (-1)^{z(t)} = \sum_{t=0}^{N-1} (-1)^{\alpha_t \cdot K \oplus y(t)} = \sum_{x \in \mathbf{F}_2^k} F(x)(-1)^{x \cdot K}$$

where F is a function from $\mathbf{F}_2^k$ into $\mathbb{Z}$ defined by

$$F(x) = \sum_{t<N,\ \alpha_t = x} (-1)^{y(t)} \ ,$$

where $F(x) = 0$ when x does not correspond to any α_t, $0 \leq t < N$. The set of all $\mathcal{Z}(K)$, $K \in \mathbf{F}_2^k$, can then be obtained by computing the fast Walsh–Hadamard transform of F with complexity $\mathcal{O}(k2^k)$.

This technique leads to an attack against the combination generator with m targeted devices, $\mathcal{R}_{i_1}, \ldots, \mathcal{R}_{i_m}$. Throughout the paper, we use the same notation as in Fig. 2: a combination generator with n constituent devices is considered. Its combining function is denoted by f and depends on n variables. The attack then uses an approximation g of f, which depends on m variables only. In this context, the data complexity of the attack is then

$$\frac{2 \ln 2 \sum_{j=1}^{m} L_{i_j}}{\varepsilon^2} \ ,$$

where $\varepsilon = \mathcal{E}(f + g)$ and the time complexity with the basic algorithm is

$$\frac{2 \ln 2 \sum_{j=1}^{m} L_{i_j}}{\varepsilon^2} \times 2^{\sum_{j=1}^{m} L_{i_j}} \ .$$

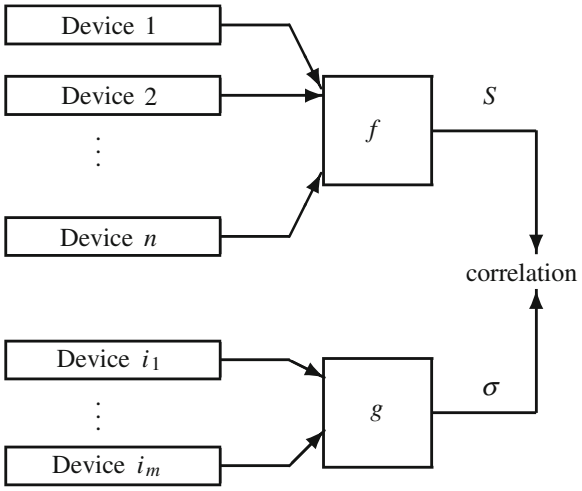


Fig. 2 Principle of correlation attacks against the combination generator

But, the FFT technique may apply, in particular when the targeted devices are LFSRs and when the best approximation g of f is linear, which is always the case when the small generator involves the smallest possible number of devices, $m = R + 1$, where R is the resiliency order of f. Then, the sequence $\boldsymbol{\sigma}$ produced by the small generator can be expressed as

$$\sigma(t) = \bigoplus_{j=1}^{m} \left(\alpha_{j,t} \cdot K_{i_j}\right) = (\alpha_{1,t}, \ldots, \alpha_{m,t}) \cdot (K_{i_1}, \ldots, K_{i_m}) \,,$$

where K_{i_j} denotes the initial state of register $\mathcal{R}_{i_j}$. The time complexity for recovering the initial states of these m registers is then reduced to

$$\sum_{j=1}^{m} L_{i_j} 2^{\sum_{j=1}^{m} L_{i_j}} + \frac{2 \ln 2 \sum_{j=1}^{m} L_{i_j}}{\varepsilon^2} \,,$$

where the first term corresponds to the FFT computation and the second one to the computation of the values of F. Therefore, the complexity of the general algorithm is divided by $\frac{2 \ln 2}{\varepsilon^2}$.

3.2 Speeding-up the exhaustive search in the general case

The previously described FFT technique only applies to the case where $\boldsymbol{z}$ depends affinely on its initial state, i.e., in the case of the combination generator, when the small generator has a linear next-state function. In particular, it cannot be used when the constituent devices are nonlinear devices as in the case of Achterbahn. However, we now present a method which leads to a similar speed-up when the combining function of the small generator can be decomposed as a sum of two functions with disjoint input variables, i.e., $g(x_{i_1}, \ldots, x_{i_m}) = g_u(x_{i_1}, \ldots, x_{i_{m'}}) + g_v(x_{i_{m'+1}}, \ldots, x_{i_m})$. This obviously occurs when the small generator corresponds to the sum of m devices, which is the practical situation when the combining function of the keystream generator has an appropriate biased linear approximation, in particular when the number of targeted devices is minimal. This technique has been introduced by [35] in the particular context of the cryptanalysis of Achterbahn, and here we provide a more general description.

More generally, our technique can be used for the general correlation problem, in order to determine among the 2^k initial states the one which maximizes

$$\sum_{t=0}^{N-1}(-1)^{z(t)} ,$$

as soon as this sequence can be expressed as the sum (modulo 2) of two sequences, $\boldsymbol{u}$ and $\boldsymbol{v}$ with independent initial states and such that the period T_u of $\boldsymbol{u}$ is known and smaller than N. The main idea of the algorithm is then to gather some computations which depend on $\boldsymbol{u}$ only, and then to exploit the fact that the period of $\boldsymbol{u}$ is smaller than N in order to compute those terms T_u times only, instead of N times.

In the case of an attack against the combination generator, the sequence $\boldsymbol{z} = \boldsymbol{S} \oplus \boldsymbol{\sigma}$ has such a decomposition if the approximation g of f can be decomposed as $g(x_{i_1}, \ldots, x_{i_m}) = g_u(x_{i_1}, \ldots, x_{i_{m'}}) + g_v(x_{i_{m'+1}}, \ldots, x_{i_m})$. Then, $\boldsymbol{u}$ corresponds to the combination by g_u of the outputs of m' devices among the m targeted devices, e.g.,

$$\boldsymbol{u} = g_u(x_{i_1}(t), \ldots, x_{i_{m'}}(t)) .$$

Indeed, it is known that

$$T_u = \prod_{j=1}^{m'} T_{i_j}$$

is a period of $\boldsymbol{u}$. In this case, we have

$$\boldsymbol{v} = \left(S(t) \oplus g_v(x_{i_{m'+1}}(t), \ldots, x_{i_m}(t))\right)_{t\geq 0} .$$

We now compute the correlation between $\boldsymbol{u}$ and $\boldsymbol{v}$. As previously explained, detecting this bias requires the knowledge of a number of samples N which exceeds

$$N_0 = \frac{2k \ln 2}{\varepsilon^2} . \tag{1}$$

Then, for each of the 2^k possible initial states of $\boldsymbol{z}$, we compute the sum $\sum_{t=0}^{N-1} z(t)$ on N bits where N is the smallest multiple of T_u which exceeds N_0, i.e.,

$$N = T_u \left\lceil \frac{N_0}{T_u} \right\rceil .$$

We denote $c = \frac{N}{T_u}$. Then, the previous sum can be rewritten in the following way:

$$\begin{aligned}
\sum_{t=0}^{N-1} z(t) &= \sum_{t=0}^{N-1} u(t) \oplus v(t) \\
&= \sum_{r=0}^{T_u-1} \sum_{q=0}^{c-1} u(qT_u + r) \oplus v(qT_u + r) \\
&= \sum_{r=0}^{T_u-1} \sum_{q=0}^{c-1} u(r) \oplus v(qT_u + r)
\end{aligned}$$

since T_u is a period of $\boldsymbol{u}$. Now, we use the fact that, for any pair (q, r),

$$u(r) \oplus v(qT_u + r) = \begin{cases} v(qT_u + r) & \text{if } u(r) = 0 \\ 1 - v(qT_u + r) & \text{if } u(r) = 1 \end{cases}.$$

Therefore, we deduce that

$$\sum_{t=0}^{N-1} z(t) = \sum_{r=0}^{T_u-1} (u(r) \oplus 1) \left(\sum_{q=0}^{c-1} v(qT_u + r) \right) + \sum_{r=0}^{T_u-1} u(r) \left(c - \sum_{q=0}^{c-1} v(qT_u + r) \right).$$

For any r, $0 \leq r < T_u$, we set

$$V(r) = \sum_{q=0}^{c-1} v(qT_u + r).$$

The vector $(V(0), \ldots, V(T_u - 1))$ consists of T_u integers which depend on the initial state of $\boldsymbol{v}$ only (i.e., the initial states of the $(m - m')$ corresponding devices in the context of an attack against the combination generator). Then, this vector can be computed independently from the initial state of $\boldsymbol{u}$. We now have

$$\begin{aligned} \sum_{t=0}^{N-1} z(t) &= \sum_{r=0}^{T_u-1} (u(r) \oplus 1)\, V(r) + \sum_{r=0}^{T_u-1} u(r)\, (c - V(r)) \\ &= \sum_{r=0}^{T_u-1} (-1)^{u(r)} \left(V(r) - \frac{c}{2} \right) + \frac{T_u c}{2}. \end{aligned}$$

It follows that

$$\sum_{t=0}^{N-1} (-1)^{z(t)} = N - 2 \sum_{t=0}^{N-1} z(t) = \sum_{r=0}^{T_u-1} (-1)^{u(r)}\, (c - 2V(r)).$$

Now, we need to compute this sum for each initial state of $\boldsymbol{u}$ and find if there is one that provides the expected bias. However, starting from a particular initial state U_0, we can find the initial state U_τ of $\boldsymbol{u}$ which maximizes this sum within the same cycle as U_0. Indeed, let U_τ denote the internal state of the generator producing $\boldsymbol{u}$ at time τ starting from U_0, for $0 \leq \tau < T_u$. Then, the sequence generated from U_τ is exactly the sequence derived from the sequence generated from U_0 shifted by τ positions, i.e., $(u(t + \tau \bmod T_u))_{0 \leq t < T_u}$. Then, starting from a particular initial state U_0 of $\boldsymbol{u}$, we want to find the value of τ which maximizes the value of

$$\sum_{r=0}^{T_u-1} (-1)^{u(r+\tau \bmod T_u)} \left(V(r) - \frac{c}{2} \right) + \frac{T_u c}{2} \quad \text{for } 0 \leq \tau < T_u$$

which corresponds to the cross-correlation between two sequences, $\boldsymbol{u}$ and $(V(0), \ldots, V(T_u - 1))$, of length T_u. This can be done efficiently with a fast Fourier transform [2, pages 306–312] with complexity $\mathcal{O}(T_u \log T_u)$. Once the maximum of the cross-correlation has been computed for a given initial state of $\boldsymbol{u}$, we have to

repeat this procedure for another value U_0 in a different cycle of $\boldsymbol{u}$. Once all initial states for $\boldsymbol{u}$ have been examined, the same algorithm has to be repeated for another initial state of $\boldsymbol{v}$. The algorithm is described in Algorithm 1 for the particular case of a correlation attack against the combination generator where

$$u(t) = g_u(x_{i_1}(t), \ldots, x_{i_{m'}}(t)) \text{ and } v(t) = S(t) \oplus g_v(x_{i_{m'+1}}(t), \ldots, x_{i_m}(t)) \ .$$

Algorithm 1 Correlation attack against the general combination generator with a speed-up of the exhaustive search for m targeted devices.

for each initial state of the last $(m - m')$ devices **do**
  for r from 0 to $T_u - 1$ **do**
    $V(r) \leftarrow c - 2\sum_{q=0}^{c-1} v(qT_u + r)$
  end for
  repeat
    choose an initial state for the first m' devices which does not belong to a previously examined cycle
    for r from 0 to $T_u - 1$ **do**
      $u(r) \leftarrow g_u(x_{i_1}(r), \ldots, x_{i_{m'}})(r)$
    end for
    Compute the following cross-correlation with an FFT

$$\mathcal{S}(\tau) = \sum_{r=0}^{T_u-1} (-1)^{u(r+\tau \bmod T_u)} V(r), \quad 0 \leq \tau < T_u$$

    if $\mathcal{S}(\tau) >$ threshold for some τ **then**
      return the initial states of the last $(m - m')$ devices and the internal states of the first m' devices after τ clocks.
    end if
  until all initial states of the first m' devices have been examined
end for

Let 2^{k_u} denote the number of possible initial states for $\boldsymbol{u}$. In most practical situations, the number of cycles of $\boldsymbol{u}$ is roughly $2^{k_u}/T_u$ since it is expected that the constituent devices of the generator do not have any short cycles. For instance, in the case of LFSRs with primitive feedback polynomials, coprime periods and with nonzero initial states, the number of possible initial states for $\boldsymbol{u}$ is equal to the period $T_u = \prod_{j=1}^{m'} T_{i_j}$, since it is assumed that the all-zero initial state is avoided for any of the LFSRs. The situation is similar in Achterbahn because all constituent devices are NLFSRs with period $T_i = 2^{L_i} - 1$ for all i. Then, the overall time complexity of the attack is given by the following formula:

$$2^{k-k_u}\left[T_u c + \frac{2^{k_u}}{T_u}(T_u \log T_u)\right] \ .$$

We point out here that even if the cross-correlation is not computed with an FFT, but for all the possible values of τ, the final time complexity would still be reduced compared to the classical attack, as this changes the second term in the previous sum

to $2^{k_u} T_u$, which divides the complexity of the classical attack by a factor $c = N/T_u$. In the situations where the generator producing $\boldsymbol{u}$ has only a few cycles, T_u is close to 2^{k_u} as previously explained. The time complexity can then be expressed as

$$2^k \left[\frac{N_0}{T_u} + \log T_u \right] ,$$

where N_0 is the data complexity, i.e., the minimal number of samples needed for the correlation attack as expressed by (1). Then, the optimal choice for the decomposition of the small generator into $\boldsymbol{u} \oplus \boldsymbol{v}$ corresponds to the situation where T_u is close to $\frac{N_0}{\log N_0}$. In this case, the time complexity is proportional to

$$2^k \log N_0 .$$

This must be compared to $2^k N_0$, which is the time complexity of the classical correlation attack described in the previous section, while the data complexity is unchanged. Let us notice here that instead of considering N, which is a multiple of T_u, we could consider N_0 samples only, as done in [36] as N was limited. This case is similar but the previous sum has to be decomposed into two parts. For the sake of simplicity, we have presented here the case where N is a multiple of T_u, as it usually does not increase the complexity and as the extension is quite straightforward.

The memory requirement corresponds to the storage of both sequences $\boldsymbol{u}$ and $\boldsymbol{V}$ which are composed of T_u bits and of T_u integers respectively.

In the case of an attack against the combination generator as described by Algorithm 1, the time complexity can then be expressed as

$$2^{\sum_{j=m'+1}^{m} L_{i_j}} \left[T_u c + \frac{2^{\sum_{j=1}^{m'} L_{i_j}}}{T_u} (T_u \log T_u) \right] ,$$

where $T_u = \prod_{j=1}^{m'} T_{i_j}$.

4 Correlation attacks using decimation

4.1 Basic principle

Another trade-off between the data and time complexity in correlation attacks can be achieved with a method introduced by Hell and Johansson in [20]. The idea is to perform the correlation attack on a decimated version of the sequence in order to reduce the size of the initial state for which we have to perform an exhaustive search. First, we give a general description of the algorithm presented by Hell and Johansson. Then, we show in Section 4.2 how it can be improved. The technique proposed in [20] applies for determining, among 2^k different initial states of the sequence $\boldsymbol{z}$, the one which maximizes

$$\sum_{t=0}^{N-1} (-1)^{z(t)}$$

as soon as $\boldsymbol{z}$ can be expressed as the sum of two sequences $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$ with independent initial states and such that a period T_d of $\boldsymbol{\gamma}$ is known and smaller than N. Then, for any δ, we have

$$z(tT_d + \delta) = \alpha(tT_d + \delta) \oplus \gamma(tT_d + \delta) = \alpha(tT_d + \delta) \oplus \gamma(\delta) \ .$$

Then, we deduce that

$$\sum_{t=0}^{N-1} (-1)^{z(tT_d+\delta)} = (-1)^{\gamma(\delta)} \sum_{t=0}^{N-1} (-1)^{D_\delta(t)} \ ,$$

where $\boldsymbol{d}_\delta = (D_\delta(t))_{t\geq 0}$ denotes the decimated sequence $(\alpha(tT_d + \delta))_{t\geq 0}$. Therefore, if $\boldsymbol{d}_\delta$ can be computed by the attacker for some δ (e.g. $\delta = 0$), finding the maximal $\sum_t (-1)^{z(t)}$ for N well-chosen values of t boils down to finding the highest magnitude for $\sum_{t=0}^{N-1} (-1)^{D_\delta(t)}$. This can be done by an exhaustive search for the initial state of $\boldsymbol{d}_\delta$. Let 2^{k_d} denote the number of initial states for $\boldsymbol{\alpha}$. Then, $k_d < k$ as $\boldsymbol{\alpha}$ does not depend on the initialisation of $\boldsymbol{\gamma}$. The attack then requires

$$N = \frac{2k_d \ln 2}{\varepsilon^2}$$

evaluations of the decimated sequence, implying that the data complexity is now

$$\frac{2k_d \ln 2 T_d}{\varepsilon^2}$$

which is usually higher than the data complexity without decimation given by (1), but the time complexity is

$$2^{k_d} N = \frac{2k_d \ln 2}{\varepsilon^2} \times 2^{k_d}$$

which improves the usual algorithm in most cases.

This improvement applies in the context of a correlation attack against the combination generator if the approximation g can be decomposed as a sum of two functions with disjoint input variables:

$$g(x_{i_1}, \ldots, x_{i_m}) = g_d(x_{i_1}, \ldots, x_{i_\partial}) + g'(x_{i_{\partial+1}}, \ldots, x_{i_m}) \ .$$

We then decimate the sequence $\boldsymbol{z} = (\boldsymbol{S} \oplus \boldsymbol{\sigma})$ by the product of the periods of the first ∂ devices among $\mathcal{R}_{i_1}, \ldots, \mathcal{R}_{i_\partial}$:

$$T_d = \prod_{j=1}^{\partial} T_{i_j} \ .$$

Then, for any $t \geq 0$ and any δ, we have

$$\begin{aligned}
\sigma(tT_d + \delta) &= g'(x_{i_{\partial+1}}(tT_d + \delta), \ldots, x_{i_m}(tT_d + \delta)) \oplus g_d(x_{i_1}(tT_d + \delta), \ldots, x_{i_\partial}(tT_d + \delta)) \\
&= g'(x_{i_{\partial+1}}(tT_d + \delta), \ldots, x_{i_m}(tT_d + \delta)) \oplus g_d(x_{i_1}(\delta), \ldots, x_{i_\partial}(\delta)) \\
&= g'(x_{i_{\partial+1}}(tT_d + \delta), \ldots, x_{i_m}(tT_d + \delta)) \oplus \gamma(\delta) \ .
\end{aligned}$$

Since the last term in the sum is a constant, the decimated sequence $(\boldsymbol{S} \oplus \boldsymbol{\sigma})$ can be computed (up to a constant) from the initial states of $(m - \partial)$ devices only, instead of

m. The correlation attack is then similar as before, except that we compute the correlation between decimated versions of $\boldsymbol{S}$ and $\boldsymbol{\alpha} = (g'(x_{i_{\partial+1}}(t), \ldots, x_{i_m}(t)))_{t\geq 0}$. Then, the magnitude of the correlation, instead of its signed value, must be maximized.

The correct initial states of the last $(m - \partial)$ devices can then be recovered with

$$N \simeq \frac{2 \ln 2 \sum_{j=\partial+1}^{m} L_{i_j}}{\varepsilon^2} \text{ samples },$$

leading to the following data complexity

$$\frac{2 \ln 2 \sum_{j=\partial+1}^{m} L_{i_j} \prod_{j=1}^{\partial} T_{i_j}}{\varepsilon^2} . \tag{2}$$

The time complexity is now

$$N \times 2^{\sum_{j=\partial+1}^{m} L_{i_j}} = \frac{2 \ln 2 \sum_{j=\partial+1}^{m} L_{i_j}}{\varepsilon^2} \times 2^{\sum_{j=\partial+1}^{m} L_{i_j}} \tag{3}$$

for the basic algorithm. When $m - \partial > 1$, the technique presented in Section 3.2 for speeding-up the exhaustive search can be applied if g' can again be decomposed as the sum of two functions with disjoint variables, leading to the following time complexity

$$2^{\sum_{j=\partial+1}^{m} L_{i_j}} \left[\frac{2 \ln 2 \sum_{j=\partial+1}^{m} L_{i_j}}{T_u \varepsilon^2} + \log T_u \right]$$

where $T_u = \prod_{j=m-m'+1}^{m} T_{i_j}$ for some $m' \geq 0$.

4.2 Improving data complexity: using several parallel decimated sequences

Decimation usually increases the data complexity of the attack, and this might be a bottleneck, for instance when the number of keystream bits produced from a single initial state is limited. Moreover, it clearly appears that the attack does not exploit this high amount of keystream bits in an optimal way since the data complexity is usually much higher than the number of keystream bits used in the attack. Therefore, we may expect to find a different trade-off between data and time complexities when a decimated sequence is used.

The general problem is to determine the initial state which provides the highest value of

$$\sum_{t=0}^{N-1} (-1)^{z(tT_d+\delta)} = (-1)^{\gamma(\delta)} \sum_{t=0}^{N-1} (-1)^{D_\delta(t)} ,$$

by exploiting the fact that $\boldsymbol{d}_\delta$ depends on 2^{k_d} initial states only. As done in [36], instead of computing this sum for a single value of δ, we rather compute a vector of Δ integers,

$$(\mathcal{D}(\delta),\ 0 \leq \delta < \Delta) \text{ where } \mathcal{D}(\delta) = \sum_{t=0}^{N'-1} (-1)^{D_\delta(t)} ,$$

but for a smaller number N' of samples in each component of the vector. We are now faced with the same situation as in a linear attack using Matsui's algorithm 2 with

several independent approximations [1, 17, 18, 24]. The attacker computes 2^{k_d} vectors with independent components where each component follows the binomial distribution with parameters N' and $\frac{1}{2}(1 \pm \varepsilon)$ for the correct initial state, and with parameters N' and $\frac{1}{2}$ otherwise. Since the sign of the bias of each $\mathcal{D}(\delta)$ is unknown, we have to perform an exhaustive search for this sign and compare the empirical probability distribution for the vector $((-1)^{D_\delta(t)})_{0\le\delta<\Delta}$ with the theoretical distribution

$$p_b(x) = 2^{-\Delta} \prod_{0\le\delta<\Delta} \left(1 + (-1)^{b_\delta \oplus x_\delta} \varepsilon\right), \ x \in \mathbf{F}_2^\Delta$$

for all Δ-bit vectors b. Several statistical tests have been proposed for comparing both distributions [1, 23]. For instance, it has been proposed in [1] to sort the possible initial states depending on the value of

$$\min_{b\in\mathbf{F}_2^\Delta} \sum_{\delta=0}^{\Delta-1} \left(\mathcal{D}(\delta) - (-1)^{b_\delta}\varepsilon\right)^2 .$$

Once the values of $\mathcal{D}(\delta)$ have been computed, the additional time complexity of the algorithm is then $\Delta 2^\Delta$ for each of the 2^{k_d} initial states. The overall time complexity is then

$$2^{k_d}(\Delta N' + \Delta 2^\Delta) ,$$

implying that the overhead is usually negligible compared to the algorithm with a single decimated sequence. This complexity can be improved if the Δ correlations $\mathcal{D}(\delta)$ are computed with the faster algorithm described in Section 3.

Then, it has been proved in [17, Proposition 3.1] that, if $\Delta\varepsilon^2 \ll 1$, the number of samples N' required for determining the correct candidate with the same error probability as for the classical attack is

$$N' = \frac{2k_d \ln 2}{\Delta\varepsilon^2} .$$

In other words, the number of required samples is divided by the number of decimated sequences which are considered. The data complexity then decreases to

$$N'T_d + \Delta = \frac{2k_d \ln 2 T_d}{\Delta\varepsilon^2} + \Delta$$

while the time complexity, equal to

$$2^{k_d}(\Delta N' + \Delta 2^\Delta) = \left(\frac{2\ln 2k_d}{\varepsilon^2} + \Delta 2^\Delta\right) \times 2^{k_d} ,$$

has a negligible overhead. The general algorithm combining the decimation technique and the speeding-up method described in Section 3 then applies when $\boldsymbol{z} = \boldsymbol{u} \oplus \boldsymbol{v} \oplus \boldsymbol{\gamma}$ where these three sequences have independent initial states of respective sizes k_u, k_v and k_γ and where $\boldsymbol{u}$ and $\boldsymbol{\gamma}$ are periodic with respective periods T_u and T_d. The algorithm is then described in Algorithm 2 when N' is the number of needed samples for each decimated sequence, i.e.

$$N' = \frac{2(k_u + k_v)\ln 2}{\Delta\varepsilon^2} .$$

Algorithm 2 Correlation attack using several decimated sequences.

for each initial state of $\boldsymbol{v}$ **do**
  for δ from 0 to $\Delta - 1$ **do**
    for r from 0 to $T_u - 1$ **do**
      $V_\delta(r) \leftarrow \frac{N'}{T_u} - 2\sum_{q=0}^{N'/T_u-1} v((qT_u + r)T_d + \delta)$
    end for
  end for
  repeat
    choose an initial state of $\boldsymbol{u}$ which does not belong to a previously examined cycle
    for δ from 0 to $\Delta - 1$ **do**
      Compute the following cross-correlation with an FFT

$$\mathcal{D}_\delta(\tau) = \sum_{r=0}^{T_u-1} (-1)^{u((r+\tau \bmod T_u)T_d+\delta)} V_\delta(r), \quad 0 \le \tau < T_u$$

    end for
    for τ from 0 to $T_u - 1$ **do**
      Compute

$$\min_{b \in \mathbf{F}_2^\Delta} \sum_{\delta=0}^{\Delta-1} \left(\mathcal{D}_\delta(\tau) - (-1)^{b_\delta}\varepsilon\right)^2$$

      if $\mathcal{S}(\tau) >$ threshold **then**
        return the initial state of $\boldsymbol{v}$ and the internal state of $\boldsymbol{u}$ after τ clocks.
      end if
    end for
  until all initial states of $\boldsymbol{u}$ have been examined
end for

The corresponding data complexity is then

$$T_d N' + \Delta = \frac{2(k_u + k_v)T_d \ln 2}{\Delta\varepsilon^2} + \Delta$$

and the time complexity is

$$2^{k_u+k_v}\left(\frac{\Delta N'}{T_u} + \Delta \log T_u + \Delta 2^\Delta\right).$$

Example In the attack against Achterbahn-80 [14], we have to find the initial state of a sequence derived from the keystream, of the form

$$\boldsymbol{S} = f(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_6, \boldsymbol{x}_8, \boldsymbol{x}_9, \boldsymbol{x}_{10}, \boldsymbol{x}_{11})$$

where each $\boldsymbol{x}_i$ is the output of a nonlinear device of length $L_i = 21 + i$ and with period $2^{L_i} - 1$. In [36], Naya Plasencia used an approximation of $\boldsymbol{S}$ of the form $\boldsymbol{\sigma} = \boldsymbol{x}_1 \oplus \boldsymbol{x}_6 \oplus \boldsymbol{x}_{10}$ which satisfies $\mathcal{E}(\boldsymbol{S} \oplus \boldsymbol{\sigma}) = 2^{-24}$. Then, we need to find the initial state of $\boldsymbol{z} = \boldsymbol{S} \oplus \boldsymbol{\sigma}$, and we decompose it as $\boldsymbol{z} = \boldsymbol{u} \oplus \boldsymbol{v} \oplus \boldsymbol{\gamma}$ with $\boldsymbol{\gamma} = \boldsymbol{x}_1$ which has

period $T_d = 2^{22} - 1$, $\boldsymbol{u} = \boldsymbol{x}_6$ which has $2^{k_u} = 2^{27}$ initial states and period $T_u = 2^{27} - 1$, and $\boldsymbol{v} = \boldsymbol{S} \oplus \boldsymbol{x}_{10}$. Since the keystream length for a given initial state is limited to 2^{52}, we can apply the previous algorithm with $\Delta = 4$. From the previous formulae, we have that each of the four decimated sequences needs to be evaluated in $N' = 2^{28.3}$ positions, implying that the data complexity is $2^{50.3}$. The time complexity is 2^{65}.

5 Correlation attacks with parity-check relations

A correlation attack can be seen as a decoding problem where the initial state of $\boldsymbol{z}$ is recovered by an ML-decoding algorithm. Since the time complexity of ML-decoding is usually too high, a well-known strategy for decoding linear codes consists in exploiting parity-check relations, i.e., linear relations between some bits of the codewords, especially sparse parity-check relations which usually make the decoding much faster. The price to pay for this is that the decoding is less efficient in the sense that more redundancy is needed. In other words, the data complexity increases. This idea has been introduced by Meier and Staffelbach and is at the origin of the so-called *fast correlation attacks* against LFSR-based generators [34]. Actually, in the case of the combination generator based on LFSRs, when the small generator producing $\boldsymbol{\sigma}$ is linear, many sparse parity-check relations for $\boldsymbol{\sigma}$ can be derived from the LFSR feedback polynomials or from their sparse multiples. This high number of relations then allows the attacker to recover its initial state. Many variants of this attack have been proposed, e.g. [5–7, 25–27, 34]. When the constituent devices are nonlinear, the number of parity-check relations is much smaller, implying that this type of attack would require a huge data complexity. A small number of linear relations can nevertheless be exploited in a distinguishing attack, as proposed by [8, 12]. The following two sections describe how such an attack can be performed against the general combination generator, by using some relations derived from the periods of the devices as first proposed in [28]. It can also be combined with an exhaustive search for the initial state of some of the devices in order to eliminate the influence of a part of the constituent devices and then reduce the time complexity. Then, we show in Section 5.3 that combining those two attacks leads to key-recovery with a good trade-off between time and data complexities.

5.1 Parity-check relations

A *parity-check relation* for a binary sequence $\boldsymbol{z} = (z(t))_{t\geq 0}$ is a linear relation between some bits of $\boldsymbol{z}$ at different instants $(t + \tau)$ where τ varies in a fixed set $\mathcal{T}$ of integers, and t takes any value:

$$\bigoplus_{\tau\in\mathcal{T}} z(t+\tau) = 0, \quad \forall t \geq 0.$$

For instance, the indexes τ corresponding to the nonzero coefficients of the characteristic polynomial of a linear recurring sequence provide a parity-check relation. A two-term parity-check relation,

$$z(t) \oplus z(t+\tau) = 0, \quad \forall t \geq 0,$$

obviously means that τ is a period of the sequence.

In the case of the combination generator, if $\boldsymbol{\sigma}$ is produced by combining devices $i_1, \ldots, i_m$ by some function g, a two-term parity-check relation for $\boldsymbol{\sigma}$ is given by

$$\sigma(t) \oplus \sigma\left(t + \prod_{j=1}^{m} T_{i_j}\right) = 0, \quad \forall t \geq 0,$$

but it can only be used if the attacker has access to keystream bits at distance $\prod_{j=1}^{m} T_{i_j}$ from each other, which is usually impossible. Then, Johansson et al. [28] have suggested to reduce the degree of the relation, i.e., the highest distance between two involved positions, by increasing the number of terms, as shown by the following simple proposition.

Proposition 1 *Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ be n sequences with periods $T_1, \ldots, T_n$. We denote*

$$\mathcal{T} = \langle T_1, \ldots, T_n \rangle = \left\{ \sum_{i=1}^{n} c_i T_i, \quad c_i \in \{0, 1\} \right\} .$$

Then, the binary sequence $\boldsymbol{x}$ defined by

$$x(t) = \bigoplus_{i=1}^{n} x_i(t)$$

satisfies

$$\bigoplus_{\tau \in \mathcal{T}} x(t + \tau) = 0, \quad \forall t \geq 0.$$

Proof We can prove that the influence of each sequence $\boldsymbol{x}_j$, $1 \leq j \leq n$, in the sum vanishes. Actually, the set $\mathcal{T}$ can be decomposed into two halves,

$$\mathcal{T}_j = \left\{ \sum_{i \in \{1,\ldots,n\} \setminus \{j\}} c_i T_i, \quad c_i \in \{0, 1\} \right\} \text{ and } T_j + \mathcal{T}_j ,$$

such that $x_j(t + \tau) = x_j(t + \tau + T_j)$ for any t and any $\tau \in \mathcal{T}_j$. Therefore, for any j, $1 \leq j \leq n$, we have

$$\bigoplus_{\tau \in \mathcal{T}} x_j(t + \tau) = \bigoplus_{\tau \in \mathcal{T}_j} \left(x_j(t + \tau) \oplus x_j(t + \tau + T_j) \right) = 0 .$$

□

Proposition 1, combined with the fact that the product of the periods of two sequences is a period for their sum, provides several trade-offs between the degree and the number of terms of a parity-check relation for $\boldsymbol{\sigma}$. For instance, if we consider the sequence $\boldsymbol{\sigma}$ defined by

$$\sigma(t) = x_1(t) \oplus x_2(t) \oplus x_3(t) ,$$

then, the following three relations are examples of parity-check relations for $\boldsymbol{\sigma}$ with different numbers of terms:

$$\begin{aligned}\sigma(t) \oplus \sigma(t + T_1T_2T_3) &= 0\\ \sigma(t) \oplus \sigma(t + T_1) \oplus \sigma(t + T_2T_3) \oplus \sigma(t + T_1 + T_2T_3) &= 0\\ \sigma(t) \oplus \sigma(t + T_1) \oplus \sigma(t + T_2) \oplus \sigma(t + T_1 + T_2) \oplus&\\ \sigma(t + T_3) \oplus \sigma(t + T_1 + T_3) \oplus \sigma(t + T_2 + T_3) \oplus \sigma(t + T_1 + T_2 + T_3) &= 0\,.\end{aligned}$$

Now, if $\boldsymbol{\sigma}$ is correlated to the keystream $\boldsymbol{S}$, then any parity-check relation for $\boldsymbol{\sigma}$ provides a biased linear relation for the keystream. Actually, for any set $\mathcal{T}$ such that $\bigoplus_{\tau\in\mathcal{T}} \sigma(t+\tau) = 0$ for all $t \geq 0$, we have

$$\bigoplus_{\tau\in\mathcal{T}} S(t+\tau) = \bigoplus_{\tau\in\mathcal{T}} S(t+\tau) \oplus \bigoplus_{\tau\in\mathcal{T}} \sigma(t+\tau) = \bigoplus_{\tau\in\mathcal{T}} (S\oplus\sigma)(t+\tau)\,.$$

Since the sequence $(\boldsymbol{S} \oplus \boldsymbol{\sigma})$ is biased with bias $\mathcal{E}(f+g)$ where g is the combining function of the small generator producing $\boldsymbol{\sigma}$, then it can be proved that the corresponding parity-check relation applied to $(\boldsymbol{S} \oplus \boldsymbol{\sigma})$ is also biased but with a smaller bias. It is worth noticing that the bias of the parity-check relation cannot be directly derived from the piling-up lemma since the terms in the sum are not statistically independent [19, 21, 35]. Moreover, there might exist two different approximations g and g' of the combining function f such that, for the same $\mathcal{T}$, we have

$$\bigoplus_{\tau\in\mathcal{T}} g(x_{i_1}(t+\tau),\ldots,x_{i_m}(t+\tau)) = 0 \text{ and } \bigoplus_{\tau\in\mathcal{T}} g'(x_{j_1}(t+\tau),\ldots,x_{j_{m'}}(t+\tau)) = 0,\ \forall t \geq 0\,.$$

In this case, the bias of the relation applied to the keystream,

$$\bigoplus_{\tau\in\mathcal{T}} f(x_1(t+\tau),\ldots,x_n(t+\tau))\,,$$

cannot be directly deduced from both biases $\mathcal{E}(f+g)$ and $\mathcal{E}(f+g')$. However, the following lower bound on the bias of the parity-check relation on the keystream has been exhibited in [4].

Theorem 1 [4, *Theorem* 5] *Let $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n$ be n sequences with least periods $T_1,\ldots,T_n$, f a Boolean function of n variables and $\boldsymbol{S} = f(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n)$. Let $\kappa_1,\ldots,\kappa_{s+1}$ be a strictly increasing sequence of integers with $\kappa_1 = 0$ and $\kappa_{s+1} = m$. Let*

$$\mathcal{T} = \langle M_1,\ldots,M_s\rangle$$

where $M_i = q_i\mathrm{lcm}(T_{\kappa_i+1},\ldots,T_{\kappa_{i+1}})$ for some integer $q_i > 0$. Assume that each M_i is coprime with all T_j with $j \notin [\kappa_i+1;\kappa_{i+1}]$. Let $\boldsymbol{PC}_{f,\mathcal{T}}$ be the sequence defined by

$$PC_{f,\mathcal{T}}(t) = \bigoplus_{\tau\in\mathcal{T}} s(t+\tau),\ \forall t \geq 0\,.$$

Then, for any Boolean function g of m variables of the form

$$g(x_1,\ldots,x_m) = \bigoplus_{i=1}^{s} g_i(x_{\kappa_i+1},\ldots,x_{\kappa_{i+1}})$$

where each g_i is a Boolean function of $(\kappa_{i+1} - \kappa_i)$ variables, we have

$$\mathcal{E}(\boldsymbol{PC}_{f,\mathcal{T}}) \geq \left[\mathcal{E}(f \oplus g)\right]^{2^s} .$$

In the following, we focus on sets $\mathcal{T}$ of the form

$$\mathcal{T} = \langle M_1, \ldots, M_s \rangle \tag{4}$$

where each M_i equals some T_{i_j} or the product of several T_{i_j} (possibly with a nonzero multiplicative factor) as defined in Theorem 1, and we will assume for the sake of simplicity that all T_{i_j} are coprime. If $\mathcal{T}$ involves all periods T_{i_j}, $1 \leq j \leq m$, then we have that

$$\mathcal{E}(\boldsymbol{PC}_{f,\mathcal{T}}) \geq \left[\mathcal{E}(f \oplus \ell)\right]^{2^s} ,$$

with $\ell = \bigoplus_{j=1}^{m} x_{i_j}$. Moreover, if $m = R + 1$ where R is the resiliency order of f, which is the usual case in practice, then this lower bound is tight [4, Theorem 12]:

$$\mathcal{E}(\boldsymbol{PC}_{f,\mathcal{T}}) = \left[\mathcal{E}(f \oplus \ell)\right]^{2^s} .$$

Therefore, this bias can be exploited for distinguishing the keystream from a random sequence.

5.2 Distinguishing attacks based on parity-check relations

The distinguishing attack consists in computing the biased sequence

$$PC_{f,\mathcal{T}}(t) = \bigoplus_{\tau \in \mathcal{T}} S(t + \tau), \ \forall t \geq 0$$

from the keystream, where $\mathcal{T}$ is defined as specified by (4). For instance, for $m = R + 1$, a natural choice for $\mathcal{T}$ is

$$\mathcal{T} = \langle T_{i_1}, \ldots, T_{i_m} \rangle .$$

Then, the attacker applies a hypothesis test in order to determine whether the computed sequence has the expected bias or not. The number of samples of the parity-check relation which are needed for detecting the bias is given by

$$N \simeq \frac{2 \ln 2}{\mathcal{E}(\boldsymbol{PC}_{f,\mathcal{T}})^2} \leq \frac{2 \ln 2}{\varepsilon^{2^{m+1}}} \tag{5}$$

where $\varepsilon = \mathcal{E}(f + \ell)$ with $\ell = \bigoplus_{j=1}^{m} x_{i_j}$. As previously discussed, this formula provides an upper bound in the general case, but it is tight for $m = R + 1$. It is worth noticing that the lower bound on $\mathcal{E}(\boldsymbol{PC}_{f,\mathcal{T}})$ implies that this bias is always positive. Therefore, the statistical test aims at maximizing the value of

$$\sum_{t=0}^{N-1} (-1)^{PC_{f,\mathcal{T}}(t)}$$

or equivalently, at minimizing $\sum_{t=0}^{N-1} PC_{f,\mathcal{T}}(t)$.

When $\mathcal{T} = \langle T_{i_1}, \ldots, T_{i_m} \rangle$, the number of keystream bits needed for the distinguishing attack is equal to

$$N + \sum_{j=1}^{m} T_{i_j} \leq \frac{2 \ln 2}{\varepsilon^{2^{m+1}}} + \sum_{j=1}^{m} T_{i_j} \ .$$

The corresponding time complexity is then

$$2^m N \leq \frac{2^{m+1} \ln 2}{\varepsilon^{2^{m+1}}}$$

where equality holds in both formulae when $m = R + 1$. The attack may then be faster than the classical correlation attack, but it has a higher data complexity.

Moreover, it does not allow the initial state of the keystream generator to be recovered.

5.3 Combining both techniques

Much more appropriate trade-offs between time and data complexity can therefore be obtained by combining both attacks. Let us consider m_1 constituent devices, namely $\mathcal{R}_{i_1}, \ldots, \mathcal{R}_{i_{m_1}}$, whose influences will be cancelled by the computation of a parity-check relation. Let ℓ denote the linear function $\ell = \bigoplus_{j=1}^{m_1} x_{i_j}$. Then, this set of m_1 devices must be chosen such that there exists a biased approximation g of $(f + \ell)$, depending only on the $(m - m_1)$ input variables with indexes $i_{m_1+1}, \ldots, i_m$. The most appropriate set of parameters in many situations is given by $m = R + 1$ and $g = \bigoplus_{j=m_1+1}^{m} x_{i_j}$.

The first step of the attack consists in computing the following parity-check relation on the keystream sequence:

$$PC_{f,\mathcal{T}}(t) = \bigoplus_{\tau \in \mathcal{T}} S(t + \tau), \ \forall t \geq 0$$

with $\mathcal{T} = \langle T_{i_1}, \ldots, T_{i_{m_1}} \rangle$.

Then, for each possible initial state of the $(m - m_1)$ devices $\mathcal{R}_{i_{m_1+1}}, \ldots, \mathcal{R}_{i_m}$, a sequence $\boldsymbol{\sigma}$ is computed by

$$\sigma(t) = g(x_{i_{m_1+1}}(t), \ldots, x_{i_m}(t)) \ .$$

The parity-check relation

$$PC_{g,\mathcal{T}}(t) = \bigoplus_{\tau \in \mathcal{T}} \sigma(t + \tau)$$

is then evaluated. If the guessed initial state is correct, then the sequences $\boldsymbol{PC_{f,\mathcal{T}}}$ and $\boldsymbol{PC_{g,\mathcal{T}}}$ are correlated. Actually, we have

$$PC_{f,\mathcal{T}}(t) \oplus PC_{g,\mathcal{T}}(t) = PC_{f,\mathcal{T}}(t) \oplus PC_{g,\mathcal{T}}(t) \oplus PC_{\ell,\mathcal{T}}(t) = PC_{f+g+\ell,\mathcal{T}}(t) \ .$$

The corresponding bias is $\mathcal{E}(\boldsymbol{PC_{f+g+\ell,\mathcal{T}}})$ which is greater than or equal to $\varepsilon^{2^{m_1}}$ with $\varepsilon = \mathcal{E}(f + g + \ell)$. Then, a correlation attack can be performed in order to detect a correlation between $\boldsymbol{PC_{f,\mathcal{T}}}$, which is derived from the keystream, and $\boldsymbol{PC_{g,\mathcal{T}}}$ which is computed for each possible initial state of the $(m - m_1)$ targeted devices.

Recovering the correct initial state among the $(2^{\sum_{j=m_1+1}^{m} L_{i_j}} - 1)$ sequences then requires

$$N \simeq \frac{2 \ln 2 \sum_{j=m_1+1}^{m} L_{i_j}}{\varepsilon^{2^{m_1+1}}} \text{ samples },$$

leading to the following data complexity

$$\frac{2 \ln 2 \sum_{j=m_1+1}^{m} L_{i_j}}{\varepsilon^{2^{m_1+1}}} + \sum_{j=1}^{m_1} T_{i_j} . \tag{6}$$

The time complexity is now

$$2^{m_1} N \times 2^{\sum_{j=m_1+1}^{m} L_{i_j}} = \frac{2^{m_1+1} \ln 2 \sum_{j=m_1+1}^{m} L_{i_j}}{\varepsilon^{2^{m_1+1}}} \times 2^{\sum_{j=m_1+1}^{m} L_{i_j}} \tag{7}$$

for the basic algorithm described by Algorithm 3. It must be noticed that the time complexity is independent from the periods and the lengths of devices $\mathcal{R}_{i_1}, \ldots, \mathcal{R}_{i_{m_1}}$. Obviously, for a given value of m, increasing the number $(m - m_1)$ of devices for which we perform an exhaustive search allows the data complexity to be reduced. It may increase the time complexity, but this is not always the case since the expression (7) for the time complexity consists of the product of two terms, one increasing with $(m - m_1)$ and the second one depending on N, which decreases when m_1 decreases. Therefore, the optimal choice for the parameters highly depends on the size of the devices and on the bias of the approximation. Finding the best trade-off between both terms is then an important task.

Algorithm 3 Correlation attack combining exhaustive search and parity-check relations.

for each t from 0 to $(N-1)$ **do**
  $PC_{f,\mathcal{T}}(t) \leftarrow \bigoplus_{\tau \in \mathcal{T}} S(t+\tau)$
end for
for each initial state of the devices $i_{m_1+1}, \ldots, i_m$ **do**
  $c \leftarrow 0$
  for each t from 0 to $(N-1)$ **do**
    $PC_{g,\mathcal{T}}(t) \leftarrow \bigoplus_{\tau \in \mathcal{T}} g(x_{i_{m_1}+1}(t+\tau), \ldots, x_{i_m}(t+\tau))$
    $c \leftarrow c + (PC_{f,\mathcal{T}}(t) \oplus PC_{g,\mathcal{T}}(t))$
  end for
  if $c >$ threshold **then**
    return the initial states of the $(m - m_1)$ targeted devices.
  end if
end for

Obviously, when $m - m_1 > 1$, the highest value of the correlation between $\boldsymbol{PC_{f,\mathcal{T}}}$ and $\boldsymbol{PC_{g,\mathcal{T}}}$ can be identified faster with Algorithm 2.

The general technique then consists in identifying m_1 devices for building the parity-check relations. Then, we search for an approximation g of $f + \ell$ with bias ε where ℓ is the sum of the m_1 variables involved in the parity-check relations. We decompose g into three functions with disjoint input variables:

$$g(x_{i_{m_1}+1}, \ldots, x_{i_m}) = g_d(x_{i_{m_1+1}}, \ldots, x_{i_{m_1+\partial}}) + g_u(x_{i_{m_1+\partial+1}}, \ldots, x_{i_{m'}}) + g_v(x_{i_{m'+1}}, \ldots, x_{i_m}) . \tag{8}$$

Table 1 Data and time complexities of all variants of the correlation attack.

	m_1	∂	m'	g	T_d	T_u	k	Data complexity	Time complexity
Basic	0	0	0	approx. of f of m var.	1	1	$\sum_{j=1}^{m} L_{i_j}$	$\frac{2k\ln 2}{\varepsilon^2}$	$2^k \times \frac{2k\ln 2}{\varepsilon^2}$
Algo 1	0	0	m'	approx. of f of m var.	1	$\prod_{j=1}^{m'} T_{i_j}$	$\sum_{j=1}^{m} L_{i_j}$	$\frac{2k\ln 2}{\varepsilon^2}$	$2^k \times \left(\frac{2k\ln 2}{T_u\varepsilon^2} + \log T_u\right)$
Algo 2	0	∂	m'	approx. of f of m var.	$\prod_{j=1}^{\partial} T_{i_j}$	$\prod_{j=\partial+1}^{m'} T_{i_j}$	$\sum_{j=\partial+1}^{m} L_{i_j}$	$\frac{2kT_d\ln 2}{\Delta\varepsilon^2} + \Delta$	$2^k \times \left(\frac{2k\ln 2}{T_u\varepsilon^2} + \Delta\log T_u + \Delta 2^{\Delta}\right)$
Section 5.2	m	0	0	$\sum_{j=1}^{m} x_{i_j}$	1	1	0	$\frac{2\ln 2}{\varepsilon^{2^{m+1}}} + \sum_{j=1}^{m} T_{i_j}$	$2^m \times \frac{2\ln 2}{\varepsilon^{2^{m+1}}}$
Algo 3	m_1	0	0	approx. of $\left(f \oplus \sum_{j=1}^{m_1} x_{i_j}\right)$ of $(m-m_1)$ var.	1	1	$\sum_{j=m_1+1}^{m} L_{i_j}$	$\frac{2k\ln 2}{\varepsilon^{2^{m_1+1}}} + \sum_{j=1}^{m_1} T_{i_j}$	$2^{k+m_1} \times \frac{2k\ln 2}{\varepsilon^{2^{m_1+1}}}$
Algos 3 + 1	m_1	0	m'	approx. of $\left(f \oplus \sum_{j=1}^{m_1} x_{i_j}\right)$ of $(m-m_1)$ var.	1	$\prod_{j=m_1+1}^{m'} T_{i_j}$	$\sum_{j=m_1+1}^{m} L_{i_j}$	$\frac{2k\ln 2}{\varepsilon^{2^{m_1+1}}} + \sum_{j=1}^{m_1} T_{i_j}$	$2^{k+m_1} \times \left(\frac{2k\ln 2}{T_u\varepsilon^{2^{m_1+1}}} + \log T_u\right)$
General	m_1	∂	m'	approx. of $\left(f \oplus \sum_{j=1}^{m_1} x_{i_j}\right)$ of $(m-m_1)$ var.	$\prod_{j=m_1+1}^{m_1+\partial} T_{i_j}$	$\prod_{j=m_1+\partial+1}^{m'} T_{i_j}$	$\sum_{j=m_1+\partial+1}^{m} L_{i_j}$	$\frac{2kT_d\ln 2}{\Delta\varepsilon^{2^{m_1+1}}} + \sum_{j=1}^{m_1} T_{i_j} + \Delta$	$2^{k+m_1} \times \left(\frac{2k\ln 2}{T_u\varepsilon^{2^{m_1+1}}} + \Delta\log T_u + \Delta 2^{\Delta}\right)$

Let

$$T_d = \prod_{j=m_1+1}^{m_1+\partial} T_{i_j}, \quad T_u = \prod_{j=m_1+\partial+1}^{m'} T_{i_j} \text{ and } k = \sum_{j=m_1+\partial+1}^{m} L_{i_j},$$

where 2^k is the number of initial states for the devices involved in both approximations g_u and g_v. Then, we need to evaluate the correlation for each of the Δ decimated sequences from

$$N' = \frac{2k \ln 2}{\Delta \varepsilon^{2^{m_1+1}}} \text{ samples.}$$

The corresponding data complexity is then

$$T_d N' + \sum_{j=1}^{m_1} T_{i_j} + \Delta = \frac{2T_d k \ln 2}{\Delta \varepsilon^{2^{m_1+1}}} + \sum_{j=1}^{m_1} T_{i_j} + \Delta \text{ keystream bits.}$$

The time complexity is

$$2^k 2^{m_1} \left(\frac{\Delta N'}{T_u} + \Delta \log T_u + \Delta 2^{\Delta} \right).$$

As extremal cases, we recover the time and data complexities of the correlation attacks presented in the previous sections. More precisely, Table 1 describes all variants of the attack. The number of variables m can take any value between 1 and $(n-1)$, while the only requirement on (m_1, ∂, m') is that the involved approximation g can be decomposed as (8).

6 Conclusions

In this paper we have successfully generalised the five correlation attacks [20, 21, 28, 35, 36] presented to analyse the successive versions of the combination generator based on NLFSRs, Achterbahn. We have also showed that some of these improvements apply to a more general problem which is encountered in some other contexts in cryptography. In the context of the general combination generator, we have defined a whole family of correlation attacks using several additional ideas against this type of cipher that provides different time-data-memory trade-offs. These are the best known attacks for the considered construction. We have provided general formulas for computing accurate complexity estimates in each case. This allows to find the optimal attack in each particular case. We hope that this work will help future designers to know a priori how the parameters of the ciphers need to be chosen for being resistant to such attacks, as well as will permit the cryptanalysts to apply in an automatic way these attacks. We believe that this generalisation of the attacks proposed against Achterbahn will provide a better understanding, which is very important for some other possible uses and for finding potential future improvements.

References

1. Biryukov, A., De Cannière, C., Quisquater, M.: On multiple linear approximations. In: Advances in Cryptology—CRYPTO 2004. Lecture Notes in Computer Science, vol. 3152, pp. 1–22. Springer, Heidelberg (2004)
2. Blahut, R.E.: Fast Algorithms for Digital Signal Processing. Addison Wesley (1985)
3. Canteaut, A., Filiol, E.: Ciphertext only reconstruction of stream ciphers based on combination generators. In: Fast Software Encryption—FSE 2000. Lecture Notes in Computer Science, vol. 1978, pp. 165–180. Springer-Verlag (2001)
4. Canteaut, A., Naya-Plasencia, M.: Parity-check relations on combination generators. IEEE Trans. Inf. Theory **58**(6), 3900–3911 (2012)
5. Canteaut, A., Trabbia, M.: Improved fast correlation attacks using parity-check equations of weight 4 and 5. In: Advances in Cryptology—EUROCRYPT 2000. Lecture Notes in Computer Science, vol. 1807, pp. 573–588. Springer-Verlag (2000)
6. Chepyshov, V., Johansson, T., Smeets, B.: A simple algorithm for fast correlation attacks on stream ciphers. In: Fast Software Encryption—FSE 2000, Lecture Notes in Computer Science, vol. 1978, pp. 124–135. Springer-Verlag (2000)
7. Chose, P., Joux, A., Mitton, M.: Fast correlation attacks: an algorithmic point of view. In: Advances in Cryptology—EUROCRYPT 2002. Lecture Notes in Computer Science, vol. 2332, pp. 209–221. Springer-Verlag (2002)
8. Coppersmith, D., Halevi, S., Jutla, C.: Cryptanalysis of stream ciphers with linear masking. In: Advances in Cryptology—CRYPTO 2002. Lecture Notes in Computer Science, vol. 2442. Springer-Verlag (2002)
9. Courtois, N.: Fast algebraic attacks on stream ciphers with linear feedback. In: Advances in Cryptology—CRYPTO 2003. Lecture Notes in Computer Science, vol. 2729, pp. 176–194. Springer-Verlag (2003)
10. Courtois, N., Meier, W.: Algebraic attacks on stream ciphers with linear feedback. In: Advances in Cryptology—EUROCRYPT 2003. Lecture Notes in Computer Science, vol. 2656, pp. 345–359. Springer-Verlag (2003)
11. ECRYPT—European Network of Excellence in Cryptology: The eSTREAM Stream Cipher Project. http://www.ecrypt.eu.org/stream/ (2004)
12. Ekdahl, P., Johansson, T.: Distinguishing attacks on SOBER-t16 and t32. In: Fast Software Encryption—FSE 2002. LNCS, vol. 2365, pp. 210–224. Springer (2002)
13. Gammel, B., Göttfert, R., Kniffler, O.: The Achterbahn stream cipher. Submission to eSTREAM. http://www.ecrypt.eu.org/stream/ (2005)
14. Gammel, B., Göttfert, R., Kniffler, O.: Achterbahn-128/80. Submission to eSTREAM. http://www.ecrypt.eu.org/stream/ (2006)
15. Gammel, B., Göttfert, R., Kniffler, O.: Status of Achterbahn and Tweaks. In: Proceedings of SASC 2006—Stream Ciphers Revisited. http://www.ecrypt.eu.org/stream/papersdir/2006/027.pdf (2006)
16. Gammel, B., Göttfert, R., Kniffler, O.: Achterbahn-128/80: design and analysis. In: Proceedings of SASC 2007—Stream Ciphers Revisited. http://www.ecrypt.eu.org/stream/papersdir/2007/020.pdf (2007)
17. Gérard, B., Tillich, J.P.: On linear cryptanalysis with many linear approximations. In: IMA International Conference, Cryptography and Coding. Lecture Notes in Computer Science, vol. 5921, pp. 112–132. Springer (2009)
18. Gérard, B., Tillich, J.P.: Advanced Linear Cryptanalysis of Block and Stream Ciphers, vol. 7, chap. Using Tools from Error Correcting Theory in Linear Cryptanalysis, pp. 87–114. IOS Press (2011)
19. Göttfert, R., Gammel, B.: On the frame length of Achterbahn-128/80. In: Proceedings of the 2007 IEEE Information Theory Workshop on Information Theory for Wireless Networks, pp. 1–5. IEEE (2007)
20. Hell, M., Johansson, T.: Cryptanalysis of Achterbahn-Version 2. In: Selected Areas in Cryptography—SAC 2006. Lecture Notes in Computer Science, vol. 4356, pp. 45–55. Springer (2006)
21. Hell, M., Johansson, T.: Cryptanalysis of Achterbahn-128/80. IET Inf. Secur. **1**(2), 47–52 (2007)
22. Hell, M., Johansson, T., Brynielsson, L.: An overview of distinguishing attacks on stream ciphers. Cryptogr. Commun. **1**(1), 71–94 (2009)

23. Hermelin, M., Cho, J., Nyberg, K.: Multidimensional extension of Matsui's Algorithm 2. In: Fast Software Encryption—FSE 2009. Lecture Notes in Computer Science, vol. 5665, pp. 209–227. Springer (2009)
24. Hermelin, M., Nyberg, K.: Advanced Linear Cryptanalysis of Block and Stream Ciphers, vol. 7, chap. Linear Cryptanalysis Using Multiple Linear Approximations, pp. 25–54. IOS Press (2011)
25. Johansson, T., Jönsson, F.: Fast correlation attacks based on turbo code techniques. In: Advances in Cryptology—CRYPTO'99. Lecture Notes in Computer Science, vol. 1666, pp. 181–197. Springer-Verlag (1999)
26. Johansson, T., Jönsson, F.: Improved fast correlation attack on stream ciphers via convolutional codes. In: Advances in Cryptology—EUROCRYPT'99. Lecture Notes in Computer Science, vol. 1592, pp. 347–362. Springer-Verlag (1999)
27. Johansson, T., Jönsson, F.: Fast correlation attacks through reconstruction of linear polynomials. In: Advances in Cryptology—CRYPTO'00. Lecture Notes in Computer Science, vol. 1880, pp. 300–315. Springer-Verlag (2000)
28. Johansson, T., Meier, W., Muller, F.: Cryptanalysis of Achterbahn. In: Fast Software Encryption—FSE 2006, Lecture Notes in Computer Science, vol. 4047, pp. 1–14. Springer (2006)
29. Joux, A.: Algorithmic Cryptanalysis. Chapman & Hall/CRC (2009)
30. Junod, P., Vaudenay, S.: Optimal key ranking procedures in a statistical cryptanalysis. In: Fast Software Encryption—FSE 2003. Lecture Notes in Computer Science, vol. 2887, pp. 235–246. Springer-Verlag (2003)
31. Lu, Y., Vaudenay, S.: Faster correlation attack on Bluetooth keystream generator E0. In: Advances in Cryptology—CRYPTO 2004. Lecture Notes in Computer Science, vol. 3152, pp. 407–425. Springer-Verlag (2004)
32. Matsui, M.: The first experimental cryptanalysis of the data encryption standard. In: Advances in Cryptology—CRYPTO'94. Lecture Notes in Computer Science, vol. 839. Springer-Verlag (1995)
33. Meier, W., Staffelbach, O.: Fast correlation attacks on stream ciphers. In: Advances in Cryptology—EUROCRYPT'88. Lecture Notes in Computer Science, vol. 330, pp. 301–314. Springer-Verlag (1988)
34. Meier, W., Staffelbach, O.: Fast correlation attack on certain stream ciphers. J. Cryptol. **1**(3), 159–176 (1989)
35. Naya-Plasencia, M.: Cryptanalysis of Achterbahn-128/80. In: Fast Software Encryption—FSE 2007. Lecture Notes in Computer Science, vol. 4593, pp. 73–86. Springer (2007)
36. Naya-Plasencia, M.: Cryptanalysis of Achterbahn-128/80 with a new keystream limitation. In: WEWoRC 2007—Second Western European Workshop in Research in Cryptology. Lecture Notes in Computer Science, vol. 4945, pp. 142–152. Springer (2008)
37. Siegenthaler, T.: Correlation-immunity of nonlinear combining functions for cryptographic applications. IEEE Trans. Inf. Theory **30**(5), 776–780 (1984)
38. Siegenthaler, T.: Decrypting a class of stream ciphers using ciphertext only. IEEE Trans. Comput. **C-34**(1), 81–84 (1985)
39. Zhang, M.: Maximum correlation analysis of nonlinear combining functions in stream ciphers. J. Cryptol. **13**(3), 301–313 (2000)

Quantum Differential and Linear Cryptanalysis

Marc Kaplan[1,2], Gaëtan Leurent[3], Anthony Leverrier[3] and María Naya-Plasencia[3]

[1] LTCI, Télécom ParisTech, 23 avenue d'Italie, 75214 Paris CEDEX 13, France
[2] School of Informatics, University of Edinburgh,
10 Crichton Street, Edinburgh EH8 9AB, UK
marc.kaplan@telecom-paristech.fr
[3] Inria Paris, France
[anthony.leverrier,gaetan.leurent,maria.naya_plasencia]@inria.fr

Abstract. Quantum computers, that may become available one day, would impact many scientific fields, most notably cryptography since many asymmetric primitives are insecure against an adversary with quantum capabilities. Cryptographers are already anticipating this threat by proposing and studying a number of potentially *quantum-safe* alternatives for those primitives. On the other hand, symmetric primitives seem less vulnerable against quantum computing: the main known applicable result is Grover's algorithm that gives a quadratic speed-up for exhaustive search.
In this work, we examine more closely the security of symmetric ciphers against quantum attacks. Since our trust in symmetric ciphers relies mostly on their ability to resist cryptanalysis techniques, we investigate quantum cryptanalysis techniques. More specifically, we consider quantum versions of differential and linear cryptanalysis. We show that it is usually possible to use quantum computations to obtain a quadratic speed-up for these attack techniques, but the situation must be nuanced: we don't get a quadratic speed-up for all variants of the attacks. This allows us to demonstrate the following non-intuitive result: the best attack in the classical world does not necessarily lead to the best quantum one. We give some examples of application on ciphers LAC and KLEIN. We also discuss the important difference between an adversary that can only perform quantum computations, and an adversary that can also make quantum queries to a keyed primitive.

Keywords: Symmetric cryptography · Differential cryptanalysis · Linear cryptanalysis · Post-quantum cryptography · Quantum attacks · Block ciphers.

1 Introduction

Large quantum computers would have huge consequences in a number of scientific fields. Cryptography would certainly be dramatically impacted: for instance, Shor's factoring algorithm [Sho97] makes asymmetric primitives such as RSA totally insecure in a post-quantum world. Even if quantum computers are unlikely to become widely available in the next couple of years, the cryptographic community has decided to start worrying about this threat and to study its impact. One compelling reason for taking action is that even current pre-quantum long-term secrets are at risk as it seems feasible for a malicious organization to simply store all encrypted data until it has access to a quantum computer. This explains why post-quantum cryptosystems, based for instance on lattices or codes, have become a very hot topic in cryptology, and researchers are now concentrating their efforts in order to provide efficient alternatives that would resist quantum adversaries.

In this paper, we focus on symmetric cryptography, the other main branch of cryptography. Symmetric primitives also suffer from a reduced ideal security in the quantum world,

IACR Transactions on Symmetric Cryptology ISSN 2519-173X, Vol. 2016, No. 1, pp. 71–94
DOI:10.13154/tosc.v2016.i1.71-94

but this security reduction turns out to be much less drastic than for many asymmetric primitives. So far, the main quantum attack on symmetric algorithms follows from Grover's algorithm [Gro96] for searching an unsorted database of size N in $O(N^{1/2})$ time. It can be applied to any generic exhaustive key search, but merely offers a quadratic speed-up compared to a classical attack. Therefore, the current consensus is that key lengths should be doubled in order to offer the same security against quantum algorithms. This was one of the motivations to require a version of AES with a 256-bit key, that appears in the initial recommendations of the European PQCRYPTO project [ABB+15]:

> "Symmetric systems are usually not affected by Shor's algorithm, but they are affected by Grover's algorithm. Under Grover's attack, the best security a key of length n can offer is $2^{n/2}$, so AES-128 offers only 2^{64} post-quantum security. PQCRYPTO recommends thoroughly analyzed ciphers with 256-bit keys to achieve 2^{128} post-quantum security."

Doubling the key length is a useful heuristic, but a more accurate analysis is definitely called for. Unfortunately, little work has been done in this direction. Only recently, a few results have started to challenge the security of some symmetric cryptography constructions against quantum adversaries. In particular, some works have studied generic attacks against symmetric constructions, or attacks against modes of operations.

First, the quantum algorithm of Simon [Sim97], which is based on the quantum Fourier transform, has been used to obtain a quantum distinguisher for the 3-round Feistel cipher [KM10], to break the quantum version of the Even-Mansour scheme [KM12], and in the context of quantum related-key attacks [RS15]. More recently, the same quantum algorithm has been used to break widely used block cipher modes of operations for MACs and authenticated encryption [KLLNP16] (see also [SS16]). All these attacks have a complexity linear in the block size, and show that some constructions in symmetric cryptography are badly broken if an adversary can make quantum queries.

Kaplan [Kap14] has also studied the quantum complexity of generic meet-in-the-middle attacks for iterated block ciphers constructions. In particular, this work shows that having access to quantum devices when attacking double iteration of block ciphers can only reduce the time by an exponent 3/2, rather than the expected quadratic improvement from Grover's algorithm. In consequence, in stark contrast with classical adversaries, double iteration of block ciphers can restore the security against quantum adversaries.

These are important steps in the right direction, providing the quantum algorithms associated to some generic attacks on different constructions. These results also show that the situation is more nuanced than a quadratic speed-up of all classical attacks. Therefore, in order to get a good understanding of the actual security of symmetric cryptography constructions against quantum adversaries, we need to develop and analyze quantum cryptanalytic techniques. In particular, a possible approach to devise new quantum attacks is to *quantize* classical ones.

Security of symmetric key ciphers. While the security of crypto-systems in public key cryptography relies on the hardness of some well-understood mathematical problems, the security of symmetric key cryptography is more heuristic. Designers argue that a scheme is secure by proving its resistance against some particular attacks. This means that only cryptanalysis and security evaluations can bring confidence in a primitive. Even when a primitive has been largely studied, implemented and standardized, it remains vital to carry on with the cryptanalysis effort using new methods and techniques. Examples of standards that turned out to be non-secure are indeed numerous (MD5, SHA1, RC4...). Symmetric security and confidence are therefore exclusively based on this constant and challenging task of cryptanalysis.

Symmetric cryptanalysis relies on a toolbox of classical techniques such as differential or linear cryptanalysis and their variants, algebraic attacks, etc. A cryptanalyst can study the security of a cipher against those attacks, and evaluate the security margin of a design

using reduced-round versions. This security margin (how far the attack is from reaching all the rounds) is a good measure of the security of a design; it can be used to compare different designs and to detect whether a cipher is close to being broken.

Since the security of symmetric primitives relies so heavily on cryptanalysis, it is crucial to evaluate how the availability of quantum computing affects it, and whether dedicated attacks can be more efficient than brute-force attacks based on Grover's algorithm. In particular, we must design the toolbox of symmetric cryptanalysis in a quantum setting in order to understand the security of symmetric algorithms against quantum adversaries. In this paper, we consider quantum versions of cryptanalytic attacks for the first time[1], evaluating how an adversary can perform some of the main attacks on symmetric ciphers with a quantum computer.

Modeling quantum adversaries. Following the notions for PRF security in a quantum setting given by Zhandry [Zha12], we consider two different models for our analysis:

Standard security: a block cipher is *standard secure* against quantum adversaries if no efficient quantum algorithm can distinguish the block cipher from PRP (or a PRF) by making only *classical* queries (later denoted as Q1).

Quantum security: a block cipher is *quantum secure* against quantum adversaries if no efficient quantum algorithm can distinguish the block cipher from PRP (or a PRF) even by making *quantum* queries (later denoted as Q2).

A Q1 adversary collects data classically and processes them with quantum operations, while a Q2 adversary can directly query the cryptographic oracle with a quantum superposition of classical inputs, and receives the superposition of the corresponding outputs. The adversary, in the second model, is very powerful. Nevertheless, it is possible to devise secure protocols against these attacks. In particular, the model was used in [BZ13b], where quantum-secure signatures were introduced. Later, the same authors showed how to construct message authentification codes secure against Q2 adversaries [BZ13a]. It was also investigated in [DFNS13] for secret-sharing schemes. This model is also mathematically well defined, and it is convenient to use it to give security definitions against quantum adversaries, a task that is often challenging [GHS15]. A more practical issue is that even if the cryptographic oracle is designed to produce classical outcomes, its implementation may use some technology, for example optical fibers, that a quantum adversary could exploit. In practice, ensuring that only classical queries are allowed seems difficult, especially in a world in which quantum resources become available. It seems more promising to assume that security against quantum queries is not granted and to study security in this model.

Modes of operation. Block ciphers are typically used in a mode of operation, in order to accommodate messages of variable length and to provide a specific security property (confidentiality, integrity...). In classical cryptography, we prove that modes of operations are secure, assuming that the block cipher is secure, and we trust the block ciphers after enough cryptanalysis has been performed. We can do the same against quantum adversaries, but proofs of security in the classical model do not always translate to proofs of security in the quantum model. In particular, common MAC and AE modes secure in the classical model have recently been broken with a Q2 attack [KLLNP16]. On the other hand, common encryption modes have been proven secure in the quantum model [ATTU16], assuming either a standard-secure PRF or a quantum-secure PRF. In this work, we focus on the security of block ciphers, but this analysis should be combined with an analysis of the quantum security of modes of operation to get a full understanding of the security of symmetric cryptography in the quantum model.

Our results. We choose to focus here on differential cryptanalysis, the truncated differential variant, and on linear cryptanalysis. We give for the first time a synthetic

[1] Previous results as [Kap14, KM10, KM12] only consider quantizing generic attacks.

description of these attacks, and study how they are affected by the availability of quantum computers. As expected, we often get a quadratic speed-up, but not for all attacks.

In this work we use the concept of quantum walks to devise quantum attacks. This framework contains a lot of well known quantum algorithms such as Grover's search or Ambainis' algorithm for element distinctness. More importantly, it allows one to compose these algorithms in the same way as classical algorithms can be composed. In order to keep our quantum attacks as simple as possible, we use a slightly modified Grover's search algorithm that can use quantum checking procedures. This simple trick comes at the cost of constant factors (ignored in our analysis), but a more involved approach, making better use of quantum walks may remove those additional factors.

We prove the following non-obvious results:

- Differential cryptanalysis and linear cryptanalysis usually offer a *quadratic gain* in the Q2 model over the classical model.
- Truncated differential cryptanalysis, however, usually offers *smaller gains* in the Q2 model.
- Therefore, the optimal quantum attack is not always a quantum version of the optimal classical attack.
- In the Q1 model, cryptanalytic attacks might offer *little gain* over the classical model when the key-length is the same as the block length (*e.g.* AES-128).
- But the gain of cryptanalytic attacks in the Q1 model can be quite significant (*similar to the Q2 model*) when the key length is longer (*e.g.* AES-256).

The rest of the paper is organized as follows. We first present some preliminaries on the classical (Section 2) and quantum (Section 3) settings. Section 4 treats differential attacks, while Section 5 deals with truncated differential attacks and Section 6 provides some applications on ciphers LAC and KLEIN. We study linear cryptanalysis in Section 7. In Section 8, we discuss the obtained results. Section 9 concludes the paper and presents some open questions.

2 Preliminaries

In the following, we consider a block cipher E, with a blocksize of n bits, and a keysize of k bits. We assume that E is an iterated design with r rounds, and we use $E^{(t)}$ to denote a reduced version with t rounds (so that $E = E^{(r)}$). When the cipher E is computed with a specific key $\kappa \in \{0,1\}^k$, its action on a block x is denoted by $E_\kappa(x)$. The final goal of an attacker is to find the secret key κ^* that was used to encrypt some data. A query to the cryptographic oracle is denoted $E(x)$, where it is implicitly assumed that E encrypts with the key κ^*, *i.e.*, $E(x) = E_{\kappa^*}(x)$.

Key-recovery attack. The key can always be found using a brute-force attack; following our notations, the complexity of such a generic attack is 2^k. This defines the ideal security, *i.e.* the security a cipher should provide. Therefore, a cipher is considered *broken* if the key can be found "faster" than with the brute-force attack, where "faster" typically means with "less encryptions". Three parameters define the efficiency of a specific attack. The *data complexity* is the number of calls to the cryptographic oracle $E(x)$. The *time complexity* is the time required to recover the key κ^*. We consider that querying the cryptographic oracle requires one unit of time, so that the data complexity is included in the time complexity. The *memory complexity* is the memory needed to perform the attack.

Distinguishers. Another type of attacks, less powerful than key-recovery ones, are distinguishers. Their aim is to distinguish a concrete cipher from an ideal one. A distinguishing attack often gives rise to a key-recovery attack and is always the sign of a weakness of the block cipher.

Our scenario. In this paper, we consider some of the main families of non-generic attacks that can be a threat to some ciphers: differential and linear attacks. We propose their quantized version for the distinguisher and the last-rounds key-recovery variants of linear, simple differentials and truncated differentials. Our aim is to provide a solid first step towards "quantizing" symmetric families of attacks. To reach this objective, due to the technicality of the attacks themselves, and even more due to the technicality of combining them with quantum tools, we consider the most basic versions of the attacks.

Success probability. For the sake of simplicity, in this paper we do not take into account the success probability in the parameters of the attacks. In particular, because it affects in the same way both classical and quantum versions, it is not very useful for the comparison we want to perform. In practice, it would be enough to increase the data complexity by a constant factor to reach any pre-specified success probability. A detailed study of the success probability of statistical attacks can be found in [BGT11].

3 Quantum algorithms

We use a number of quantum techniques in order to devise quantum attacks. Most of them are based on well-known quantum algorithms that have been studied extensively over the last decades. The equivalent to the classical brute-force attack in the quantum world is to search through the key space using a Grover's search algorithm [Gro96], leading to complexity $2^{k/2}$. Our goal is to devise quantum attacks that might be a threat to symmetric primitives by displaying a smaller complexity than the generic quantum exhaustive search.

3.1 Variations on Grover's algorithm

Although Grover's algorithm is usually presented as a search in an unstructured database, we use in our applications the following slight generalization (see [San08] for a nice exposition on quantum-walk-based search algorithms). The task is to find a marked element from a set X. We denote by $M \subseteq X$ the subset of marked elements and assume that we know a lower bound ε on the fraction $|M|/|X|$ of marked elements. A classical algorithm to solve this problem is to repeat $O(1/\varepsilon)$ times: (i) sample an element from X, (ii) check if it is marked.

The cost of this algorithm can therefore be expressed as a function of two parameters: the *Setup cost* S, which is the cost of sampling a uniform element from X, and the *Checking cost* C, which is the cost of checking if an element is marked. The cost considered by the algorithm can be the time or the number of queries to the input. It suffices to consider specifically one of those resources when quantifying the Setup and Checking cost.

Similarly, Grover's algorithm [Gro96] is a quantum search procedure that finds a marked element, and whose complexity can be written as a function of the *quantum Setup cost* S, which is the cost of constructing a uniform superposition of all elements in X, and the *quantum Checking cost* C, which is the cost of applying a controlled-phase gate to the marked elements. Notice that a classical or a quantum algorithm that checks membership to M can easily be modified to get a controlled-phase.

Theorem 1 (Grover)**.** *There exists a quantum algorithm which, with high probability, finds a marked element, if there is any, at cost of order* $\frac{\mathrm{S}+\mathrm{C}}{\sqrt{\varepsilon}}$.

In particular, the setup and the checking steps can themselves be quantum procedures. Assume for instance that the set X is itself a subset of a larger set $\tilde{X}$. Grover's algorithm can then find an element $x \in X$ at a cost $(\tilde{X}/X)^{1/2}$, assuming that the setup and checking procedures are easy. Moreover, a closer look at the algorithm shows that if one ignores the final measurement that returns one element, the algorithm produces a uniform superposition of the elements in X, which can be used to setup another Grover search.

Grover's algorithm can also be written as a special case of amplitude amplification, a quantum technique introduced by Brassard, Høyer and Tapp in order to boost the success probability of quantum algorithms [BHMT02]. Intuitively, assume that a quantum algorithm $\mathcal{A}$ produces a superposition of outputs in a good subspace G and outputs in a bad subspace B. Then there exists a quantum algorithm that calls $\mathcal{A}$ as a subroutine to amplify the amplitude of good outputs.

If $\mathcal{A}$ was a classical algorithm, repeating it $\Theta(1/a)$, where a is the probability of producing a good output, would lead to a new algorithm with constant success probability. Just as Grover's algorithm, the amplitude amplification technique achieves the same result with a quadratic improvement [BHMT02]. The intuitive reason is that quantum operations allow to amplify the amplitudes of good output states, and that the corresponding probabilities are given by the squares of the amplitudes. Therefore, the amplification is quadratically faster than in the classical case.

Theorem 2 (Amplitude amplification)**.** *Let $\mathcal{A}$ be a quantum algorithm that, with no measurement, produces a superposition $\sum_{x\in G}\alpha_x|x\rangle + \sum_{y\in B}\alpha_y|y\rangle$. Let $a = \sum_{x\in G}|\alpha_x|^2$ be the probability of obtaining, after measurement, a state in the* good *subspace G.*

Then, there exists a quantum algorithm that calls $\mathcal{A}$ and $\mathcal{A}^{-1}$ as subroutines $\Theta(1/\sqrt{a})$ times and produces an outcome $x \in G$ with a probability at least $\max(a, 1-a)$.

A variant of quantum amplification amplitude can be used to count approximately, again with a quadratic speed-up over classical algorithms [BHT98].

Theorem 3 (Quantum counting)**.** *Let $F : \{0, \dots N-1\} \to \{0,1\}$ be a Boolean function, and $p = |F^{-1}(1)|/N$. For every positive integer D, there is a quantum algorithm that makes D queries to F and, with probability at least $8/\pi^2$, outputs an estimate p' to p such that $|p - p'| \leq 2\pi\sqrt{p}/D + \pi^2/D^2$.*

3.2 Quantum search of pairs

We also use Ambainis' quantum algorithm for the element distinctness problem. In our work, we use it to search for collisions.

Theorem 4 (Ambainis [Amb07])**.** *Given a list of numbers $x_1, \dots, x_n$, there exists a quantum algorithm that finds, with high probability, a pair of indices (i, j) such that $x_i = x_j$, if there exists one, at a cost $O(n^{2/3})$.*

The quantum algorithm proposed by Ambainis can easily be adapted to finding a pair satisfying $x_i + x_j = w$ for any given w (when the x_i's are group elements and the "+" operation can be computed efficiently).

Ambainis' algorithm can also be adapted to search in a list $\{x_1, \dots, x_n\}$ for a pair of indices (i, j) such that (x_i, x_j) satisfies some relation R, with the promise that the input contains at least k possible pairs satisfying R. If the input of the problem is a uniformly random set of pairs, it is sufficient, in order to find one, to run Ambainis' algorithm on a smaller random subset of inputs.

Theorem 5. *Consider a list of numbers $x_1, \dots, x_n$ with $x_i \in X$ and a set of pairs $\mathcal{D} \subset X \times X$ such that $\mathcal{D}$ contains exactly k pairs. There exists a quantum algorithm that finds, with high probability, a pair (i, j) such that $(x_i, x_j) \in \mathcal{D}$, at a cost $O(n^{2/3}k^{-1/3})$ on average over uniformly distributed inputs.*

Proof. For a uniformly chosen subset $X' \subset X$ such that $|X'| = n/\sqrt{k}$, there is, with constant probability, at least one pair from $\mathcal{D}$ in $X' \times X'$. According to Theorem 4, the cost of finding this pair is $O(n^{2/3}k^{-1/3})$. Therefore, the quantum algorithm starts by sampling a random X' and then runs Ambainis' algorithm on this subset.

Table 1: Notations used in the attacks.

n	block-size
k	key-size
Δ_{in}	size (log) of the set of input differences
Δ_{out}	size (log) of the set of output differences
Δ_{fin}	size (log) of the set of differences $\mathcal{D}_{\text{fin}}$ after last rounds
h_S	probability ($-\log$) of the differential characteristic ($h_S < n$)
h_T	probability ($-\log$) of the truncated differential characteristic
h_{out}	probability ($-\log$) of generating δ_{out} from $\mathcal{D}_{\text{fin}}$
k_{out}	number of key bits required to invert the last rounds
$C_{k_{\text{out}}}$	cost of recovering the last round subkey from a good pair
$C^*_{k_{\text{out}}}$	quantum cost of recovering the last round subkey from a good pair
ε	bias of the linear approximation
ℓ	number of linear approximations (Matsui's algorithm 1)

Notice that if the algorithm runs on uniformly random inputs, the set X' does not need to be itself chosen at random. Any sufficiently large subset will contain one of the pairs with high probability, with high probability over the distribution of inputs.

Before ending this section on quantum algorithms, we make a remark on the outputs produced by quantum-walk-based algorithms, such as Ambainis' or Grover's algorithm. In our applications, we use these not necessarily to produce some output, but to prepare a superposition of the outputs. Similarly to Grover's algorithm, this can be done by running the algorithm without performing the final measurement. However, since Ambainis' algorithm uses a quantum memory to maintain some data structure, the superposition could in principle include the data from the memory. This issue does not happen with Grover's algorithm precisely because it does not require any data structure.

In our case, the algorithm ends in a superposition of nodes containing at least one of the searched pairs. It has no consequence for our application, because we are nesting this procedure in Grover's algorithm. Alternatively, it is possible to use amplitude amplification afterwards in order to amplify the amplitude on the good nodes. However, this could be an issue when nesting our algorithm in an arbitrary quantum algorithm. For a discussion on nested quantum walks, see [JKM13].

4 Differential Cryptanalysis

Differential cryptanalysis was introduced in [BS90] by Biham and Shamir. It studies the propagation of differences in the input of a function (δ_{in}) and their influence on the generated output difference (δ_{out}). In this section, we present the two main types of differential attacks on block ciphers in the classical world: the *differential distinguisher* and the *last-rounds attack*, and then analyze their complexities for quantum adversaries.

4.1 Classical Adversary

Differential attacks exploit the fact that there exists an input difference δ_{in} and an output difference δ_{out} to a cipher E such that

$$h_S := -\log \Pr_x[E(x \oplus \delta_{\text{in}}) = E(x) \oplus \delta_{\text{out}}] < n, \tag{1}$$

i.e., such that we can detect some non-random behaviour of the differences of plaintexts x and $x \oplus \delta_{\text{in}}$. Here, "$\oplus$" represents the bitwise xor of bit strings of equal length. The

value of h_S is generally computed for a random key, and as usual in the literature, we will assume that Eq. (1) approximately holds for the secret key κ^*. Such a relation between δ_{in} and δ_{out} is typically found by studying the internal structure of the primitive in detail. While it seems plausible that a quantum computer could also be useful to find good pairs $(\delta_{\text{in}}, \delta_{\text{out}})$, we will not investigate this problem here, but rather focus on attacks that can be mounted once a good pair satisfying Eq. (1) is given.

4.1.1 Differential Distinguisher

This non-random behaviour can already be used to attack a cryptosystem by distinguishing it from a random function. This distinguisher is based on the fact that, for a random function and a fixed δ_{in}, obtaining the δ_{out} difference in the output would require 2^n trials, where n is the size of the block. On the other hand, for the cipher E, if we collect 2^{h_S} input pairs verifying the input difference δ_{in}, we can expect to obtain one pair of outputs with output difference δ_{out}. The complexity of such a distinguisher exploiting Eq. (1) is 2^{h_S+1} in both data and time, and is negligible in terms of memory:

$$T_{\text{C}}^{\text{s. dist.}} = D_{\text{C}}^{\text{s. dist.}} = 2^{h_S+1}. \tag{2}$$

Here, the subscript C refers to classical and *s. dist.* to "simple distinguisher" by opposition to its truncated version later in the text.

Assuming that such a distinguisher exists for the first R rounds of a cipher, we can transform the attack into a key recovery on more rounds by adding some rounds at the end or beginning of the cipher. This is called a *last-rounds attack*, and allows to attack more rounds than the distinguisher, typically one or two, depending on the cipher.

4.1.2 Last-Rounds Attack

For simplicity and without loss of generality, we consider that the rounds added to the distinguisher are placed at the end. We attack a total of $r = R + r_{\text{out}}$ rounds, where R are the rounds covered by the distinguisher. The main goal of the attack is to reduce the key space that needs to be searched exhaustively from 2^k to some $2^{k'}$ with $k' < k$. For this, we use the fact that we have an advantage for finding an input x such that $E^{(R)}(x) \oplus E^{(R)}(x \oplus \delta_{\text{in}}) = \delta_{\text{out}}$.

For a pair that generates the difference δ_{out} after R rounds, we denote by $\mathcal{D}_{\text{fin}}$ the set of possible differences generated in the output after the final r_{out} rounds, the size of this set by $2^{\Delta_{\text{fin}}} = |\mathcal{D}_{\text{fin}}|$. Let $2^{-h_{\text{out}}}$ denote the probability of generating the difference δ_{out} from a difference in $\mathcal{D}_{\text{fin}}$ when computing r_{out} rounds in the backward direction, and by k_{out} the number of key bits involved in these rounds. The goal of the attack is to construct a list L of candidates for the partial key that contains almost surely the correct value, and that has size strictly less than $2^{k_{\text{out}}}$. For this, one starts with lists L_M and L_K where L_M is a random subset of 2^{h_S} possible messages and L_K contains all possible k_{out}-bit strings. From Eq. (1), the list L_M contains an element x such that $E^{(R)}(x) \oplus E^{(R)}(x \oplus \delta_{\text{in}}) = \delta_{\text{out}}$ with high probability. Let us apply two successive tests to the lists.

The first test keeps only the $x \in L_M$ such that $E(x) \oplus E(x \oplus \delta_{\text{in}}) \in \mathcal{D}_{\text{fin}}$. The probability of satisfying this equation is $2^{\Delta_{\text{fin}}-n}$. This gives a new list L'_M of size $|L'_M| = 2^{h_S+\Delta_{\text{fin}}-n}$. The cost of this first test is 2^{h_S+1}.

The second test considers the set $L'_M \times L_K$ and keeps only the couples (x, κ) such that $E_\kappa^{(R)}(x) + E_\kappa^{(R)}(x + \delta_{in}) = \delta_{\text{out}}$. This is done by computing backward the possible partial keys for a given difference in $\mathcal{D}_{\text{out}}$. Denote $C_{k_{\text{out}}}$ the average cost of generating those keys for a given input pair. Notice that $C_{k_{\text{out}}}$ can be 1 when the number of rounds added is reasonably small[2], and is upper bounded by $2^{k_{\text{out}}}$, that is, $1 \leq C_{k_{\text{out}}} \leq 2^{k_{\text{out}}}$. For a random

[2]For example, using precomputation tables with the values that allow the differential transitions through the S-Boxes.

pair (x, κ), the probability of passing this test is $2^{-h_{\text{out}}}$. The size of the resulting set is therefore expected to be $2^{-h_{\text{out}}} \times |L'_M| \times |L_K| = 2^{h_S+\Delta_{\text{fin}}-n+k_{\text{out}}-h_{\text{out}}}$. The cost of this step is $C_{k_{\text{out}}} 2^{h_S+\Delta_{\text{fin}}-n}$.

The previous step produces a list of candidates for the partial key corresponding to the key bits involved in the last r_{out} rounds and leading to a difference δ_{out} after R rounds. The last step of the attack consists in performing an exhaustive search within all partial keys of this set completed with all possible $k - k_{\text{out}}$ bits. The cost of this step is $2^{h_S+\Delta_{\text{fin}}-n+k-h_{\text{out}}}$.

In practice, the lists do not need to be built and everything can be performed "on the fly". Consequently, memory needs can be made negligible. The total time complexity is:

$$T_{\text{C}}^{\text{s. att.}} = 2^{h_S+1} + 2^{h_S+\Delta_{\text{fin}}-n} \left(C_{k_{\text{out}}} + 2^{k-h_{\text{out}}} \right), \tag{3}$$

while the data complexity of this classical attack is $D_{\text{C}}^{\text{s. att.}} = 2^{h_S+1}$. The attack is more efficient than an exhaustive search if $T_{\text{C}}^{\text{s. att.}} < 2^k$.

4.2 Quantum Adversary

We first give attacks in the Q2 model, using superposition queries.

4.2.1 Differential Distinguisher in the Q2 model

The distinguisher consists in applying a Grover search over the set of messages X, of size 2^n. More precisely, the algorithm makes $2^{h_S/2+1}$ queries to the encryption cipher, trying to find a marked element $x \in M = \{x \in X \,:\, E(x \oplus \delta_{\text{in}}) = E(x) \oplus \delta_{\text{out}}\}$. If it finds any, it outputs "concrete". If it does not, it outputs "random".

With the notations of the previous sections, the fraction of marked elements is $\varepsilon = 2^{-h_S}$ and Grover's algorithm finds a marked element after $\frac{1}{\sqrt{\varepsilon}} = 2^{h_S/2}$ iterations, each one requiring two queries to the encryption cipher. The time and data complexities are:

$$T_{\text{Q2}}^{\text{s. dist.}} = D_{\text{Q2}}^{\text{s. dist.}} = 2^{h_S/2+1}. \tag{4}$$

It remains to prove that in the case of a random function, the probability of finding a marked element is negligible. Assume that the probability of finding a marked element after $2^{h_S/2}$ quantum queries is δ. Then, this can be wrapped into an amplitude amplification procedure (Theorem 2), leading to a bounded error algorithm making $(1/\sqrt{\delta})2^{h_S/2}$ queries. Since Grover's algorithm is optimal, we get that $(1/\sqrt{\delta})2^{h_S/2} \geq 2^{n/2}$, leading to $\delta \leq 2^{h_S-n}$.

4.2.2 Last-Rounds Attack in the Q2 model

An important point of the attack in the Q2 model is that it should avoid creating lists. Instead, the algorithm queries the cryptographic algorithm whenever it needs to sample an element from the list.

The quantum attack can be described as a Grover search, with quantum procedures for the setup and checking phases. The algorithm searches in the set $X = \{x : E(x \oplus \delta_{\text{in}}) \oplus E(x) \in \mathcal{D}_{\text{fin}}\}$ for a message such that $E^{(R)}(x \oplus \delta_{\text{in}}) \oplus E^{(R)}(x) = \delta_{\text{out}}$. This procedure outputs a message and when it is found, it suffices to execute the sequence corresponding to the checking of the Grover search once more: generate partial key candidates and search among them, completed with all possible remaining $k - k_{\text{out}}$ bits. This outputs the correct key and only adds a constant overhead factor to Grover search. Notice that using tailor-made quantum walks, it should be possible to suppress this overhead. Here we use Grover search to keep the attacks as simple as possible.

The setup phase prepares a uniform superposition of the $x \in X$; this costs $\text{S} = 2^{(n-\Delta_{\text{fin}})/2}$ using Grover's algorithm. The checking phase takes a value x and must determine whether $(x, x \oplus \delta_{\text{in}})$ is a good pair; it consists of the following successive steps:

1. Compute all possible partial keys κ_{out} for the k_{out} bits that intervene in the last r_{out} rounds, assuming that $(x, x \oplus \delta_{\text{in}})$ is a good pair $(E^{(R)}(x \oplus \delta_{\text{in}}) \oplus E^{(R)}(x) = \delta_{\text{out}})$;
2. Complete the key by searching exhaustively using a Grover search, checking if the obtained key is the correct one.

The cost of computing all possible partial keys is $C^*_{k_{\text{out}}}$. The number of partial keys is $2^{k_{\text{out}}-h_{\text{out}}}$, then completed by $k - k_{\text{out}}$ remaining bits. The cost of checking through all of them is thus $\mathrm{C} = C^*_{k_{\text{out}}} + 2^{(k-h_{\text{out}})/2}$.

The procedure succeeds whenever a message x is found such that $E^{(R)}(x) \oplus E^{(R)}(x \oplus \delta_{\text{in}}) = \delta_{\text{out}}$. Therefore, the probability of finding a marked element is lower bounded by $\varepsilon \geq 2^{-h_S-\Delta_{\text{fin}}+n}$. This is the conditional probability of getting $E^{(R)}(x \oplus \delta_{\text{in}}) \oplus E^{(R)}(x) = \delta_{\text{out}}$ given that the output difference is in $\mathcal{D}_{\text{fin}}$.

The total cost of the attack in the Q2 model is:

$$T^{\text{s. att.}}_{\text{Q2}} = 2^{h_S/2+1} + 2^{(h_S+\Delta_{\text{fin}}-n)/2}\left(C^*_{k_{out}} + 2^{(k-h_{\text{out}})/2}\right), \quad D^{\text{s. att.}}_{\text{Q2}} = 2^{h_S/2+1}, \tag{5}$$

with a data complexity identical to that of the distinguisher.

4.2.3 Last-Rounds Attack in the Q1 model

We can also have a speed-up for the last-round attack in the Q1 model. In this model, the quantum operations only take place after a classical acquisition of the data. In particular, the data complexity will be the same as for a classical adversary. After the first filtering step of the classical last-round attack, $2^{h_S-n+\Delta_{\text{fin}}}$ couples satisfying $E(x) \oplus E(x \oplus \delta_{\text{in}}) \in \mathcal{D}_{\text{fin}}$ are obtained. The attacker then uses a quantum algorithm to generate the partial keys κ_{out}, and a Grover search among those, completed with all possible remaining $k - k_{\text{out}}$ bits of the key, in order to find the key. This leads to data and time complexities of:

$$D^{\text{s. att.}}_{\text{Q1}} = 2^{h_S+1}, \quad T^{\text{s. att.}}_{\text{Q1}} = 2^{h_S+1} + 2^{(h_S+\Delta_{\text{fin}}-n)/2}\left(C^*_{k_{out}} + 2^{(k-h_{\text{out}})/2}\right), \tag{6}$$

where $C^*_{k_{\text{out}}}$ denotes the average time complexity of generating the partial keys of length k_{out}, on a *quantum* computer.

Let us point out that any classical attack with data complexity smaller than square root of the exhaustive search of the key can be translated into an effective attack also in the Q1 model. This is more likely to happen for larger keys, where the limiting terms are often the second and third terms of $T^{\text{s. att.}}_{\text{Q1}}$ in Eq. (6). See a detailed example in 6.2. The fact that long keys are more likely to "maintain" the validity of the attacks is an interesting result, as longer keys correspond to the recommendations for post-quantum symmetric primitives. In these cases, the Q1 model is particularly meaningful.

4.2.4 Generating partial keys on a quantum computer

We investigate further the average cost $C_{k_{\text{out}}}$ of generating the partial keys compatible with some input x such that $E(x) \oplus E(x \oplus \delta_{\text{in}}) \in \mathcal{D}_{\text{fin}}$. These partial keys correspond to the key bits involved in a transition from $\mathcal{D}_{\text{fin}}$ to δ_{out}.

Using a classical computer and a precomputation table, this usually takes constant time, but can be up to $2^{k_{\text{out}}}$ in the worst case. It turns out that the worst case this can be sped up using a quantum computer.

Let $K = 2^{k_{\text{out}}-h_{\text{out}}}$ be the average number of partial key candidates compatible with some input x, and denote $N = 2^{k_{\text{out}}}$. Finding one partial key can be done using Grover search with $(N/K)^{1/2}$ steps. The cost of finding a second one is $(N/(K-1))^{1/2}$, and so on. This leads to the following upper bound on $C^*_{k_{\text{out}}}$, the quantum version of $C_{k_{\text{out}}}$:

$$C^*_{k_{\text{out}}} \leq \sqrt{N}\sum_{i=1}^{K} i^{-1/2} \leq \quad 2\sqrt{NK}.$$

Replacing with our parameters, this gives $C^*_{k_{\text{out}}} \leq 2^{k_{\text{out}}-h_{\text{out}}/2+1}$.

5 Truncated Differential Cryptanalysis

Truncated differential cryptanalysis was introduced by Knudsen [Knu94] in 94. Instead of fixed input and output differences, it considers sets of differences (like the differences in the output in the last-rounds attack that we have considered in the previous section).

We assume in the following that we are given two sets $\mathcal{D}_{\text{in}}$ and $\mathcal{D}_{\text{out}}$ of input and output differences such that the probability of generating a difference in $\mathcal{D}_{\text{out}}$ from one in $\mathcal{D}_{\text{in}}$ is 2^{-h_T}. We further consider that $\mathcal{D}_{\text{in}}$ and $\mathcal{D}_{\text{out}}$ are vector spaces.

5.1 Classical Adversary

As in the simple differential case, we first present the differential distinguisher based on the non-random property of the differences behaviour, and then discuss the last-rounds attack obtained from the truncated differential distinguishers.

5.1.1 Truncated Differential Distinguisher

Let $2^{\Delta_{\text{in}}}$ and $2^{\Delta_{\text{out}}}$ denote the sizes of the input and output sets of differences, respectively. For simplicity and without loss of generality, we assume to have access to an encryption oracle, and therefore only consider the truncated differential as directed from input to output[3]. We denote by 2^{-h_T} the probability of generating a difference in $\mathcal{D}_{\text{out}}$ from one in $\mathcal{D}_{\text{in}}$. The condition for the distinguisher to work is that $2^{-h_T} > 2^{\Delta_{\text{out}}-n}$. In this analysis, we assume that $2^{-h_T} \gg 2^{\Delta_{\text{out}}-n}$.

The advantage of truncated differentials is that they allow the use of structures, *i.e.*, sets of plaintext values that can be combined into input pairs with a difference in $\mathcal{D}_{\text{in}}$ in many different ways: one can generate $2^{2\Delta_{\text{in}}-1}$ pairs using a single structure of size $2^{\Delta_{\text{in}}}$. This reduces the data complexity compared to simple differential attacks.

Two cases need to be considered. If $\Delta_{\text{in}} \geq (h_T+1)/2$, we build a single structure $\mathcal{S}$ of size $2^{(h_T+1)/2}$ such that for all pairs $(x, y) \in \mathcal{S} \times \mathcal{S}$, $x \oplus y \in \mathcal{D}_{\text{in}}$. This structure generates 2^{h_T} pairs. If $\Delta_{\text{in}} \leq (h_T+1)/2$, we have to consider multiple structures $\mathcal{S}_i$. Each structure contains $2^{\Delta_{\text{in}}}$ elements, and generates $2^{2\Delta_{\text{in}}-1}$ pairs of elements. We consider $2^{h_T-2\Delta_{\text{in}}+1}$ such structures in order to have 2^{h_T} candidate pairs.

In both cases, we have 2^{h_T} candidate pairs. With high probability, one of these pairs shall satisfy $E(x) \oplus E(y) \in \mathcal{D}_{\text{out}}$, something that should not occur for a random function if $2^{-h_T} \gg 2^{\Delta_{\text{out}}-n}$. Therefore detecting a single valid pair gives an efficient distinguisher.

The attack then works by checking if, for a pair generated by the data, the output difference belongs to $\mathcal{D}_{\text{out}}$. Since $\mathcal{D}_{\text{out}}$ is assumed to be a vector space, this can be reduced to trying to find a collision on $n - \Delta_{\text{out}}$ bits of the output. Once the data is generated, looking for a collision is not expensive (*e.g.* using a hash table), which means that time and data complexities coincide:

$$D_{\text{C}}^{\text{tr. dist.}} = \max\{2^{(h_T+1)/2}, 2^{h_T-\Delta_{\text{in}}+1}\}, \quad T_{\text{C}}^{\text{tr. dist.}} = \max\{2^{(h_T+1)/2}, 2^{h_T-\Delta_{\text{in}}+1}\}. \quad (7)$$

5.1.2 Last-Rounds Attack

Last-rounds attacks work similarly as in the case of simple differential cryptanalysis. For simplicity, we assume that r_{out} rounds are added at the end of the truncated differential. The intermediate set of differences is denoted $\mathcal{D}_{\text{out}}$, and its size is $2^{\Delta_{\text{out}}}$. The set $\mathcal{D}_{\text{fin}}$,

[3] In the case where the other direction provides better complexities, we could instead perform queries to a decryption oracle and change the roles of input and output in the attack. We assume that the most interesting direction has been chosen.

of size $2^{\Delta_{\text{fin}}}$ denotes the possible differences for the outputs after the final round. The probability of reaching a difference in $\mathcal{D}_{\text{out}}$ from a difference in $\mathcal{D}_{\text{in}}$ is 2^{-h_T}, and the probability of reaching a difference in $\mathcal{D}_{\text{out}}$ from a difference in $\mathcal{D}_{\text{fin}}$ is $2^{-h_{\text{out}}}$. Applying the same algorithm as in the simple differential case, the data complexity remains the same as for the distinguisher:

$$D_{\text{C}}^{\text{tr. att.}} = \max\{2^{(h_T+1)/2}, 2^{h_T-\Delta_{\text{in}}+1}\}. \tag{8}$$

The time complexity in this case is:

$$T_{\text{C}}^{\text{tr. att.}} = \max\{2^{(h_T+1)/2}, 2^{h_T-\Delta_{\text{in}}+1}\} + 2^{h_T+\Delta_{\text{fin}}-n}\left(C_{k_{\text{out}}} + 2^{k-h_{\text{out}}}\right), \tag{9}$$

where $C_{k_{\text{out}}}$ is the average cost of finding all the partial key candidates corresponding to a pair of data with a difference in $\mathcal{D}_{\text{out}}$. As mentioned earlier, $C_{k_{\text{out}}}$ ranges from 1 to $2^{k_{\text{out}}}$.

5.2 Quantum Adversary

The truncated differential cryptanalysis is similar to the simple differential cryptanalysis, except that $\mathcal{D}_{\text{in}}$ and $\mathcal{D}_{\text{out}}$ are now sets instead of two fixed bit strings.

5.2.1 Truncated Differential Distinguisher

Similarly to simple differential cryptanalysis, the distinguisher can only be more efficient in the Q2 model. This comes from the fact that in both cases, the data complexity is the bottleneck. Since the Q1 model does not provide any advantage over the classical one in data collection, there is no advantage in this model.

We use Ambainis' algorithm for element distinctness, given in Theorem 4, in order to search for collisions inside the structures. If a single structure is involved, the algorithm searches for a pair of messages (x, y) in a set of size $2^{(h_T+1)/2}$, such that $E(x) \oplus E(y) \in \mathcal{D}_{\text{out}}$. Since there is, on average, only one such pair, this can be done using a quantum algorithm with $2^{(h_T+1)/3}$ queries.

If multiple structures are required, the strategy is to search for one structure that contains a pair (x, y) such that $E(x) \oplus E(y) \in \mathcal{D}_{\text{out}}$. This is done with a Grover search on the structure, using Ambainis' algorithm for the checking phase. This returns a structure containing a desired pair, which is sufficient for the distinguisher. The setup cost is constant. The checking step, consisting in searching for a specific pair inside a structure of size $2^{\Delta_{\text{in}}}$, can be done with $\text{C} = 2^{2\Delta_{\text{in}}/3}$ queries. Finally, since there is, with high probability, at least one structure in $2^{h_T-2\Delta_{\text{in}}+1}$ containing a pair such that $E(x) \oplus E(y) \in \mathcal{D}_{\text{out}}$, we get a lower bound on the success probability $\varepsilon \geq 2^{2\Delta_{\text{in}}-h_T-1}$. Using Theorem 1, the total queries complexity is at most $2^{(h_T+1)/2-\Delta_{\text{in}}/3}$.

Combining both results leads to overall data and time complexities given by:

$$D_{\text{Q2}}^{\text{tr. dist.}} = T_{\text{Q2}}^{\text{tr. dist.}} = \max\left\{2^{(h_T+1)/3}, 2^{(h_T+1)/2-\Delta_{\text{in}}/3}\right\}. \tag{10}$$

Similarly to the the quantum simple differential distinguisher, applying the same algorithm to a random function, and stopping it after the same number of queries only provides a correct answer with negligible probability.

5.2.2 Last-Rounds Attack in the Q1 model

As seen in Section 5.1.2, last-round attacks for truncated differential cryptanalysis are very similar to attacks with a simple differential. The attack in the Q1 model will differ from the attack of Section 4.2.3 only in the first step, when querying the encryption function

with the help of structures. We start by generating a list of 2^{h_T} pairs with differences in $\mathcal{D}_{\text{in}}$, which is done with data complexity:

$$D_{\text{Q1}}^{\text{tr. att.}} = \max\{2^{(h_T+1)/2}, 2^{h_T-\Delta_{\text{in}}+1}\}. \tag{11}$$

The second step is to filter the list of elements to keep only the pairs (x, y) such that $E(x) \oplus E(y) \in \mathcal{D}_{\text{fin}}$. Notice that such a filtering can be done at no cost. It suffices to sort the elements according to the values of their image, while constructing the list.

Finally, similarly to the Q1 simple differential attack, a quantum search algorithm is run on the filtered pairs, and the checking procedure consists in generating the partial key candidates completed with $k - k_{\text{out}}$ bits, and searching exhaustively for the key used in the cryptographic oracle. In the Q1 model, the quantum speed-up only occurs in this step.

The average cost of generating the partial keys on a quantum computer is denoted by $C^*_{k_{\text{out}}}$. The average number of partial keys for a given pair of input is $2^{k_{\text{out}}-h_{\text{out}}}$. The fraction ε of marked elements is $\varepsilon = 2^{-h_T-\Delta_{\text{fin}}+n}$, the setup cost is $S = 1$ and the checking cost, a Grover search over the key space, is $\mathrm{C} = C^*_{k_{out}} + 2^{(k-h_{\text{out}})/2}$. This gives a total cost:

$$T_{\text{Q1}}^{\text{tr. att.}} = \max\left\{2^{(h_T+1)/2}, 2^{h_T-\Delta_{\text{in}}+1}\right\} + 2^{(h_T+\Delta_{\text{fin}}-n)/2}\left(C^*_{k_{out}} + 2^{(k-h_{\text{out}})/2}\right). \tag{12}$$

5.2.3 Last-Rounds Attack in the Q2 model

In the Q2 model, we want to avoid building classical lists. Instead, we query the cryptographic oracle each time we need to sample a specific element. This is challenging in the case of truncated differential because the use of structures made of lists is crucial. The idea is to query the elements of the list on the fly.

Assume first that $h_T \leq 2\Delta_{\text{in}} - 1$. Then, it is possible to get 2^{h_T} pairs with differences in $\mathcal{D}_{\text{in}}$ with a single structure, $\mathcal{S}$, of size $2^{(h_T+1)/2}$. The attack runs a Grover search over $X = \{(x, y) \in \mathcal{S} \times \mathcal{S} : E(x) \oplus E(y) \in \mathcal{D}_{\text{fin}}\}$. The checking procedure is the same as for the quantum simple differential attack. For a given a pair of inputs, it generates all possible partial keys, and completes them to try to get the key used by cryptographic oracle. This procedure returns a pair (x, y). The final step is to execute the checking procedure in Grover search once more, suitably modified to return the key given the pair (x, y).

We analyze the setup cost of the attack. To prepare a superposition of the pairs in X, we use a new quantum search algorithm given in Theorem 5. This algorithm searches in a list for a pair of elements with a certain property, considering there exist k such pairs. In our case, the list of elements is $\mathcal{S}$ of size $2^{(h_T+1)/2}$. The total number of elements such that $E(x) \oplus E(y) \in \mathcal{D}_{\text{fin}}$ is therefore $2^{h_T-n+\Delta_{\text{fin}}}$. The algorithm of Theorem 5 prepares a superposition of elements in X in time $\mathrm{S} = 2^{(h_T+1)/3-(h_T-n+\Delta_{\text{fin}})/3} = 2^{(n-\Delta_{\text{fin}}+1)/3}$. The cost of the checking procedure is $\mathrm{C} = C^*_{k_{\text{out}}} + 2^{(k-h_{\text{out}})/2}$, as before. The procedure is successful whenever a pair (x, y) such that $E^{(R)}(x) \oplus E^{(R)}(y) \in \mathcal{D}_{\text{out}}$ is found. Given that the search is among pairs satisfying $x \oplus y \in \mathcal{D}_{\text{in}}$ and $E(x) \oplus E(y) \in \mathcal{D}_{\text{fin}}$, the probability for a pair to be good is $\varepsilon = 2^{-h_T-\Delta_{\text{fin}}+n}$. This gives a total running time:

$$2^{h_T/2-(n-\Delta_{\text{fin}})/6} + 2^{(h_T+\Delta_{\text{fin}}-n)/2}\left(C^*_{k_{\text{out}}} + 2^{(k-h_{\text{out}})/2}\right).$$

Suppose now that multiple structures $\mathcal{S}_i$ of size $2^{\Delta_{\text{in}}}$ are required, where i goes from 1 to $2^{h_T-2\Delta_{\text{in}}+1}$. The search is now over the set $X = \bigcup_i \{(x, y) \in \mathcal{S}_i \times \mathcal{S}_i : E(x) \oplus E(y) \in \mathcal{D}_{\text{fin}}\}$. To get a superposition of the pairs in X, we compose a Grover search over the structure with the algorithm from Theorem 5 inside the structures S_i. This returns a superposition of the pairs in X, together with some additional quantum registers containing the structures the pairs belong to, and the data structure used by our new search algorithm. This additional data does not disturb the Grover search (see Section 3.2). In each structure, the average number of pairs in X is $2^{2\Delta_{\text{in}}-1-n+\Delta_{\text{fin}}}$. The total cost of the setup phase is

therefore $\mathrm{S} = 2^{h_T/3-2\Delta_{\mathrm{in}}/3+2/3+(n-\Delta_{\mathrm{fin}})/3}$. The rest of the attack is similar to the previous case. Putting everything together, the total running time and data complexities of the quantum truncated differential attack in the Q2 model are:

$$T_{\mathrm{Q2}}^{\mathrm{tr.\,att.}} = \max\left\{2^{h_T/2}, 2^{5h_T/6-2\Delta_{\mathrm{in}}/3+2/3}\right\} 2^{-(n-\Delta_{\mathrm{fin}})/6} + 2^{(h_T+\Delta_{\mathrm{fin}}-n)/2}\left(C^*_{k_{\mathrm{out}}} + 2^{(k-h_{\mathrm{out}})/2}\right), \tag{13}$$

$$D_{\mathrm{Q2}}^{\mathrm{tr.\,att.}} = \max\left\{2^{h_T/2}, 2^{h_T-\Delta_{\mathrm{in}}+1}\right\} 2^{-(n-\Delta_{\mathrm{fin}})/6}. \tag{14}$$

6 Applications on existing ciphers

In this section we describe three examples of classical and quantum differential attacks against block ciphers. We have chosen examples of real proposed ciphers where some of the best known attacks are simple variants of differential cryptanalysis. This allows us to illustrate the important counter-intuitive points that we want to highlight, by comparing the best classical attacks and the best quantum attacks. We first consider the block cipher used in the authenticated encryption scheme LAC [ZWW+14], and build for it a classical simple differential distinguisher and a more efficient classical truncated distinguisher. We quantize these attacks, and obtain that the quantum truncated distinguisher performs worse than a generic quantum exhaustive search. In the next application we consider the lightweight block cipher KLEIN [GNL12]. Its 64-bit key version, KLEIN-64, has been recently broken [LN14] by a truncated differential last-rounds attack. When quantizing this attack, we show that it no longer works in the quantum world, and therefore KLEIN-64 is no longer broken. Finally, we consider KLEIN-96 and the best known attack [LN14] against this cipher. We show that its quantum variant still works in the post-quantum world (both in the Q1 and the Q2 models). These applications illustrate what we previously pointed out and believe to be particularly meaningful: block ciphers with longer keys, following the natural recommendation for resisting to generic quantum attacks, are those for which the truncated attacks are more likely to still break the cryptosystem in the postquantum world. Consequently, it is crucial to understand and compute the optimized quantum complexity of the different families of attacks, as we have started doing in this paper.

6.1 Application 1: LAC

We now show an example where a truncated differential attack is more efficient than a simple differential attack using a classical computer, but the opposite is true with a quantum computer.

We consider the reduced version of LBlock [WZ11] used in LAC [ZWW+14]. According to [Leu15], the best known differential for the full 16 rounds has probability $2^{-61.5}$. This yields a classical distinguisher with complexity $2^{62.5}$ and a quantum distinguisher with complexity $2^{31.75}$. The corresponding truncated differential has the following characteristics[4]:

$$n = 64 \qquad \Delta_{\mathrm{in}} = 12 \qquad \Delta_{\mathrm{out}} = 20 \qquad \tilde{h}_T \approx 55.3$$

We note that $\tilde{h}_T > n - \Delta_{\mathrm{out}}$, which is too large to provide a working attack. However, $\tilde{h}_T$ only considers pairs following a given characteristic, and we expect additional pairs to randomly give an output difference in $\mathcal{D}_{\mathrm{out}}$. Therefore, we estimate the probability of the truncated differential as $2^{h_T} = 2^{-44} + 2^{-55.3}$. In order to check this hypothesis, we

[4] We consider the truncated differential with $\mathcal{D}_{\mathrm{in}} =$ 000000000000**0* and $\mathcal{D}_{\mathrm{out}} =$ 0000***00000**00. If the input differential is non-zero on all active bytes, a pair follows the truncated differential when 14 sums of active bytes cancel out, and 3 sums of active bytes don't cancel out. This gives a probability $(15/16)^6 \cdot (1/15)^{14} \approx 2^{-55.3}$.

implemented a reduced version of LAC with 3-bit APN S-Boxes, and verified that a bias can be detected[5]. In every structure, the probability that a pair follows the truncated differential is $2^{23} \cdot 2^{h_T} = 2^{-21} + 2^{-32.3}$, rather than 2^{-21} for a random permutation.

As explained in Section 3 (Theorem 3), this bias can be detected after examining $2 \cdot 2^{-21} \cdot 2^{32.3 \cdot 2} = 2^{44.6}$ structures, *i.e.* $2^{56.6}$ plaintexts in a classical attack (following [BGT11]). In a quantum setting, we use quantum counting [BHT98, Mos98, BHMT02] and examine $4\pi \cdot 2^{-21/2} \cdot 2^{32.3} \approx 2^{25.4}$ structures, for a total cost of $2^{25.4} \cdot 2^{2/3 \cdot 12} = 2^{33.4}$.

To summarize, the best attack in the classical setting is a truncated differential attack (with complexity $2^{60.9}$ rather than $2^{62.5}$ for a simple differential attack), while the best attack in the quantum setting is a simple differential attack (with complexity $2^{31.75}$ rather than $2^{33.4}$ for a truncated differential attack). Moreover, the quantum truncated differential attack is actually less efficient than a generic attack using Grover's algorithm.

6.2 Application 2: KLEIN-64 and KLEIN-96

6.2.1 KLEIN-64

We consider exactly the attack from [LN14]. We omit here the details of the cipher and the truncated differential, but provide the parameters needed to compute the complexity.

When taking into account the attack that provides the best time complexity, we have[6]: $h_T = 69.5$, $\Delta_{\text{in}} = 16$, $\Delta_{\text{fin}} = 32$, $k = 64$, $k_{\text{out}} = 32$, $n = 64$, $C_{k_{\text{out}}} = 2^{20}$ and $h_{\text{out}} = 45$.

In this case, we can recover the time and data complexities from the original result as[7] $D = 2^{54.5}$ and $T = 2^{54.5} + 2^{57.5} + 2^{56.5} = 2^{58.2}$, which is considerably faster than exhaustive search (2^{64}), breaking in consequence the cipher.

In the quantum scenario, the complexity of the generic exhaustive search, which we use to measure the security, is 2^{32}. The cipher is considered broken if we can retrieve the key with smaller complexity. When considering the Q2 or the Q1 case, the two last terms in the time complexity are quadratically accelerated. More precisely, the third is accelerated by square root, the second has a square root in $2^{h_T - n + \Delta_{\text{fin}}}$, which is then multiplied by $C^*_{k_{\text{out}}}$. As shown in Section 4.2.4, $C^*_{k_{\text{out}}}$ is $2^{k_{\text{out}} - h_{\text{out}}/2 + 1} = 2^{11}$ instead of 2^{20}. Consequently, the second term is also completely accelerated by a square root. But this is not the case of the first term, corresponding to data generation. In the Q1 case, it stays the same, being larger than 2^{32} and invalidating the attack. In the Q2 model, the first term becomes $2^{42.6}$, which is also clearly larger than 2^{32}, thus the attack does not work.

We have seen here an example of a primitive broken in the classical world, but remaining secure [8] in the quantum one, for both models.

6.2.2 KLEIN-96

Here we consider the attack of type III given in [LN14], as it is the only one with data complexity lower than 2^{48}, and therefore the only possible candidate for providing also an attack in the Q1 model.

[5] The truncated path for the reduced version has a probability $2^{h_T} = 2^{-33} + 2^{-40.5}$. We ran 32 experiments with 2^{31} structures of 2^9 plaintexts each. With a random function we expect about $2^{31} \cdot 2^9 \cdot (2^9 - 1)/2 \cdot 2^{-33} = 32704$ pairs satisfying the truncated differential, and about 32890 with LAC. The median number of pairs we found is 33050 and it was larger than 32704 is 31 out of 32 experiments. This agrees with our predictions.

[6] For the attacks from [LN14] on KLEIN, h_T is always bigger than $n - \Delta_{\text{in}}$, but the distinguisher from Δ_{in} to Δ_{out} still works exactly as described in Section 5.1.2 because we compare with the probability of producing the truncated differential path and not just the truncated differential.

[7] The slight difference with respect to [LN14] is because here we have not taken into account the relative cost with respect to one encryption, for the sake of simplicity.

[8] We want to point out that notions "not-secure" (i.e. can be attacked in practice) and "broken" (i.e. can be attacked faster than brute-force), are not the same, though they are difficult to dissociate.

The parameters of this classical attack are: $h_T = 78$, $\Delta_{\text{in}} = 32$, $\Delta_{\text{fin}} = 32$, $k_{\text{out}} = 48$, $n = 64$, $C_{k_{\text{out}}} = 2^{30}$ and $h_{\text{out}} = 52$. We compute and obtain the same complexities as the original results in time and data: $D = 2^{47}$ and $T = 2^{47} + 2^{46+30} + 2^{90}$. When quantizing this attack, we have to compare the complexities with $2^{96/2} = 2^{48}$.

In the Q1 model we obtain $2^{47} + 2^{23+23} + 2^{45} = 2^{47.7}$, which is lower that 2^{48}, so the attack still works. The second term comes from $C^*_{k_{\text{out}}} 2^{(h_T - n + \Delta_{\text{out}})/2}$. We can compute $C^*_{k_{\text{out}}}$ as before, obtaining $2^{48-26+1} = 2^{23}$.

In the Q2 model, the first term is reduced to 2^{39} and becomes negligible, with the final complexity at $2^{39} + 2^{46} + 2^{45} = 2^{46.6}$.

7 Linear Cryptanalysis

Linear cryptanalysis was discovered in 1992 by Matsui [MY92, Mat93]. The idea of linear cryptanalysis is to approximate the round function with a linear function, in order to find a linear approximation correlated to the non-linear encryption function E. We describe the linear approximations using linear masks; for instance, an approximation for one round is written as $E^{(1)}(x)[\chi'] \approx x[\chi]$ where χ and χ' are linear masks for the input and output, respectively, and $x[\chi] = \bigoplus_{i:\chi_i=1} x_i$. Here, "$\approx$" means that the probability that the two values are equal is significantly larger than with a random permutation.

The cryptanalyst has to build linear approximations for each round, such that the output mask of a round is equal to the input mask of the next round. The piling-up lemma is then used to evaluate the correlation of the approximation for the full cipher. As for differential cryptanalysis, we assume here that the linear approximation is given and use it with a quantum computer to obtain either a distinguishing attack or a key recovery attack. In this section, we consider linear distinguishers and key recovery attacks following from Matsui's work [Mat93].

7.1 Classical Adversary

7.1.1 Linear distinguisher

In the following, C denotes the ciphertext obtained when encrypting the plaintext P with the key K. We assume that we know a linear approximation with masks (χ_P, χ_C, χ_K) and constant term $\chi_0 \in \{0, 1\}$ satisfying $\Pr\left[C[\chi_C] = P[\chi_P] \oplus K[\chi_K] \oplus \chi_0\right] = (1 + \varepsilon)/2$, with $\varepsilon \gg 2^{-n/2}$; or, omitting the key dependency:

$$\Pr\left[C[\chi_C] = P[\chi_P]\right] = (1 \pm \varepsilon)/2.$$

An attacker can use this to distinguish E from a random permutation. The attack requires $D = A/\varepsilon^2$ known plaintexts P_i and the corresponding ciphertexts C_i, where A is a small constant (*e.g.* $A = 10$). The attacker computes the observed bias $\hat{\varepsilon} = \left|2\#\left\{i : C_i[\chi_C] = P_i[\chi_P]\right\}/D - 1\right|$, and concludes that the data is random if $\hat{\varepsilon} \leq \varepsilon/2$ and that it comes from E otherwise.

If the data is generated by a random permutation, then the expected value of $\hat{\varepsilon}$ is 0, whereas, if it is generated by E, the expected value of $\hat{\varepsilon}$ is ε. We can compute the success probability of the attack assuming that the values of $C_i[\chi_C] \oplus P_i[\chi_P]$ are identically distributed Bernoulli random variables, with parameter $1/2$ or $1/2 \pm \varepsilon$. From Hoeffding's inequality, we get:

$$\Pr\left[\hat{\varepsilon} \geq \varepsilon/2 \middle| \text{random permutation}\right] \leq 2\exp\left(-2\frac{\varepsilon^2}{4^2}D\right) \leq 2\exp\left(-\frac{A}{8}\right),$$

$$\Pr\left[\hat{\varepsilon} \leq \varepsilon/2 \middle| \text{cipher } E\right] \leq \exp\left(-2\frac{\varepsilon^2}{4^2}D\right) \leq \exp\left(-\frac{A}{8}\right);$$

both error terms can also be made arbitrarily small by increasing A.

Overall, the complexity of the linear distinguisher is

$$D_{\mathrm{C}}^{\text{lin. dist.}} = T_{\mathrm{C}}^{\text{lin. dist.}} = 1/\varepsilon^2. \tag{15}$$

As explained in Section 2, we do not take into account the factor A that depends on the success probability, and keep only the asymptotic term in the complexity.

7.1.2 Key-recovery using an r-round approximation (Matsui's Algorithm 1)

The linear distinguisher readily gives one key bit according to the sign of the bias: if $K[\chi_K] = 0$, then we expect $\#\{i : C_i[\chi_C] = P_i[\chi_P] \oplus \chi_0\} > D/2$. The attack can be repeated with different linear approximations in order to recover more key bits. If we have ℓ independent linear approximations $(\chi_P^j, \chi_C^j, \chi_K^j, \chi_0^j)$ with bias at least ε, the total complexity is:

$$D_{\mathrm{C}}^{\mathrm{Mat.1}} = 1/\varepsilon^2, \qquad T_{\mathrm{C}}^{\mathrm{Mat.1}} = \ell/\varepsilon^2 + 2^{k-\ell}. \tag{16}$$

7.1.3 Last-rounds attack (Matsui's Algorithm 2)

Alternatively, linear cryptanalysis can be used in a last-rounds attack that will often be more efficient. Following the notations of the previous sections, we consider a total of $R + r_{\text{out}}$ rounds, with an R-round linear distinguisher $(\chi_P, \chi_{C'})$ with bias ε, and we use partial decryption for the last r_{out} rounds.

We denote by k_{out} the number of key bits necessary to compute $C'[\chi_{C'}]$, where $C' = E^{-r_{\text{out}}}(C)$ from C. The attack proceeds as follows:

1. Initialize a set of $2^{k_{\text{out}}}$ counters $X_{k'}$ to zero, for each key candidate.
2. For each (P, C) pair, and for every partial key guess k', compute C' from C and k', and increment $X_{k'}$ if $P[\chi_P] = C'[\chi_{C'}]$.
3. This gives $X_{k'} = \#\left\{P, C : E_{k'}^{-r_{\text{out}}}(C)[\chi_{C'}] = P[\chi_P]\right\}$.
4. Select the partial key k' with the maximal absolute value of $X_{k'}$.

This gives the following complexity:

$$D_{\mathrm{C}}^{\mathrm{Mat.2}} = 1/\varepsilon^2 \qquad T_{\mathrm{C}}^{\mathrm{Mat.2}} = 2^{k_{\text{out}}}/\varepsilon^2 + 2^{k-k_{\text{out}}}, \tag{17}$$

where, as before, we neglect constant factors.

We note that this algorithm can be improved using a distillation phase where we count the number of occurrences of partial plaintexts and ciphertexts, and an analysis phase using only these counters rather the full data set. In some specific cases, the analysis phase can be improved by exploiting the Fast Fourier Transform [CSQ07], but we will focus on the simpler case described here.

7.2 Quantum Adversary

7.2.1 Distinguisher in the Q2 model

As in the previous sections, a speed-up for distinguishers is only observed for the Q2 model. The distinguisher is based on the quantum approximate counting algorithm of Theorem 3. As in the classical case, the goal is to distinguish between two Bernoulli distributions with parameter $1/2$ and $1/2 + \varepsilon$, respectively.

Using the quantum approximate counting algorithm, it is sufficient to make $O(1/\varepsilon)$ queries in order to achieve an ε-approximation. The data complexity of the quantum distinguisher is therefore,

$$D_{\mathrm{Q2}}^{\text{lin. dist.}} = T_{\mathrm{Q2}}^{\text{lin. dist.}} = 1/\varepsilon, \tag{18}$$

which constitutes a quadratic speed-up compared to the classical distinguisher.

7.2.2 Key-recovery using an r-round approximation in the Q1 model

Each linear relation allows the attacker to recover a bit of the key using $1/\varepsilon^2$ data, as the classical model. Once ℓ bits of the key have been recovered, one can apply Grover's algorithm to obtain the full key. For ℓ linear relations, the attack complexity is therefore:

$$D_{\mathrm{Q1}}^{\mathrm{Mat.1}} = \ell/\varepsilon^2 \qquad T_{\mathrm{Q1}}^{\mathrm{Mat.1}} = \ell/\varepsilon^2 + 2^{(k-\ell)/2}. \tag{19}$$

7.2.3 Key-recovery using an r-round approximation in the Q2 model

Each linear relation allows the attacker to recover a bit of the key using $1/\varepsilon$ data. If there are ℓ such relations, the attack complexity is:

$$D_{\mathrm{Q2}}^{\mathrm{Mat.1}} = \ell/\varepsilon \qquad T_{\mathrm{Q2}}^{\mathrm{Mat.1}} = \ell/\varepsilon + 2^{(k-\ell)/2}. \tag{20}$$

Note that we do not *a priori* obtain a quadratic improvement for the data complexity compared to the classical model. This is because the same data can be used many times in the classical model, whereas it is unclear whether something similar can be achieved using Grover's algorithm.

7.2.4 Last-rounds attack in the Q1 model

As usual for the Q1 model, one samples the same quantity of data as in the classical model and stores it in a quantum memory. Then the idea is to perform two successive instances of Grover's algorithm: the goal of the first one is to find a partial key of size k_{out} for which a bias ε is detected for the first R rounds: this has complexity $2^{k_{\mathrm{out}}/2}/\varepsilon$ with quantum counting; the second Grover aims at finding the rest of the key and has complexity $2^{(k-k_{\mathrm{out}})/2}$. Overall, the complexity of the attack is

$$D_{\mathrm{Q1}}^{\mathrm{Mat.2}} = 1/\varepsilon^2 \qquad T_{\mathrm{Q1}}^{\mathrm{Mat.2}} = 1/\varepsilon^2 + 2^{k_{\mathrm{out}}/2}/\varepsilon + 2^{(k-k_{\mathrm{out}})/2}. \tag{21}$$

7.2.5 Last-rounds attack in the Q2 model.

The strategy is similar, but the first step of the algorithm, *i.e.* finding the correct partial key, can be improved compared to the Q1 model. One uses a Grover search to obtain the partial key, and the checking step of Grover now consists of performing an approximate counting to detect the bias. Overall, the complexity of the attack is

$$D_{\mathrm{Q2}}^{\mathrm{Mat.2}} = 2^{k_{\mathrm{out}}/2}/\varepsilon \qquad T_{\mathrm{Q2}}^{\mathrm{Mat.2}} = 2^{k_{\mathrm{out}}/2}/\varepsilon + 2^{(k-k_{\mathrm{out}})/2}. \tag{22}$$

8 Discussion

In this section, we first recall all the time complexities obtained through the paper. The data complexities correspond to the first term of each expression for the differential attacks. Next, we discuss how these results affect the post-quantum security of symmetric ciphers with respect to differential and linear attacks. As a remainder, notations are given in Table 1.

Simple Differential Distinguishers:

$$T_{\mathrm{C}}^{\mathrm{s.\,dist.}} = 2^{h_S+1} \qquad T_{\mathrm{Q2}}^{\mathrm{s.\,dist.}} = 2^{h_S/2+1}$$

Simple Differential Last-Rounds Attacks:

$$\begin{aligned}
T_{\mathrm{C}}^{\mathrm{s.\,att.}} &= 2^{h_S+1} &&+ 2^{h_S+\Delta_{\mathrm{fin}}-n}\left(C_{k_{\mathrm{out}}} + 2^{k-h_{\mathrm{out}}}\right)\\
T_{\mathrm{Q1}}^{\mathrm{s.\,att.}} &= 2^{h_S+1} &&+ 2^{(h_S+\Delta_{\mathrm{fin}}-n)/2}\left(C^*_{k_{out}} + 2^{(k-h_{\mathrm{out}})/2}\right)\\
T_{\mathrm{Q2}}^{\mathrm{s.\,att.}} &= 2^{h_S/2+1} &&+ 2^{(h_S+\Delta_{\mathrm{fin}}-n)/2}\left(C^*_{k_{out}} + 2^{(k-h_{\mathrm{out}})/2}\right)
\end{aligned}$$

Truncated Differential Distinguishers:

$$T_{\mathrm{C}}^{\mathrm{tr.\,dist.}} = \max\{2^{(h_T+1)/2}, 2^{h_T-\Delta_{\mathrm{in}}+1}\} \quad T_{\mathrm{Q2}}^{\mathrm{tr.\,dist.}} = \max\left\{2^{(h_T+1)/3}, 2^{(h_T+1)/2-\Delta_{\mathrm{in}}/3}\right\}$$

Truncated Differential Last-Rounds Attacks:

$$\begin{aligned}
T_{\mathrm{C}}^{\mathrm{tr.\,att.}} &= \max\left\{2^{(h_T+1)/2}, 2^{h_T-\Delta_{\mathrm{in}}+1}\right\} &&+ 2^{h_T+\Delta_{\mathrm{fin}}-n}\left(C_{k_{\mathrm{out}}} + 2^{k-h_{\mathrm{out}}}\right)\\
T_{\mathrm{Q1}}^{\mathrm{tr.\,att.}} &= \max\left\{2^{(h_T+1)/2}, 2^{h_T-\Delta_{\mathrm{in}}+1}\right\} &&+ 2^{(h_T+\Delta_{\mathrm{fin}}-n)/2}\left(C^*_{k_{out}} + 2^{(k-h_{\mathrm{out}})/2}\right)\\
T_{\mathrm{Q2}}^{\mathrm{tr.\,att.}} &= \max\left\{2^{h_T/2}, 2^{5h_T/6-2\Delta_{\mathrm{in}}/3+2/3}\right\}2^{-(n-\Delta_{\mathrm{fin}})/6} &&+ 2^{(h_T+\Delta_{\mathrm{fin}}-n)/2}\left(C^*_{k_{\mathrm{out}}} + 2^{(k-h_{\mathrm{out}})/2}\right)
\end{aligned}$$

Linear Distinguishers:

$$T_{\mathrm{C}}^{\mathrm{lin.\,dist.}} = 1/\varepsilon^2 \qquad T_{\mathrm{Q2}}^{\mathrm{lin.\,dist.}} = 1/\varepsilon$$

Linear Attacks:

$$\begin{aligned}
T_{\mathrm{C}}^{\mathrm{Mat.1}} &= \ell/\varepsilon^2 + 2^{k-\ell} & T_{\mathrm{C}}^{\mathrm{Mat.2}} &= 2^{k_{\mathrm{out}}}/\varepsilon^2 + 2^{k-k_{\mathrm{out}}}\\
T_{\mathrm{Q1}}^{\mathrm{Mat.1}} &= \ell/\varepsilon^2 + 2^{(k-\ell)/2} & T_{\mathrm{Q1}}^{\mathrm{Mat.2}} &= 1/\varepsilon^2 + 2^{k_{\mathrm{out}}/2}/\varepsilon + 2^{(k-k_{\mathrm{out}})/2}.\\
T_{\mathrm{Q2}}^{\mathrm{Mat.1}} &= \ell/\varepsilon + 2^{(k-\ell)/2} & T_{\mathrm{Q2}}^{\mathrm{Mat.2}} &= 2^{k_{\mathrm{out}}/2}/\varepsilon + 2^{(k-k_{\mathrm{out}})/2}
\end{aligned}$$

The first observation we make is that the cost of a quantum differential or linear attack is at least the square root of the cost of the corresponding classical attack. In particular, if a block cipher is resistant to classical differential and/or linear cryptanalysis (*i.e.* classical attacks cost at least 2^k), it is also resistant to the corresponding quantum cryptanalysis (*i.e.* quantum differential and/or linear attacks cost at least $2^{k/2}$). However, a quadratic speed-up is not always possible with our techniques; in particular truncated attacks might be less accelerated than simple differential ones.

Q1 model *vs* Q2 model. We have studied quantum cryptanalysis with the notion of standard security (Q1 model with only classical encryption queries) and quantum security (Q2 model with quantum superposition queries). As expected, the Q2 model is stronger, and we often have a smaller quantum acceleration in the Q1 model. In particular, the data complexity of attack in the Q1 model is the same as the data complexity of classical attacks. Still, there are important cases where quantum differential or linear cryptanalysis can be more efficient than Grover's search in the Q1 model, which shows that quantum cryptanalysis is also relevant in the more realistic setting with only classical queries.

Quantum differential and linear attacks are more threatening to ciphers with larger key sizes. Though it seems counter-intuitive, the fact is that larger key sizes also mean higher security claims to consider a cipher as secure. In the complexity figure given above, the terms that depend on the key size (the right hand size terms) are likely to be the bottleneck for ciphers with long keys with respect to the internal state size. In all the

attacks studied here, this term is quadratically improved using quantum computation, in both models. Therefore, attacks against those ciphers will get the most benefits from quantum computers. We illustrated this effect in Section 6.2, by studying KLEIN with two different key sizes.

This effect is very strong in the Q1 model because most attacks have a data complexity larger than $2^{n/2}$ (because $h_S > n/2$, $h_T > n/2$, or $\varepsilon < 2^{-n/4}$). If the keysize is equal to n, this makes those attacks less efficient than Grover's search, but they become interesting when k is larger than n. In particular, with $k \geq 2n$, the data complexity is always smaller than $2^{k/2}$.

This observation is particularly relevant because the recommended strategy against quantum adversaries is to use longer keys [ABB+15]. We show that with this strategy, it is likely that classical attacks that break the cryptosystem lead to quantum attacks that also break it, even in the Q1 model where the adversary only makes classical queries to the oracle.

The best attack might change from the classical to the quantum world. Since truncated differential attacks use collision finding in the data analysis step, they do not enjoy a quadratic improvement in the quantum setting. Therefore, as we show in Section 6.1, a truncated differential attack might be the best known attack in the classical world, while the simple differential might become the best in the quantum world. In particular, simply quantizing the best known attack does not ensure obtaining the best possible attack in the post-quantum world, which emphasizes the importance of studying quantum symmetric cryptanalysis.

More strikingly, there are cases where differential attacks are more efficient than brute force in the classical world, but quantum differential attacks are not faster than Grover's algorithm, as we show in the example of Section 6.2.1.

9 Conclusion and open questions

Our work is an important step towards building a quantum symmetric cryptanalysis toolbox. Our results have corroborated our first intuition that symmetric cryptography does not seem ready for the post-quantum world. This not a direct conclusion from the paper, though indirectly the first logical approach for quantum symmetric cryptanalysis would be to quantize the best classical attack, and that would simplify the task. As we know for sure applications where the best attacks might change exist, cryptanalysis must be started anew. The non-intuitive behaviors shown in our examples of applications help to illustrate the importance of understanding how symmetric attacks work in the quantum world, and therefore, of our results. For building trust against quantum adversaries, this work should be extended, and other classical attacks should be investigated. Indeed, we have concluded that quantizing the best known classical differential attacks may not give the best quantum attack. This emphasizes the importance of studying and finding the best quantum attacks, including all known families of cryptanalysis.

We have devised quantum attacks that break classical cryptosystems faster than a quantum exhaustive search. However, the quantum-walk-based techniques used here can only lead to polynomial speed-ups, and the largest gap is quadratic, achieved by Grover's algorithm. Although this is significant, it can not be interpreted as a collapse of cryptography against quantum adversaries similar to public-key cryptography based on the hardness of factoring. However, we already mentioned that attacks based on the quantum Fourier transform, which is at the core of Shor's algorithm for factoring and does not fall in the framework of quantum walks, have been found for symmetric ciphers [KM10, KM12, RS15, KLLNP16].

We end by mentioning a few open questions that we leave for future work. In this work, we have studied quantum versions of differential and linear cryptanalysis. In each of these cases, we were either given a differential characteristics or a linear approximation to begin with, and used quantum algorithms to exploit them to perform a key recovery attack for instance. A natural question is whether quantum computers can also be useful to come up with good differential characteristics or linear approximations in the first place.

So far, we have only scratched the surface of linear cryptanalysis by quantizing the simplest versions of classical attacks, that is excluding more involved constructions using counters or the fast Fourier transform. Of course, since the quantum Fourier transform offers a significant speed-up compared to its classical counterpart, it makes sense to investigate whether it can be used to obtain more efficient quantum linear cryptanalysis.

A major open question in the field of quantum cryptanalysis is certainly the choice of the right model of attack. In this work, we investigated two such models. The Q2 model might appear rather extreme and perhaps even unrealistic since it is unclear why an attacker could access the cipher in superposition. But this model has the advantage of consistency. Also, a cipher secure in this model will remain secure in any setting. On the other hand, the Q1 model appears more realistic, but might be a little bit too simplistic. In particular, it seems important to better understand the interface between the classical register that stores the data that have been obtained by querying the cipher and the quantum register where they must be transferred in order to be further processed by the quantum computer.

References

[ABB+15] Daniel Augot, Lejla Batina, Daniel J Bernstein, Joppe Bos, Johannes Buchmann, Wouter Castryck, Orr Dunkelman, Tim Güneysu, Shay Gueron, Andreas Hülsing, et al. Initial recommendations of long-term secure post-quantum systems. *Available at* http://pqcrypto.eu.org/docs/initial-recommendations.pdf, 2015.

[Amb07] A. Ambainis. Quantum walk algorithm for element distinctness. *SIAM J. Comput.*, 37(1):210–239, 2007.

[ATTU16] Mayuresh Vivekanand Anand, Ehsan Ebrahimi Targhi, Gelo Noel Tabia, and Dominique Unruh. Post-quantum security of the CBC, CFB, OFB, CTR, and XTS modes of operation. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*, volume 9606 of *Lecture Notes in Computer Science*, pages 44–63. Springer, 2016.

[BGT11] Céline Blondeau, Benoît Gérard, and Jean-Pierre Tillich. Accurate estimates of the data complexity and success probability for various cryptanalyses. *Des. Codes Cryptography*, 59(1-3):3–34, 2011.

[BHMT02] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. In *Quantum computation and information (Washington, DC, 2000)*, volume 305 of *Contemp. Math.*, pages 53–74. Amer. Math. Soc., Providence, RI, 2002.

[BHT98] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum counting. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings*, volume 1443 of *Lecture Notes in Computer Science*, pages 820–831. Springer, 1998.

[BS90] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.

[BZ13a] Dan Boneh and Mark Zhandry. Quantum-secure message authentication codes. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 592–608. Springer, 2013.

[BZ13b] Dan Boneh and Mark Zhandry. Secure signatures and chosen ciphertext security in a quantum computing world. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 361–379. Springer, 2013.

[CSQ07] Baudoin Collard, François-Xavier Standaert, and Jean-Jacques Quisquater. Improving the time complexity of matsui's linear cryptanalysis. In Kil-Hyun Nam and Gwangsoo Rhee, editors, *Information Security and Cryptology - ICISC 2007, 10th International Conference, Seoul, Korea, November 29-30, 2007, Proceedings*, volume 4817 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2007.

[DFNS13] Ivan Damgård, Jakob Funder, Jesper Buus Nielsen, and Louis Salvail. Superposition attacks on cryptographic protocols. In Carles Padró, editor, *Information Theoretic Security - 7th International Conference, ICITS 2013, Singapore, November 28-30, 2013, Proceedings*, volume 8317 of *Lecture Notes in Computer Science*, pages 142–161. Springer, 2013.

[GHS15] Tommaso Gagliardoni, Andreas Hülsing, and Christian Schaffner. Semantic security and indistinguishability in the quantum world. *arXiv preprint arXiv:1504.05255*, 2015.

[GNL12] Zheng Gong, Svetla Nikova, and Yee Wei Law. KLEIN: A new family of lightweight block ciphers. In *RFID. Security and Privacy - 7th International Workshop, RFIDSec 2011, Amherst, USA, June 26-28, 2011, Revised Selected Papers*, volume 7055 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2012.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 212–219. ACM, 1996.

[JKM13] S. Jeffery, R. Kothari, and F. Magniez. Nested quantum walks with quantum data structures. In *Proceedings of 24th AMC-SIAM symposium on discrete algorithms*, 2013.

[Kap14] Marc Kaplan. Quantum attacks against iterated block ciphers. *CoRR*, abs/1410.1434, 2014.

[KLLNP16] Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. Breaking symmetric cryptosystems using quantum period finding. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016 (to appear)*, Lecture Notes in Computer Science. Springer, 2016.

[KM10] H. Kuwakado and M. Morii. Quantum distinguisher between the 3-round Feistel cipher and the random permutation. In *Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on*, pages 2682–2685, June 2010.

[KM12] H. Kuwakado and M. Morii. Security on the quantum-type Even-Mansour cipher. In *Information Theory and its Applications (ISITA), 2012 International Symposium on*, pages 312–316, Oct 2012.

[Knu94] Lars R. Knudsen. Truncated and higher order differentials. In Bart Preneel, editor, *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1994.

[Leu15] Gaëtan Leurent. Differential forgery attack against LAC. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, volume 9566 of *Lecture Notes in Computer Science*, pages 217–224. Springer, 2015.

[LN14] Virginie Lallemand and María Naya-Plasencia. Cryptanalysis of KLEIN. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*, pages 451–470. Springer, 2014.

[Mat93] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993.

[Mos98] Michele Mosca. Quantum searching, counting and amplitude amplification by eigenvector analysis. In *MFCS'98 workshop on Randomized Algorithms*, pages 90–100, 1998.

[MY92] Mitsuru Matsui and Atsuhiro Yamagishi. A new method for known plaintext attack of FEAL cipher. In *Advances in Cryptology - EUROCRYPT '92, Workshop on the Theory and Application of of Cryptographic Techniques, Balatonfüred, Hungary, May 24-28, 1992, Proceedings*, pages 81–91, 1992.

[RS15] Martin Roetteler and Rainer Steinwandt. A note on quantum related-key attacks. *Information Processing Letters*, 115(1):40–44, 2015.

[San08] Miklos Santha. Quantum walk based search algorithms. In *Theory and Applications of Models of Computation*, pages 31–46. Springer, 2008.

[Sho97] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.

[Sim97] Daniel R Simon. On the power of quantum computation. *SIAM journal on computing*, 26(5):1474–1483, 1997.

[SS16] Thomas Santoli and Christian Schaffner. Using simon's algorithm to attack symmetric-key cryptographic primitives. *arXiv preprint arXiv:1603.07856*, 2016.

[WZ11] Wenling Wu and Lei Zhang. Lblock: A lightweight block cipher. In *Applied Cryptography and Network Security - 9th International Conference, ACNS 2011, Nerja, Spain, June 7-10, 2011. Proceedings*, volume 6715 of *Lecture Notes in Computer Science*, pages 327–344, 2011.

[Zha12] Mark Zhandry. How to construct quantum random functions. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 679–687. IEEE Computer Society, 2012.

[ZWW+14] Lei Zhang, Wenling Wu, Yanfeng Wang, Shengbao Wu, and Jian Zhang. LAC: A Lightweight Authenticated Encryption Cipher. Submission to CAESAR. Available from: http://competitions.cr.yp.to/round1/lacv1.pdf (v1), March 2014.

Breaking Symmetric Cryptosystems using Quantum Period Finding

Marc Kaplan[1,2], Gaëtan Leurent[3] Anthony Leverrier[3], and María Naya-Plasencia[3]

[1] LTCI, Télécom ParisTech, 23 avenue d'Italie, 75214 Paris CEDEX 13, France
[2] School of Informatics, University of Edinburgh,
10 Crichton Street, Edinburgh EH8 9AB, UK
[3] Inria Paris, France

Abstract. Due to Shor's algorithm, quantum computers are a severe threat for public key cryptography. This motivated the cryptographic community to search for quantum-safe solutions. On the other hand, the impact of quantum computing on secret key cryptography is much less understood. In this paper, we consider attacks where an adversary can query an oracle implementing a cryptographic primitive in a quantum superposition of different states. This model gives a lot of power to the adversary, but recent results show that it is nonetheless possible to build secure cryptosystems in it.

We study applications of a quantum procedure called *Simon's algorithm* (the simplest quantum period finding algorithm) in order to attack symmetric cryptosystems in this model. Following previous works in this direction, we show that several classical attacks based on finding collisions can be dramatically sped up using Simon's algorithm: finding a collision requires $\Omega(2^{n/2})$ queries in the classical setting, but when collisions happen with some hidden periodicity, they can be found with only $O(n)$ queries in the quantum model.

We obtain attacks with very strong implications. First, we show that the most widely used modes of operation for authentication and authenticated encryption (*e.g.* CBC-MAC, PMAC, GMAC, GCM, and OCB) are completely broken in this security model. Our attacks are also applicable to many CAESAR candidates: CLOC, AEZ, COPA, OTR, POET, OMD, and Minalpher. This is quite surprising compared to the situation with encryption modes: Anand *et al.* show that standard modes are secure with a quantum-secure PRF.

Second, we show that Simon's algorithm can also be applied to slide attacks, leading to an exponential speed-up of a classical symmetric cryptanalysis technique in the quantum model.

Keywords: post-quantum cryptography, symmetric cryptography, quantum attacks, block ciphers, modes of operation, slide attack.

1 Introduction

The goal of post-quantum cryptography is to prepare cryptographic primitives to resist quantum adversaries, *i.e.* adversaries with access to a quantum com-

puter. Indeed, cryptography would be particularly affected by the development of large-scale quantum computers. While currently used asymmetric cryptographic primitives would suffer from devastating attacks due to Shor's algorithm [42], the status of symmetric ones is not so clear: generic attacks, which define the security of ideal symmetric primitives, would get a quadratic speed-up thanks to Grover's algorithm [23], hinting that doubling the key length could restore an equivalent ideal security in the post-quantum world. Even though the community seems to consider the issue settled with this solution [6], only very little is known about real world attacks, that determine the real security of used primitives. Very recently, this direction has started to draw attention, and interesting results have been obtained. New theoretical frameworks to take into account quantum adversaries have been developed [11,12,19,22,15,2].

Simon's algorithm [43] is central in quantum algorithm theory. Historically, it was an important milestone in the discovery by Shor of his celebrated quantum algorithm to solve integer factorization in polynomial time [42]. Interestingly, Simon's algorithm has also been applied in the context of symmetric cryptography. It was first used to break the 3-round Feistel construction [30] and then to prove that the Even-Mansour construction [31] is insecure with superposition queries. While Simon's problem (which is the problem solved with Simon's algorithm) might seem artificial at first sight, it appears in certain constructions in symmetric cryptography, in which ciphers and modes typically involve a lot of structure.

These first results, although quite striking, are not sufficient for evaluating the security of actual ciphers. Indeed, the confidence we have on symmetric ciphers depends on the amount of cryptanalysis that was performed on the primitive. Only this effort allows researchers to define the security margin which measures how far the construction is from being broken. Thanks to the large and always updated cryptanalysis toolbox built over the years in the *classical* world, we have solid evaluations of the security of the primitives against classical adversaries. This is, however, no longer the case in the post-quantum world, *i.e.* when considering quantum adversaries.

We therefore need to build a complete cryptanalysis toolbox for quantum adversaries, similar to what has been done for the classical world. This is a fundamental step in order to correctly evaluate the post-quantum security of current ciphers and to design new secure ciphers for the post-quantum world.

Our results. We make progresses in this direction, and open new surprising and important ranges of applications for Simon's algorithm in symmetric cryptography:

1. The original formulation of Simon's algorithm is for functions whose collisions happen only at some hidden period. We extend it to functions that have more collisions. This leads to a better analysis of previous applications of Simon's algorithm in symmetric cryptography.
2. We then show an attack against the LRW construction, used to turn a blockcipher into a tweakable block cipher [32]. Like the results on 3-round Feistel

and Even-Mansour, this is an example of construction with provable security in the classical setting that becomes insecure against a quantum adversary.

3. Next, we study block cipher modes of operation. We show that some of the most common modes for message authentication and authenticated encryption are completely broken in this setting. We describe forgery attacks against standardized modes (CBC-MAC, PMAC, GMAC, GCM, and OCB), and against several CAESAR candidates, with complexity only $O(n)$, where n is the size of the block. In particular, this partially answers an open question by Boneh and Zhandry [13]: "Do the CBC-MAC or NMAC constructions give quantum-secure PRFs?".
   Those results are in stark contrast with a recent analysis of encryption modes in the same setting: Anand *et al.* show that some classical encryption modes are secure against a quantum adversary when using a quantum-secure PRF [3]. Our results imply that some authentication and authenticated encryption schemes remain insecure with *any* block cipher.
4. The last application is a quantization of slide attacks, a popular family of cryptanalysis that is independent of the number of rounds of the attacked cipher. Our result is the first exponential speed-up obtained directly by a quantization of a classical cryptanalysis technique, with complexity dropping from $O(2^{n/2})$ to $O(n)$, where n is the size of the block.

These results imply that for the symmetric primitives we analyze, doubling the key length is not sufficient to restore security against quantum adversaries. A significant effort on quantum cryptanalysis of symmetric primitives is thus crucial for our long-term trust in these cryptosystems.

The attack model. We consider attacks against classical cryptosystems using quantum resources. This general setting broadly defines the field of post-quantum cryptography. But attacking specific cryptosystems requires a more precise definition of the operations the adversary is allowed to perform. The simplest setting allows the adversary to perform local quantum computation. For instance, this can be modeled by the quantum random oracle model, in which the adversary can query the oracle in an arbitrary superposition of the inputs [11,14,48,44]. A more practical setting allows quantum queries to the hash function used to instantiate the oracle on a quantum computer.

We consider here a much stronger model in which, in addition to local quantum operations, an adversary is granted an access to a possibly remote cryptographic oracle in superposition of the inputs, and obtains the corresponding superposition of outputs. In more detail, if the encryption oracle is described by a classical function $\mathcal{O}_k : \{0,1\}^n \to \{0,1\}^n$, then the adversary can make standard quantum queries $|x\rangle|y\rangle \mapsto |x\rangle|\mathcal{O}_k(x) \oplus y\rangle$, where x and y are arbitrary n-bit strings and $|x\rangle$, $|y\rangle$ are the corresponding n-qubit states expressed in the computational basis. A circuit representing the oracle is given in Figure 1. Moreover, any superposition $\sum_{x,y} \lambda_{x,y}|x\rangle|y\rangle$ is a valid input to the quantum oracle, who then returns $\sum_{x,y} \lambda_{x,y}|x\rangle|y \oplus O_k(x)\rangle$. In previous works, these attacks have been called *superposition attacks* [19], *quantum chosen message attacks* [13] or *quantum security* [47].

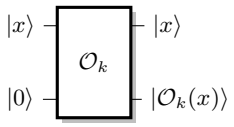


Fig. 1. The quantum cryptographic oracle.

Simon's algorithm requires the preparation of the uniform superposition of all n-bit strings, $\frac{1}{\sqrt{2^n}} \sum_x |x\rangle|0\rangle$[4]. For this input, the quantum encryption oracle returns $\frac{1}{\sqrt{2^n}} \sum_x |x\rangle|\mathcal{O}_k(x)\rangle$, the superposition of all possible pairs of plaintext-ciphertext. It might seem at first that this model gives an overwhelming power to the adversary and is therefore uninteresting. Note, however, that the laws of quantum mechanics imply that the measurement of such a $2n$-qubit state can only reveal $2n$ bits of information, making this model nontrivial.

The simplicity of this model, together with the fact that it encompasses any reasonable model of quantum attacks makes it very interesting. For instance, [12] gave constructions of message authenticated codes that remain secure against superposition attacks. A similar approach was initiated by [19], who showed how to construct secure multiparty protocols when an adversary can corrupt the parties in superposition. A protocol that is proven secure in this model may truthfully be used in a quantum world.

Our work shows that superposition attacks, although they are not trivial, allow new powerful strategies for the adversary. Modes of operation that are provably secure against classical attacks can then be broken. There exist a few options to prevent the attacks that we present here. A possibility is to forbid all kind of quantum access to a cryptographic oracle. In a world where quantum resources become available, this restriction requires a careful attention. This can be achieved for example by performing a quantum measurement of any incoming quantum query to the oracle. But this task involves meticulous engineering of quantum devices whose outcome remains uncertain. Even information theoretically secure quantum cryptography remains vulnerable to attacks on their implementations, as shown by attacks on quantum key distribution [49,34,45].

A more realistic approach is to develop a set of protocols that remains secure against superposition attacks. Another advantage of this approach is that it also covers more advanced scenarios, for example when an encryption device is given to the adversary as an obfuscated algorithm. Our work shows how important it is to develop protocols that remain secure against superposition attacks.

Regarding symmetric cryptanalysis, we have already mentioned the protocol of Boneh and Zhandry for MACs that remains secure against superposition attacks. In particular, we answer negatively to their question asking wether CBC-MAC is secure in their model. Generic quantum attacks against symmetric cryptosystems have also been considered. For instance, [27] studies the security of iterated block ciphers, and Anand et al. investigated the security of various

[4] When there is no ambiguity, we write $|0\rangle$ for the state $|0 \dots 0\rangle$ of appropriate length.

modes of operations for encryption against superposition attacks [3]. They show that OFB and CTR remain secure, while CBC and CFB are not secure in general (with attacks involving Simon's algorithm), but are secure if the underlying PRF is quantum secure. Recently, [28] considers symmetric families of cryptanalysis, describing quantum versions of differential and linear attacks.

Cryptographic notions like indistinguishability or semantic security are well understood in a classical world. However, they become difficult to formalize when considering quantum adversaries. The quantum chosen message model is a good framework to study these [22,15,2].

In this paper, we consider forgery attacks: the goal of the attacker is to forge a tag for some arbitrary message, without the knowledge of the secret key. In a quantum setting, we follow the EUF-qCMA security definition that was given by Boneh and Zhandry [12]. A message authentication code is broken by a quantum existential forgery attack if after q queries to the cryptographic oracle, the adversary can generate at least $q+1$ valid messages with corresponding tags.

Organization. The paper is organized as follows. First, Section 2 introduces Simon's algorithm and explains how to modify it in order to handle functions that only approximately satisfy Simon's promise. This variant seems more appropriate for symmetric cryptography and may be of independent interest. Section 3 summarizes known quantum attacks against various constructions in symmetric cryptography. Section 4 presents the attack against the LRW constructions. In Section 5, we show how Simon's algorithm can be used to obtain devastating attacks on several widely used modes of operations: CBC-MAC, PMAC, GMAC, GCM, OCB, as well as several CAESAR candidates. Section 6 shows the application of the algorithm to slide attacks, providing an exponential speed-up. The paper ends in Section 7 with a conclusion, pointing out possible new directions and applications.

2 Simon's algorithm and attack strategy

In this section, we present Simon's problem [43] and the quantum algorithm for efficiently solving it. The simplest version of our attacks directly exploits this algorithm in order to recover some secret value of the encryption algorithm. Previous works have already considered such attacks against 3-round Feistel schemes and the Even-Mansour construction (see Section 3 for details).

Unfortunately, it is not always possible to recast an attack in terms of Simon's problem. More precisely, Simon's problem is a promise problem, and in many cases, the relevant promise (that only a structured class of collisions can occur) is not satisfied, far from it in fact. We show in Theorem 1 below that, however, these additional collisions do not lead to a significant increase of the complexity of our attacks.

2.1 Simon's problem and algorithm

We first describe Simon's problem, and then the quantum algorithm for solving it. We refer the reader to the recent review by Montanaro and de Wolf on quantum

property testing for various applications of this algorithm [37]. We assume here a basic knowledge of the quantum circuit model. We denote the addition and multiplication in a field with 2^n elements by "$\oplus$" and "$\cdot$", respectively.

We consider that the access to the input of Simon's problem, a function f, is made by querying it. A classical query oracle is a function $x \mapsto f(x)$. To run Simon's algorithm, it is required that the function f can be queried quantum-mechanically. More precisely, it is supposed that the algorithm can make arbitrary quantum superpositions of queries of the form $|x\rangle|0\rangle \mapsto |x\rangle|f(x)\rangle$.

Simon's problem is the following:

Simon's problem: Given a Boolean function $f : \{0,1\}^n \to \{0,1\}^n$ and the promise that there exists $s \in \{0,1\}^n$ such that for any $(x,y) \in \{0,1\}^n$, $[f(x) = f(y)] \Leftrightarrow [x \oplus y \in \{0^n, s\}]$, the goal is to find s.

This problem can be solved classically by searching for collisions. The optimal time to solve it is therefore $\Theta(2^{n/2})$. On the other hand, Simon's algorithm solves this problem with quantum complexity $O(n)$. Recall that the Hadamard transform $H^{\otimes n}$ applied on an n-qubit state $|x\rangle$ for some $x \in \{0,1\}^n$ gives $H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y\in\{0,1\}^n} (-1)^{x\cdot y}|y\rangle$, where $x \cdot y := x_1 y_1 \oplus \cdots \oplus x_n y_n$.

The algorithm repeats the following five quantum steps.

1. Starting with a $2n$-qubit state $|0\rangle|0\rangle$, one applies a Hadamard transform $H^{\otimes n}$ to the first register to obtain the quantum superposition
$$\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle|0\rangle.$$
2. A quantum query to the function f maps this to the state
$$\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle|f(x)\rangle.$$
3. Measuring the second register in the computational basis yields a value $f(z)$ and collapses the first register to the state:
$$\frac{1}{\sqrt{2}}(|z\rangle + |z \oplus s\rangle).$$
4. Applying again the Hadamard transform $H^{\otimes n}$ to the first register gives:
$$\frac{1}{\sqrt{2}} \frac{1}{\sqrt{2^n}} \sum_{y\in\{0,1\}^n} (-1)^{y\cdot z} \left(1 + (-1)^{y\cdot s}\right) |y\rangle.$$
5. The vectors y such that $y \cdot s = 1$ have amplitude 0. Therefore, measuring the state in the computational basis yields a random vector y such that $y \cdot s = 0$.

By repeating this subroutine $O(n)$ times, one obtains $n-1$ independent vectors orthogonal to s with high probability, and s can be recovered using basic linear algebra. Theorem 1 gives the trade-off between the number of repetitions of the subroutine and the success probability of the algorithm.

2.2 Dealing with unwanted collisions

In our cryptanalysis scenario, it is not always the case that the promise of Simon's problem is perfectly satisfied. More precisely, by construction, there will always exist an s such that $f(x) = f(x \oplus s)$ for any input x, but there might be many more collisions than those of this form. If the number of such unwanted collisions is too large, one might not be able to obtain a full rank linear system of equations from Simon's subroutine after $O(n)$ queries. Theorem 1 rules this out provided that f does not have too many collisions of the form $f(x) = f(x \oplus t)$ for some $t \notin \{0, s\}$.

For $f : \{0,1\}^n \to \{0,1\}^n$ such that $f(x \oplus s) = f(x)$ for all x, consider

$$\varepsilon(f, s) = \max_{t \in \{0,1\}^n \setminus \{0,s\}} \Pr_x[f(x) = f(x \oplus t)]. \tag{1}$$

This parameter quantifies how far the function is from satisfying Simon's promise. For a random function, one expects $\varepsilon(f, s) = \Theta(n2^{-n})$, following the analysis of [18]. On the other hand, for a constant function, $\varepsilon(f, s) = 1$ and it is impossible to recover s.

The following theorem, whose proof can be found in Appendix A, shows the effect of unwanted collisions on the success probability of Simon's algorithm.

Theorem 1 (Simon's algorithm with approximate promise). *If $\varepsilon(f, s) \leq p_0 < 1$, then Simon's algorithm returns s with cn queries, with probability at least $1 - \left(2\left(\frac{1+p_0}{2}\right)^c\right)^n$.*

In particular, choosing $c \geq 3/(1 - p_0)$ ensures that the error decreases exponentially with n. To apply our results, it is therefore sufficient to prove that $\varepsilon(f, s)$ is bounded away from 1.

Finally, if we apply Simon's algorithm without any bound on $\varepsilon(f, s)$, we can not always recover s unambiguously. Still if we select a random value t orthogonal to all vectors u_i returned by each step of the algorithm, t satisfy $f(x \oplus t) = f(x)$ with high probability.

Theorem 2 (Simon's algorithm without promise). *After cn steps of Simon's algorithm, if t is orthogonal to all vectors u_i returned by each step of the algorithm, then $\Pr_x[f(x \oplus t) = f(t)] \geq p_0$ with probability at least $1 - \left(2\left(\frac{1+p_0}{2}\right)^c\right)^n$.*

In particular, choosing $c \geq 3/(1 - p_0)$ ensures that the probability is exponentially close to 1.

2.3 Attack strategy

The general strategy behind our attacks exploiting Simon's algorithm is to start with the encryption oracle $E_k : \{0,1\}^n \to \{0,1\}^n$ and exhibit a new function f that satisfies Simon's promise with two additional properties: the adversary should be able to query f in superposition if he has quantum oracle access to E_k, and the knowledge of the string s should be sufficient to break the cryptographic scheme. In the following, this function is called Simon's function.

In most cases, our attacks correspond to a classical collision attack. In particular, the value s will usually be the difference in the internal state after processing a fixed pair of messages (α_0, α_1), *i.e.* $s = E(\alpha_0) \oplus E(\alpha_1)$. The input of f will be inserted into the state with the difference s so that $f(x) = f(x \oplus s)$.

In our work, this function f is of the form:

$$\begin{aligned} f^1 : x \quad &\mapsto P(\widetilde{E}(x) + \widetilde{E}(x \oplus s)) \quad \text{or,} \\ f^2 : b, x \quad &\mapsto \begin{cases} \widetilde{E}(x) & \text{if } b = 0, \\ \widetilde{E}(x \oplus s) & \text{if } b = 1, \end{cases} \end{aligned}$$

where $\widetilde{E}$ is a simple function obtained from E_k and P a permutation. It is immediate to see that f^1 and f^2 have periods s for f^1 or $1||s$ for f^2.

In most applications, Simon's function satisfies $f(x) = f(y)$ for $y \oplus x \in \{0, s\}$, but also for additional inputs x, y. Theorem 1 extends Simon's algorithm precisely to this case. In particular, if the additional collisions of f are random, then Simon's algorithm is successful. When considering explicit constructions, we can not in general prove that the unwanted collisions *are* random, but rather that they *look random enough*. In practice, if the function $\varepsilon(f, s)$ is not bounded, then some of the primitives used in the construction have are far from ideal. We can show that this happens with low probability, and would imply an classical attack against the system. Applying Theorem 1 is not trivial, but it stretches the range of application of Simon's algorithm far beyond its original version.

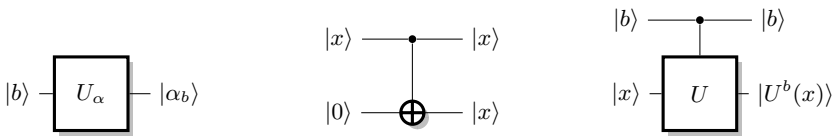
Construction of Simon's functions. To make our attacks as clear as possible, we provide the diagrams of circuits computing the function f. These circuits use a little number of basic building blocks represented in Figure 2.

In our attacks, we often use a pair of arbitrary constants α_0 and α_1. The choice of the constant is indexed by a bit b. We denote by U_α the gate that maps b to α_b (See Figure 2.1). For simplicity, we ignore here the additional qubits required in practice to make the transform reversible through padding.

Although it is well known that arbitrary quantum states cannot be cloned, we use the $CNOT$ gate to copy classical information. More precisely, a CNOT gate can copy states in the computational basis: $CNOT : |x\rangle|0\rangle \to |x\rangle|x\rangle$. This transform is represented in Figure 2.2.

Finally, any unitary transform U can be controlled by a bit b. This operation, denoted U^b maps x to $U(x)$ if $b = 1$ and leaves x unchanged otherwise. In the quantum setting, the qubit $|b\rangle$ can be in a superposition of 0 and 1, resulting in a superposition of $|x\rangle$ and $|U(x)\rangle$. The attacks that we present in the following sections only make use of this procedure when the attacker knows a classical description of the unitary to be controlled. In particular, we do not apply it to the cryptographic oracle.

When computing Simon's function, *i.e.* the function f on which Simon's algorithm is applied, the registers containing the value of f must be unentangled with any other working register. Otherwise, these registers, which might hinder the periodicity of the function, have to be taken into account in Simon's algorithm and the whole procedure could fail.



2.1. One-to-one mapping. **2.2.** CNOT gate. **2.3.** Controlled Unitary.

Fig. 2. Circuit representation of basic building blocks.

3 Previous works

Previous works have used Simon's algorithm to break the security of classical constructions in symmetric cryptography: the Even-Mansour construction and the 3-round Feistel scheme. We now explain how these attacks work with our terminology and extend two of the results. First, we show that the attack on the Feistel scheme can be extended to work with random functions, where the original analysis held only for random permutations. Second, using our analysis Simon's algorithm with approximate promise, we make the number of queries required to attack the Even-Mansour construction more precise. These observations have been independently made by Santoli and Schaffner [40]. They use a slightly different approach, which consists in analyzing the run of Simon's algorithm for these specific cases.

3.1 Applications to a three-round Feistel scheme

The Feistel scheme is a classical construction to build a random permutation out of random functions or random permutations. In a seminal work, Luby and Rackoff proved that a three-round Feistel scheme is a secure pseudo-random permutation [33].

A three-round Feistel scheme with input (x_L, x_R) and output $(y_L, y_R) = E(x_L, x_R)$ is built from three round functions R_1, R_2, R_3 as (see Figure 3):

$$(u_0, v_0) = (x_L, x_R), \quad (u_i, v_i) = (v_{i-1} \oplus R_i(u_{i-1}), u_{i-1}), \quad (y_L, y_R) = (u_3, v_3).$$

In order to distinguish a Feistel scheme from a random permutation in a quantum setting, Kuwakado and Morii [30] consider the case were the R_i are permutations, and define the following function, with two arbitrary constants α_0 and α_1 such that $\alpha_0 \neq \alpha_1$:

$$\begin{aligned} f : \{0,1\} \times \{0,1\}^n &\to \{0,1\}^n \\ b, x \quad &\mapsto y_R \oplus \alpha_b, \quad \text{where } (y_R, y_L) = E(\alpha_b, x) \\ f(b,x) &= R_2(x \oplus R_1(\alpha_b)) \end{aligned}$$

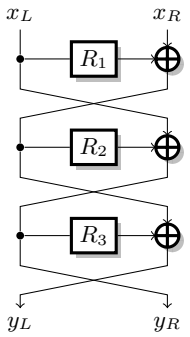


Fig. 3. Three-round Feistel scheme.

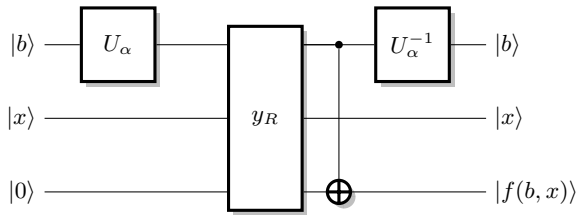


Fig. 4. Simon's function for Feistel.

In particular, this f satisfies $f(b,x) = f(b \oplus 1, x \oplus R_1(\alpha_0) \oplus R_1(\alpha_1))$. Moreover,

$$\begin{aligned} f(b',x') = f(b,x) &\Leftrightarrow x' \oplus R_1(\alpha_{b'}) = x \oplus R_1(\alpha_b) \\ &\Leftrightarrow \begin{cases} x' \oplus x = 0 & \text{if } b' = b \\ x' \oplus x = R_1(\alpha_0) \oplus R_1(\alpha_1) & \text{if } b' \neq b \end{cases} \end{aligned}$$

Therefore, the function satisfies Simon's promise with $s = 1 \parallel R_1(\alpha_0) \oplus R_1(\alpha_1)$, and we can recover $R_1(\alpha_0) \oplus R_1(\alpha_1)$ using Simon's algorithm. This gives a distinguisher, because Simon's algorithm applied to a random permutation returns zero with high probability. This can be seen from Theorem 2, using the fact that with overwhelming probability[18], there is no value $t \neq 0$ such that $\Pr_x[f(x \oplus t) = f(x)] > 1/2$ for a random permutation f.

We can also verify that the value $R_1(\alpha_0) \oplus R_1(\alpha_1)$ is correct with two additional classical queries $(y_L, y_R) = E(\alpha_0, x)$ and $(y'_L, y'_R) = E(\alpha_1, x \oplus R_1(\alpha_0) \oplus R_1(\alpha_1))$ for a random x. If the value is correct, we have $y_R \oplus y'_R = \alpha_0 \oplus \alpha_1$.

Note that in their attack, Kuwakado and Morii implicitly assume that the adversary can query in superposition an oracle that returns solely the left part y_L of the encryption. If the adversary only has access to the complete encryption oracle E, then a query in superposition would return two *entangled* registers containing the left and right parts, respectively. In principle, Simon's algorithm requires the register containing the input value to be completely disentangled from the others.

Feistel scheme with random functions. Kuwakado and Morii [30] analyze only the case where the round functions R_i are permutations. We now extend this analysis to *random functions* R_i. The function f defined above still satisfies $f(b,x) = f(b \oplus 1, x \oplus R_1(\alpha_0) \oplus R_1(\alpha_1))$, but it doesn't satisfy the exact promise of Simon's algorithm: there are additional collisions in f, between inputs with random differences. However, the previous distinguisher is still valid: at the end of Simon's algorithm, there exist at least one non-zero value orthogonal to all

the values y measured at each step: s. This would not be the case with a random permutation.

Moreover, we can show that $\varepsilon(f, 1 \,\|\, s) < 1/2$ with overwhelming probability, so that Simon's algorithm still recovers $1 \,\|\, s$ following Theorem 1. If $\varepsilon(f, 1 \,\|\, s) > 1/2$, there exists (τ, t) with $(\tau, t) \notin \{(0,0), (1,s)\}$ such that: $\Pr[f(b,x) = f(b \oplus \tau, x \oplus t)] > 1/2$. Assume first that $\tau = 0$, this implies:

$$\Pr[f(0,x) = f(0, x \oplus t)] > 1/2 \quad \text{or} \quad \Pr[f(1,x) = f(1, x \oplus t)] > 1/2.$$

Therefore, for some b, $\Pr[R_2(x \oplus R_1(\alpha_b)) = R_2(x \oplus t \oplus R_1(\alpha_b))] > 1/2$, *i.e.* $\Pr[R_2(x) = R_2(x \oplus t)] > 1/2$. Similarly, if $\tau = 1$, $\Pr[R_2(x \oplus R_1(\alpha_0)) = R_2(x \oplus t \oplus R_1(\alpha_1))] > 1/2$, *i.e.* $\Pr[R_2(x) = R_2(x \oplus t \oplus R_1(\alpha_0) \oplus R_1(\alpha_1))] > 1/2$.

To summarize, if $\varepsilon(f, 1 \,\|\, s) > 1/2$, there exists $u \neq 0$ such that $\Pr[R_2(x) = R_2(x \oplus u)] > 1/2$. This only happens with negligible probability for a random choice of R_2 as shown in [18].

3.2 Application to the Even-Mansour construction

The Even-Mansour construction is a simple construction to build a block cipher from a public permutation [21]. For some permutation P, the cipher is:

$$E_{k_1,k_2}(x) = P(x \oplus k_1) \oplus k_2.$$

Even and Mansour have shown that this construction is secure in the random permutation model, up to $2^{n/2}$ queries, where n is the size of the input to P.

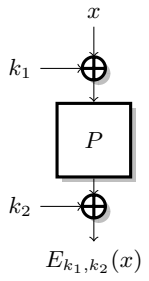


Fig. 5. Even-Mansour scheme.

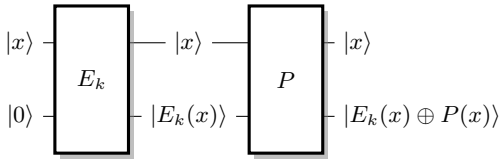


Fig. 6. Simon's function for Even-Mansour.

However, Kuwakado and Morii [31] have shown that the security of this construction collapses if an adversary can query an encryption oracle with a superposition of states. More precisely, they define the following function:

$$\begin{aligned} f : \{0,1\}^n &\to \{0,1\}^n \\ x &\mapsto E_{k_1,k_2}(x) \oplus P(x) = P(x \oplus k_1) \oplus P(x) \oplus k_2. \end{aligned}$$

In particular, f satisfies $f(x \oplus k_1) = f(x)$ (interestingly, the slide with a twist attack of Biryukov and Wagner[8] uses the same property). However, there are additional collisions in f between inputs with random differences. As in the attack against the Feistel scheme with random round functions, we use Theorem 1, to show that Simon's algorithm recovers k_1[5].

We show that $\varepsilon(f, k_1) < 1/2$ with overwhelming probability for a random permutation P, and if $\varepsilon(f, k_1) > 1/2$, then there exists a classical attack against the Even-Mansour scheme. Assume that $\varepsilon(f, k_1) > 1/2$, that is, there exists t with $t \notin \{0, k_1\}$ such that $\Pr[f(x) = f(x \oplus t)] > 1/2$, *i.e.*,

$$p = \Pr[P(x) \oplus P(x \oplus k_1) \oplus P(x \oplus t) \oplus P(x \oplus t \oplus k_1) = 0] > 1/2.$$

This correspond to higher order differential for P with probability $1/2$, which only happens with negligible probability for a random choice of P. In addition, this would imply the existence of a simple classical attack against the scheme:

1. Query $y = E_{k_1,k_2}(x)$ and $y' = E_{k_1,k_2}(x \oplus t)$
2. Then $y \oplus y' = P(x) \oplus P(x \oplus t)$ with probability at least one half

Therefore, for any instantiation of the Even-Mansour scheme with a fixed P, either there exist a classical distinguishing attack (this only happens with negligible probability with a random P), or Simon's algorithm successfully recovers k_1. In the second case, the value of k_2 can then be recovered from an additional classical query: $k_2 = E(x) \oplus P(x \oplus k_1)$.

In the next sections, we give new applications of Simon's algorithm, to break various symmetric cryptography schemes.

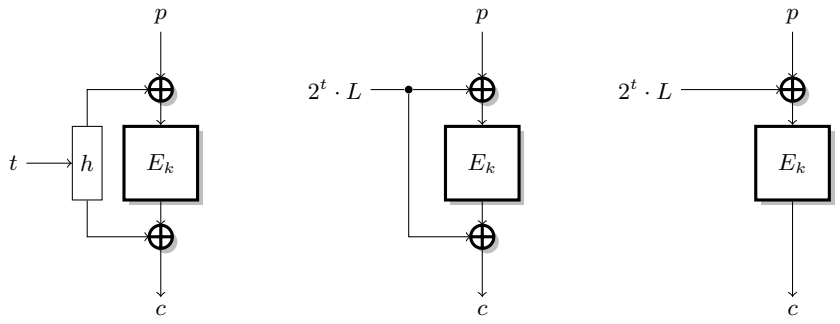
4 Application to the LRW construction

We now show a new application of Simon's algorithm to the LRW construction. The LRW construction, introduced by Liskov, Rivest and Wagner [32], turns a block cipher into a tweakable block cipher, *i.e.* a family of unrelated block ciphers. The tweakable block cipher is a very useful primitive to build modes for encryption, authentication, or authenticated encryption. In particular, tweakable block ciphers and the LRW construction were inspired by the first version of OCB, and later versions of OCB use the tweakable block ciphers formalism. The LRW construction uses a (almost) universal hash function h (which is part of the key), and is defined as (see also Figure 7):

$$\widetilde{E}_{t,k}(x) = E_k(x \oplus h(t)) \oplus h(t).$$

We now show that the LRW construction is not secure in a quantum setting. We fix two arbitrary tweaks t_0, t_1, with $t_0 \neq t_1$, and we define the following

[5] Note that Kuwakado and Morii just assume that each step of Simon's algorithm gives a random vector orthogonal to k_1. Our analysis is more formal and captures the conditions on P required for the algorithm to be successful.



7.1. LRW construction. **7.2.** XEX construction. **7.3.** XE construction.

Fig. 7. The LRW construction, and efficient instantiations XEX (CCA secure) and XE (only CPA secure).

function:

$$\begin{aligned} f : \{0,1\}^n &\to \{0,1\}^n \\ x \quad &\mapsto \widetilde{E}_{t_0,k}(x) \oplus \widetilde{E}_{t_1,k}(x) \\ f(x) &= E_k\big(x \oplus h(t_0)\big) \oplus h(t_0) \oplus E_k\big(x \oplus h(t_1)\big) \oplus h(t_1). \end{aligned}$$

Given a superposition access to an oracle for an LRW tweakable block cipher, we can build a circuit implementing this function, using the construction given in Figure 8. In the circuit, the cryptographic oracle $\widetilde{E}_{t,k}$ takes two inputs: the block x to be encrypted and the tweak t. Since the tweak comes out of $\widetilde{E}_{t,k}$ unentangled with the other register, we do not represent this output in the diagram. In practice, the output is forgotten by the attacker.

It is easy to see that this function satisfies $f(x) = f(x \oplus s)$ with $s = h(t_0) \oplus h(t_1)$. Furthermore, the quantity $\varepsilon(f, s) = \max_{t \in \{0,1\}^n \setminus \{0,s\}} \Pr[f(x) = f(x \oplus t)]$ is bounded with overwhelming probability, assuming that E_k behaves as a random permutation. Indeed if $\varepsilon(f, s) > 1/2$, there exists some t with $t \notin \{0, s\}$ such that $\Pr[f(x) = f(x \oplus t)] > 1/2$, *i.e.*,

$$\Pr[E_k\big(x\big) \oplus E_k\big(x \oplus s\big) \oplus E_k\big(x \oplus t\big)\big) \oplus E_k\big(x \oplus t \oplus s\big) = 0] > 1/2$$

This correspond to higher order differential for E_k with probability $1/2$, which only happens with negligible probability for a random permutation. Therefore, if E is a pseudo-random permutation family, $\varepsilon(f, s) \leq 1/2$ with overwhelming probability, and running Simon's algorithm with the function f returns $h(t_0) \oplus h(t_1)$. The assumption that E behaves as a PRP family is required for the security proof of LRW, so it is reasonable to make the same assumption in an attack. More concretely, a block cipher with a higher order differential with probability $1/2$ as seen above would probably be broken by classical attacks. The attack is not immediate because the differential can depend on the key, but it would seem to

indicate a structural weakness. In the following sections, some attacks can also be mounted using Theorem 2 without any assumptions on E.

In any case, there exist at least one non-zero value orthogonal to all the values y measured during Simon's algorithm: s. This would not be the case if f is a random function, which gives a distinguisher between the LRW construction and an ideal tweakable block cipher with $O(n)$ quantum queries to $\widetilde{E}$.

In practice, most instantiations of LRW use a finite field multiplication to define the universal hash function h, with a secret offset L (usually computed as $L = E_k(0)$). Two popular constructions are:

- $h(t) = \gamma(t) \cdot L$, used in OCB1 [39], OCB3 [29] and PMAC [10], with a Gray encoding γ of t,
- $h(t) = 2^t \cdot L$, the XEX construction, used in OCB2 [38].

In both cases, we can recover L from the value $h(t_0) \oplus h(t_1)$ given by the attack.

This attack is important, because many recent modes of operation are inspired by the LRW construction, and the XE and XEX instantiations, such as CAESAR candidates AEZ [24], COPA [4], OCB [29], OTR [36], Minalpher [41], OMD [17], and POET [1]. We will see in the next section that variants of this attack can be applied to each of these modes.

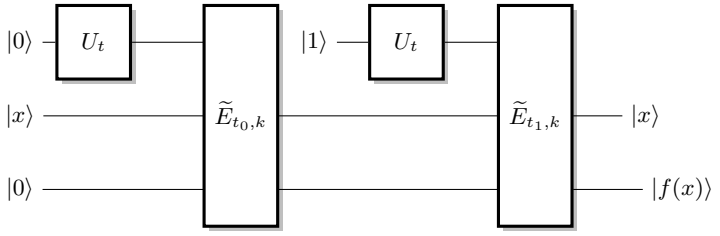


Fig. 8. Simon's function for LRW.

5 Application to block cipher modes of operations

We now give new applications of Simon's algorithm to the security of block cipher modes of operations. In particular, we show how to break the most popular and widely used block-cipher based MACs, and message authentication schemes: CBC-MAC (including variants such as XCBC [9], OMAC [25], and CMAC [20]), GMAC [35], PMAC [10], GCM [35] and OCB [29]. We also show attacks against several CAESAR candidates. In each case, the mode is proven secure up to $2^{n/2}$ in the classical setting, but we show how, by a reduction to Simon's problem, forgery attacks can be performed with superposition queries at a cost of $O(n)$.

Notations and preliminaries. We consider a block cipher E_k, acting on blocks of length n, where the subscript k denotes the key. For simplicity, we

only describe the modes with full-block messages, the attacks can trivially be extended to the more general modes with arbitrary inputs. In general, we consider a message M divided into ℓ n-bits block: $M = m_1 \| \dots \| m_\ell$. We also assume that the MAC is not truncated, *i.e.* the output size is n bits. In most cases, the attacks can be adapted to truncated MACS.

5.1 Deterministic MACs: CBC-MAC and PMAC

We start with deterministic Message Authentication Codes, or MACs. A MAC is used to guarantee the authenticity of messages, and should be immune against forgery attacks. The standard security model is that it should be hard to forge a message with a valid tag, even given access to an oracle that computes the MAC of any chosen message (of course the forged message must not have been queried to the oracle).

To translate this security notion to the quantum setting, we assume that the adversary is given an oracle that takes a quantum superposition of messages as input, and computes the superposition of the corresponding MAC.

CBC-MAC. CBC-MAC is one of the first MAC constructions, inspired by the CBC encryption mode. Since the basic CBC-MAC is only secure when the queries are prefix-free, there are many variants of CBC-MAC to provide security for arbitrary messages. In the following we describe the Encrypted-CBC-MAC variant [5], using two keys k and k', but the attack can be easily adapted to other variants [9,25,20]. On a message $M = m_1 \| \dots \| m_\ell$, CBC-MAC is defined as (see Figure 9):

$$x_0 = 0 \qquad x_i = E_k(x_{i-1} \oplus m_i) \qquad \text{CBC-MAC}(M) = E_{k'}(x_\ell)$$

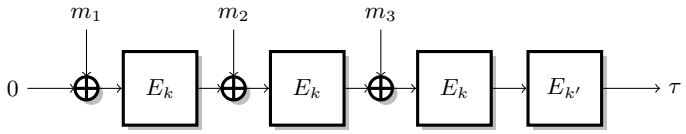


Fig. 9. Encrypt-last-block CBC-MAC.

CBC-MAC is standardized and widely used. It has been proved to be secure up to the birthday bound [5], assuming that the block cipher is indistinguishable from a random permutation.

Attack. We can build a powerful forgery attack on CBC-MAC with very low complexity using superposition queries. We fix two arbitrary message blocks α_0, α_1, with $\alpha_0 \neq \alpha_1$, and we define the following function:

$$\begin{aligned} f : \{0,1\} \times \{0,1\}^n &\to \{0,1\}^n \\ b, x \quad &\mapsto \text{CBC-MAC}(\alpha_b \| x) = E_{k'}\left(E_k\big(x \oplus E_k(\alpha_b)\big)\right). \end{aligned}$$

The function f can be computed with a single call to the cryptographic oracle, and we can build a quantum circuit for f given a black box quantum circuit for CBC-MAC$_k$. Moreover, f satisfies the promise of Simon's problem with $s = 1 \,\|\, E_k(\alpha_0) \oplus E_k(\alpha_1)$:

$$\begin{aligned} f(0,x) &= E_{k'}(E_k(x \oplus E_k(\alpha_1))), \\ f(1,x) &= E_{k'}(E_k(x \oplus E_k(\alpha_0))), \\ f(b,x) &= f(b \oplus 1, x \oplus E_k(\alpha_0) \oplus E_k(\alpha_1)). \end{aligned}$$

More precisely:

$$\begin{aligned} f(b',x') = f(b,x) &\Leftrightarrow x \oplus E_k(\alpha_b) = x' \oplus E_k(\alpha_{b'}) \\ &\Leftrightarrow \begin{cases} x' \oplus x = 0 & \text{if } b' = b \\ x' \oplus x = E_k(\alpha_0) \oplus E_k(\alpha_1) & \text{if } b' \neq b \end{cases} \end{aligned}$$

Therefore, an application of Simon's algorithm returns $E_k(\alpha_0) \oplus E_k(\alpha_1)$. This allows to forge messages easily:

1. Query the tag of $\alpha_0 \,\|\, m_1$ for an arbitrary block m_1;
2. The same tag is valid for $\alpha_1 \,\|\, m_1 \oplus E_k(\alpha_0) \oplus E_k(\alpha_1)$.

In order to break the formal notion of EUF-qCMA security, we must produce $q+1$ valid tags with only q queries to the oracle. Let $q' = O(n)$ denote the number of of quantum queries made to learn $E_k(\alpha_0) \oplus E_k(\alpha_1)$. The attacker will repeats the forgery step step $q'+1$ times, in order to produce $2(q'+1)$ messages with valid tags, after a total of $2q'+1$ classical and quantum queries to the cryptographic oracle. Therefore, CBC-MAC is broken by a quantum existential forgery attack.

After some exchange at early stages of the work, an extension of this forgery attack has been found by Santoli and Schaffner [40]. Its main advantage is to handle oracles that accept input of fixed length, while our attack works for oracles accepting messages of variable length.

PMAC. PMAC is a parallelizable block-cipher based MAC designed by Rogway [38]. PMAC is based on the XE construction: the construction uses secret offsets Δ_i derived from the secret key to turn the block cipher into a tweakable block cipher. More precisely, the PMAC algorithm is defined as

$$c_i = E_k(m_i \oplus \Delta_i) \qquad \text{PMAC}(M) = E_k^*\big(m_\ell \oplus \sum c_i\big)$$

where E^* is a tweaked variant of E. We omit the generation of the secret offsets because they are irrelevant to our attack.

First attack. When PMAC is used with two-block messages, it has the same structure as CBC-MAC: $\text{PMAC}(m_1 \,\|\, m_2) = E_k^*(m_2 \oplus E_k(m_1 \oplus \Delta_0))$. Therefore we can use the attack of the previous section to recover $E_k(\alpha_0) \oplus E_k(\alpha_1)$ for arbitrary values of α_0 and α_1. Again, this leads to a simple forgery attack. First, query the tag of $\alpha_0 \,\|\, m_1 \,\|\, m_2$ for arbitrary blocks m_1, m_2. The same tag is

valid for $\alpha_1 \parallel m_1 \parallel m_2 \oplus E_k(\alpha_0) \oplus E_k(\alpha_1)$. As for CBC-MAC, these two steps can be repeated $t+1$ times, where t is the number of quantum queries issued. The adversary then produces $2(t+1)$ messages after only $2t+1$ queries to the cryptographic oracle.

Second attack. We can also build another forgery attack on PMAC where we recover the difference between two offsets Δ_i, following the attack against LRW given in Section 4. More precisely, we use the following function:

$$\begin{aligned} f : \{0,1\}^n &\to \{0,1\}^n \\ m \quad &\mapsto \mathrm{PMAC}(m \parallel m \parallel 0^n) = E_k^* \left(E_k(m \oplus \Delta_0) \oplus E_k(m \oplus \Delta_1)\right). \end{aligned}$$

In particular, it satisfies $f(m \oplus s) = f(m)$ with $s = \Delta_0 \oplus \Delta_1$. Furthermore, we can show that $\varepsilon(f, s) \leq 1/2$ when E is a good block cipher[6], and we can apply Simon's algorithm to recover $\Delta_0 \oplus \Delta_1$. This allows to create forgeries as follows:

1. Query the tag of $m_1 \parallel m_1$ for an arbitrary block m_1;
2. The same tag is valid for $m_1 \oplus \Delta_0 \oplus \Delta_1 \parallel m_1 \oplus \Delta_0 \oplus \Delta_1$.

As mentioned in Section 4, the offsets in PMAC are defined as $\Delta_i = \gamma(i) \cdot L$, with $L = E_k(0)$ and γ a Gray encoding. This allows to recover L from $\Delta_0 \oplus \Delta_1$, as $L = (\Delta_0 \oplus \Delta_1) \cdot (\gamma(0) \oplus \gamma(1))^{-1}$. Then we can compute all the values Δ_i, and forge arbitrary messages.

We can also mount an attack without any assumption on $\varepsilon(f, s)$, using Theorem 2. Indeed, with a proper choice of parameters, Simon's algorithm will return a value $t \neq 0$ that satisfies $\Pr_x[f(x \oplus t) = f(x)] \geq 1/2$. This value is not necessarily equal to s, but it can also be used to create forgeries in the same way, with success probability at least $1/2$.

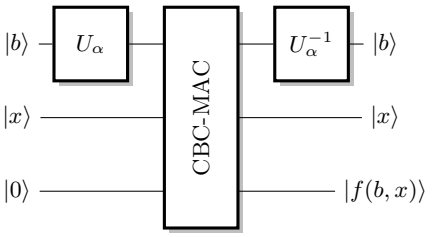


Fig. 10. Simon's function for CBC-MAC.

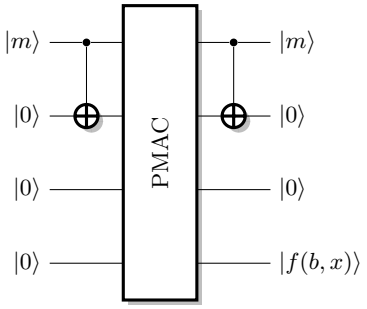


Fig. 11. Simon's function for the second attack against PMAC.

[6] Since this attack is just a special case of the LRW attack of Section 4, we don't repeat the detailed proof.

5.2 Randomized MAC: GMAC

GMAC is the underlying MAC of the widely used GCM standard, designed by McGrew and Viega [35], and standardized by NIST. GMAC follows the Carter-Wegman construction [16]: it is built from a universal hash function, using polynomial evaluation in a Galois field. As opposed to the constructions of the previous sections, GMAC is a randomized MAC; it requires a second input N, which must be non-repeating (a nonce). GMAC is essentially defined as:

$$\begin{aligned}\mathrm{GMAC}(N, M) &= \mathrm{GHASH}(M \parallel \mathrm{len}(M)) \oplus E_k(N||1)\\ \mathrm{GHASH}(M) &= \sum_{i=1}^{\mathrm{len}(M)} m_i \cdot H^{\mathrm{len}(M)-i+1} \quad \text{with } H = E_k(0),\end{aligned}$$

where $\mathrm{len}(M)$ is the length of M.

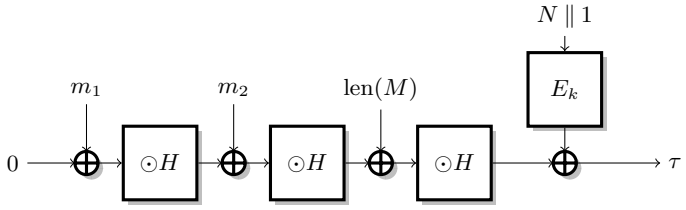


Fig. 12. GMAC

Attack. When the polynomial is evaluated with Horner's rule, the structure of GMAC is similar to that of CBC-MAC (see Figure 12). For a two-block message, we have $\mathrm{GMAC}(m_1 \parallel m_2) = \big((m_1 \cdot H) \oplus m_2\big) \cdot H \oplus E_k(N \parallel 1)$. Therefore, we us the same f as in the CBC-MAC attack, with fixed blocks α_0 and α_1:

$$\begin{aligned}f_N : \{0,1\} \times \{0,1\}^n &\to \{0,1\}^n\\ b, x \quad &\mapsto \mathrm{GMAC}(N, \alpha_b \parallel x) = \alpha_b \cdot H^2 \oplus x \cdot H \oplus E_k(N||1).\end{aligned}$$

In particular, we have:

$$\begin{aligned}f(b', x') = f(b, x) &\Leftrightarrow \alpha_b \cdot H^2 \oplus x \cdot H = \alpha_{b'} \cdot H^2 \oplus x' \cdot H\\ &\Leftrightarrow \begin{cases} x' \oplus x = 0 & \text{if } b' = b\\ x' \oplus x = (\alpha_0 \oplus \alpha_1) \cdot H & \text{if } b' \neq b \end{cases}\end{aligned}$$

Therefore f_N satisfies the promise of Simon's algorithm with $s = 1 \,\|\, (\alpha_0 \oplus \alpha_1) \cdot H$.

Role of the nonce. There is an important caveat regarding the use of the nonce. In a classical setting, the nonce is chosen by the adversary under the constraint that it is non-repeating, *i.e.* the oracle computes $N, M \mapsto \mathrm{GMAC}(N, M)$. However, in the quantum setting, we don't have a clear definition of non-repeating

if the nonce can be in superposition. To sidestep the issue, we use a weaker security notion where the nonce is chosen at random by the oracle, rather than by the adversary (following the IND-qCPA definition of [13]). The oracle is then $M \mapsto (r, \mathrm{GMAC}(r, M))$. If we can break the scheme in this model, the attack will also be valid with any reasonable CPA security definition.

In this setting we can access the function f_N only for a random value of N. In particular, we cannot apply Simon's algorithm as is, because this requires $O(n)$ queries to the *same* function f_N. However, a single step of Simon's algorithm requires a single query to the f_N function, and returns a vector orthogonal to s, for any random choice of N. Therefore, we can recover $(\alpha_0 \oplus \alpha_1) \cdot H$ after $O(n)$ steps, even if each step uses a different value of N. Then, we can recover H easily, and it is easy to generate forgeries when H is known:

1. Query the tag of $N, m_1 \| m_2$ for arbitrary blocks m_1, m_2 (under a random nonce N).
2. The same tag is valid for $m_1 \oplus 1 \| m_2 \oplus H$ (with the same nonce N).

As for CBC-MAC, repeating these two steps leads to an existential forgery attack.

5.3 Classical Authenticated Encryption Schemes: GCM and OCB

We now give applications of Simon's algorithm to break the security of standardized authenticated encryption modes. The attacks are similar to the attacks against authentication modes, but these authenticated encryption modes are nonce-based. Therefore we have to pay special attention to the nonce, as in the attack against GMAC. In the following, we assume that the nonce is randomly chosen by the MAC oracle, in order to avoid issues with the definition of non-repeating nonce in a quantum setting.

Extending MAC attacks to authenticated encryption schemes. We first present a generic way to apply MAC attacks in the context of an authenticated encryption scheme. More precisely, we assume that the tag of the authenticated encryption scheme is computed as $f(g(A), h(M, N))$, *i.e.* the authentication of the associated data A is independent of the nonce N. This is the case in many practical schemes (*e.g.* GCM, OCB) for efficiency reasons.

In this setting, we can use a technique similar to our attack against GMAC: we define a function $M \mapsto f_N(M)$ for a fixed nonce N, such that for any nonce N, $f_N(M) = f_N(M \oplus \Delta)$ for some secret value Δ. Next we use Simon's algorithm to recover Δ, where each step of Simon's algorithm is run with a random nonce, and returns a vector orthogonal to Δ. Finally, we can recover Δ, and if f_N was carefully built, the knowledge of Δ is sufficient for a forgery attack.

The CCM mode is a notable exception, where all the computations depend on the nonce. In particular, there is no obvious way to apply our attacks to CCM.

Extending GMAC attack to GCM. GCM is one of the most widely used authenticated encryption modes, designed by McGrew and Viega [35]. GMAC is the composition of the counter mode for encryption with GMAC (computed over the associated data and the ciphertext) for authentication.

In particular, when the message is empty, GCM is just GMAC, and we can use the attack of the previous section to recover the hash key H. This immediately allows a forgery attack.

OCB. OCB is another popular authenticated encryption mode, with a very high efficiency, designed by Rogaway *et al.* [39,38,29]. Indeed, OCB requires only ℓ block cipher calls to process an ℓ-block message, while GCM requires ℓ block cipher calls, and ℓ finite field operations. OCB is build from the LRW construction discussed in Section 4. OCB takes as input a nonce N, a message $M = m_1 \| \dots \| m_\ell$, and associated data $A = a_1 \| \dots a_@$, and returns a ciphertext $C = c_1 \| \dots \| c_\ell$ and a tag τ:

$$c_i = E_k(m_i \oplus \Delta_i^N) \oplus \Delta_i^N, \quad \tau = E_k\left(\Delta_\ell'^N \oplus \sum m_i\right) \oplus \sum b_i, \quad b_i = E_k(a_i \oplus \Delta_i).$$

Extending PMAC attack to OCB. In particular, when the message is empty, OCB reduces to a randomized variant of PMAC:

$$\mathrm{OCB}_k(N, \varepsilon, A) = \phi_k(N) \oplus \sum b_i, \qquad b_i = E_k(a_i \oplus \Delta_i).$$

Note that the Δ_i values used for the associated data are independent of the nonce N. Therefore, we can apply the second PMAC attack previously given, using the following function:

$$\begin{aligned} f_N : \{0,1\}^n &\to \{0,1\}^n \\ x &\mapsto \mathrm{OCB}_k(N, \varepsilon, x \| x) \\ f_N(x) &= E_k(x \oplus \Delta_0) \oplus E_k(x \oplus \Delta_1) \oplus \phi_k(N) \end{aligned}$$

Again, this is a special case of the LRW attack of Section 4. The family of functions satisfies $f_N(a \oplus \Delta_0 \oplus \Delta_1) = f_N(a)$, for any N, and $\varepsilon(f_N, \Delta_0 \oplus \Delta_1) \leq 1/2$ with overwhelming probability if E is a PRP. Therefore we can use the variant of Simon's algorithm to recover $\Delta_0 \oplus \Delta_1$. Two messages with valid tags can then be generated by a single classical queries:

1. Query the authenticated encryption C, τ of $M, a \| a$ for an arbitrary message M, and an arbitrary block a (under a random nonce N).
2. C, τ is also a valid authenticated encryption of $M, a \oplus \Delta_0 \oplus \Delta_1 \| a \oplus \Delta_0 \oplus \Delta_1$, with the same nonce N.

Repeating these steps lead again to an existential forgery attack.

Alternative attack against OCB. For some versions of OCB, we can also mount a different attack targeting the encryption part rather than the authentication part. The goal of this attack is also to recover the secret offsets, but we

target the Δ_i^N used for the encryption of the message. More precisely, we use the following function:

$$\begin{aligned} f_i : \{0,1\}^n &\to \{0,1\}^n \\ m \quad &\mapsto c_1 \oplus c_2, \text{where } (c_1, c_2, \tau) = \mathrm{OCB}_k(N, m \parallel m, \varepsilon) \\ f_i(m) &= E_k(m \oplus \Delta_1^N) \oplus \Delta_1^N \oplus E_k(m \oplus \Delta_2^N) \oplus \Delta_2^N \end{aligned}$$

This function satisfies $f_N(m \oplus \Delta_1^N \oplus \Delta_2^N) = f_N(m)$ and $\varepsilon(f_N, \Delta_0^N \oplus \Delta_1^N) \leq 1/2$, with the same arguments as previously. Moreover, in OCB1 and OCB3, the offsets are derived as $\Delta_i^N = \Phi_k(N) \oplus \gamma(i) \cdot E_k(0)$ for some function Φ (based on the block cipher E_k). In particular, $\Delta_1^N \oplus \Delta_2^N$ is independent of N:

$$\Delta_1^N \oplus \Delta_2^N = (\gamma(1) \oplus \gamma(2)) \cdot E_k(0).$$

Therefore, we can apply Simon's algorithm to recover $\Delta_1^N \oplus \Delta_2^N$. Again, this leads to a forgery attack, by repeating the following two steps:

1. Query the authenticated encryption $c_1 \parallel c_2, \tau$ of $m \parallel m, A$ for an arbitrary block m, and arbitrary associated data A (under a random nonce N).
2. $c_2 \oplus \Delta_0^N \oplus \Delta_1^N \parallel c_1 \oplus \Delta_0^N \oplus \Delta_1^N, \tau$ is also a valid authenticated encryption of $m \oplus \Delta_0^N \oplus \Delta_1^N \parallel m \oplus \Delta_0^N \oplus \Delta_1^N, A$ with the same nonce N.

The forgery is valid because we swap the inputs of the first and second block ciphers. In addition, we have $\sum m_i = \sum m'_i$, so that the tag is still valid.

5.4 New Authenticated Encryption Schemes: CAESAR Candidates

In this section, we consider recent proposals for authenticated encryption, submitted to the ongoing CAESAR competition. Secret key cryptography has a long tradition of competitions: AES and SHA-3 for example, were chosen after the NIST competitions organized in 1997 and 2007, respectively. The CAESAR competition[7] aims at stimulating research on authenticated encryption schemes, and to define a portfolio of new authenticated encryption schemes. The competition is currently in the second round, with 29 remaining algorithms.

First, we point out that the attacks of the previous sections can be used to break several CAESAR candidates:

- CLOC [26] uses CBC-MAC to authenticate the message, and the associated data is processed independently of the nonce. Therefore, the CBC-MAC attack can be extended to CLOC[8].
- AEZ [24], COPA [4], OTR [36] and POET [1] use a variant of PMAC to authenticate the associated data. In both cases, the nonce is not used to process the associated data, so that we can extend the PMAC attack as we did against OCB[9].

[7] `http://competitions.cr.yp.to/`

[8] This is not the case for the related mode SILC, because the nonce is processed before the data in CBC-MAC.

[9] Note that AEZ, COPA and POET also claim security when the nonce is misused, but our attacks are nonce-respecting.

– The authentication of associated data in OMD [17] and Minalpher [41] are also variants of PMAC (with a PRF that is not block cipher), and the attack can be applied.

In the next section, we show how to adapt the PMAC attack to Minalpher and OMD, since the primitives are different.

Minalpher. Minalpher [41] is a permutation-based CAESAR candidate, where the permutation is used to build a tweakable block-cipher using the tweakable Even-Mansour construction. When the message is empty (or fixed), the authentication part of Minalpher is very similar to PMAC. With associated data $A = a_1 \| \dots a_@$, the tag is computed as:

$$b_i = P(a_i \oplus \Delta_i) \oplus \Delta_i \qquad \tau = \phi_k \left(N, M, a_@ \oplus \sum_{i=1}^{@-1} b_i \right)$$
$$\Delta_i = y^i \cdot L' \qquad L' = P(k \| 0) \oplus (k \| 0)$$

where ϕ_k is a permutation (we omit the description of ϕ_k because it is irrelevant for our attack). Since the tag is a function of $a_@ \oplus \sum_{i=1}^{@-1} b_i$, we can use the same attacks as against PMAC. For instance, we define the following function:

$$\begin{aligned} f_N : \{0,1\} \times \{0,1\}^n &\to \{0,1\}^n \\ b, x \quad &\mapsto \text{Minalpher}(N, \varepsilon, \alpha_b \| x) = \phi_k(N, \varepsilon, P(\alpha_b \oplus \Delta_1) \oplus \Delta_1 \oplus x). \end{aligned}$$

In particular, we have:

$$\begin{aligned} f_N(b', x') = f_N(b, x) &\Leftrightarrow P(\alpha_{b'} \oplus \Delta_1) \oplus x' = P(\alpha_b \oplus \Delta_1) \oplus x \\ &\Leftrightarrow \begin{cases} x' \oplus x = 0 & \text{if } b' = b \\ x' \oplus x = P(\alpha_0 \oplus \Delta_1) \oplus P(\alpha_1 \oplus \Delta_1) & \text{if } b' \neq b \end{cases} \end{aligned}$$

Since $s = P(\alpha_0 \oplus \Delta_1) \oplus P(\alpha_1 \oplus \Delta_1)$ is independent of N, we can easily apply Simon's algorithm to recover s, and generate forgeries.

OMD. OMD [17] is a compression-function-based CAESAR candidate. The internal primitive is a keyed compression function denoted F_k. Again, when the message is empty the authentication is very similar to PMAC. With associated data $A = a_1 \| \dots a_@$, the tag is computed as:

$$b_i = F_k(a_i \oplus \Delta_i) \qquad \tau = \phi_k(N, M) \oplus \sum b_i$$

We note that the Δ_i used for the associated data do not depend on the nonce. Therefore we can use the second PMAC attack with the following function:

$$\begin{aligned} f_N : \{0,1\}^n &\to \{0,1\}^n \\ x \quad &\mapsto \text{OMD}(N, \varepsilon, x \| x) \\ f_N(x) &= \phi_k(N, \varepsilon) \oplus F_k(x \oplus \Delta_1) \oplus F_k(x \oplus \Delta_2) \end{aligned}$$

This is the same form as seen when extending the PMAC attack to OCB, therefore we can apply the same attack to recover $s = \Delta_1 \oplus \Delta_2$ and generate forgeries.

6 Simon's algorithm applied to slide attacks

In this section we show how Simon's algorithm can be applied to a cryptanalysis family: slide attacks. In this case, the complexity of the attack drops again exponentially, from $O(2^{n/2})$ to $O(n)$ and therefore becomes much more dangerous. To the best of our knowledge this is the first symmetric cryptanalytic technique that has an exponential speed-up in the post-quantum world.

The principle of slide attacks In 1999, Wagner and Biryukov introduced the technique called *slide attack* [7]. It can be applied to block ciphers made of r applications of an identical round function R, each one parametrized by the same key K. The attack works independently of the number of rounds, r. Intuitively, for the attack to work, R has to be vulnerable to known plaintext attacks.

The attacker collects $2^{n/2}$ encryptions of plaintexts. Amongst these couples of plaintext-ciphertext, with large probability, he gets a "slid" pair, that is, a pair of couples (P_0, C_0) and (P_1, C_1) such that $R(P_0) = P_1$. This immediately implies that $R(C_0) = C_1$. For the attack to work, the function R needs to allow for an efficient recognition of such pairs, which in turns makes the key extraction from R easy. A trivial application of this attack is the key-alternate cipher with blocks of n bits, identical subkeys and no round constants. The complexity is then approximately $2^{n/2}$. The speed-up over exhaustive search given by this attack is then quadratic, similar to the quantum attack based on Grover's algorithm.

This attack is successful, for example, to break the TREYFER block cipher [46], with a data complexity of 2^{32} and a time complexity of $2^{32+12} = 2^{44}$ (where 2^{12} is the cost of identifying the slid pair by performing some key guesses). Comparatively, the cost for an exhaustive search of the key is 2^{64}.

Exponential quantum speed-up of slide attacks We consider the attack represented in Figure 13. The unkeyed round function is denoted P and the whole encryption function E_k.

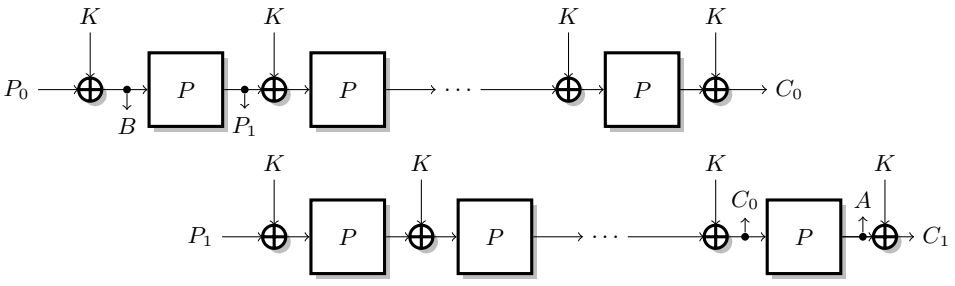


Fig. 13. Representation of a slid-pair used in a slide attack.

We define the following function:

$$f : \{0,1\} \times \{0,1\}^n \to \{0,1\}^n$$
$$b, x \quad \mapsto \begin{cases} P(E_k(x)) \oplus x & \text{if } b = 0, \\ E_k(P(x)) \oplus x & \text{if } b = 1. \end{cases}$$

The slide property shows that all x satisfy $P(E_k(x)) \oplus k = E_k(P(x \oplus k))$. This implies that f satisfies the promise of Simon's problem with $s = 1 \parallel k$:

$$f(0,x) = P(E_k(x)) \oplus x = E_k(P(x \oplus k)) \oplus k \oplus x = f(1, x \oplus k).$$

In order to apply Theorem 1, we bound $\varepsilon(f, 1\|k)$, assuming that both $E_k \circ P$ and $P \circ E_k$ are indistinguishable from random permutations. If $\varepsilon(f, 1\|k) > 1/2$, there exists (τ, t) with $(\tau, t) \notin \{(0,0), (1,k)\}$ such that: $\Pr[f(b,x) = f(b \oplus \tau, x \oplus t)] > 1/2$. Let us assume $\tau = 0$. This implies

$$\Pr[f(0,x) = f(0, x \oplus t)] > 1/2 \quad \text{or} \quad \Pr[f(1,x) = f(1, x \oplus t)] > 1/2,$$

which is equivalent to

$$\Pr[P(E_k(x)) = P(E_k(x \oplus t)) \oplus t] > 1/2 \quad \text{or} \quad \Pr[E_k(P(x)) = E_k(P(x \oplus t)) \oplus t] > 1/2.$$

In particular, there is a differential in $P \circ E_k$ or $E_k \circ P$ with probability $1/2$.
Otherwise, $\tau = 1$. This implies

$$\Pr[P(E_k(x)) \oplus x = E_k(P(x \oplus t)) \oplus x \oplus t] > 1/2$$
$$\textit{i.e.} \quad \Pr[E_k(P(x \oplus k)) \oplus k = E_k(P(x \oplus t)) \oplus t] > 1/2.$$

Again, it means there is a differential in $E_k \circ P$ with probability $1/2$.

Finally we conclude that $\varepsilon(f, 1 \parallel k) \leq 1/2$, unless $E_k \circ P$ or $P \circ E_k$ have differentials with probability $1/2$. If E_k behave as a random permutation, $E_k \circ P$ and $P \circ E_k$ also behave as random permutations, and these differential are only found with negligible probability. Therefore, we can apply Simon's algorithm, following Theorem 1, and recover k.

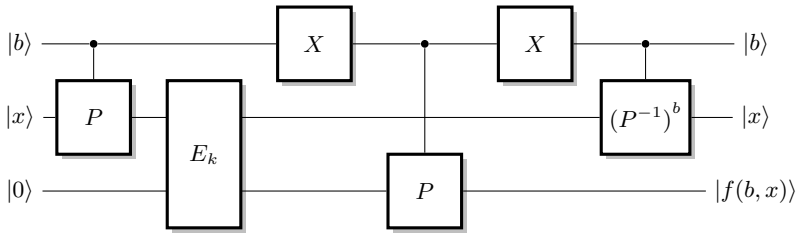


Fig. 14. Simon's function for slide attacks. The X gate is the quantum equivalent of the `NOT` gate that flips the qubit $|0\rangle$ and $|1\rangle$.

7 Conclusion

We have been able to show that symmetric cryptography is far from ready for the post quantum world. We have found exponential speed-ups on attacks on symmetric cryptosystems. In consequence, some cryptosystems that are believed to be safe in a classical world become vulnerable in a quantum world.

With the speed-up on slide attacks, we provided the first known exponential quantum speed-up of a classical attack. This attack now becomes very powerful. An interesting follow-up would be to seek other such speed-ups of generic techniques. For authenticated encryption, we have shown that many modes of operations that are believed to be solid and secure in the classical world, become completely broken in the post-quantum world. More constructions might be broken following the same ideas.

Acknowledgements

We would like to thank Thomas Santoli and Christian Schaffner for sharing an early stage manuscript of their work [40], Michele Mosca for discussions and LTCI for hospitality. This work was supported by the Commission of the European Communities through the Horizon 2020 program under project number 645622 PQCRYPTO. MK acknowledges funding through grants ANR-12-PDOC-0022-01 and ESPRC EP/N003829/1.

References

1. Abed, F., Fluhrer, S.R., Forler, C., List, E., Lucks, S., McGrew, D.A., Wenzel, J.: Pipelineable on-line encryption. In: Cid, C., Rechberger, C. (eds.) Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers. Lecture Notes in Computer Science, vol. 8540, pp. 205–223. Springer (2014)
2. Alagic, G., Broadbent, A., Fefferman, B., Gagliardoni, T., Schaffner, C., Jules, M.S.: Computational security of quantum encryption. arXiv preprint arXiv:1602.01441 (2016)
3. Anand, M.V., Targhi, E.E., Tabia, G.N., Unruh, D.: Post-quantum security of the CBC, CFB, OFB, CTR, and XTS modes of operation. In: Takagi, T. (ed.) Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9606, pp. 44–63. Springer (2016)
4. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Tischhauser, E., Yasuda, K.: Parallelizable and authenticated online ciphers. In: Sako, K., Sarkar, P. (eds.) Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I. Lecture Notes in Computer Science, vol. 8269, pp. 424–443. Springer (2013)
5. Bellare, M., Kilian, J., Rogaway, P.: The security of the cipher block chaining message authentication code. J. Comput. Syst. Sci. 61(3), 362–399 (2000)

6. Bernstein, D.J.: Introduction to post-quantum cryptography. In: Post-quantum cryptography, pp. 1–14. Springer (2009)
7. Biryukov, A., Wagner, D.: Slide attacks. In: Knudsen, L.R. (ed.) Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1636, pp. 245–259. Springer (1999)
8. Biryukov, A., Wagner, D.: Advanced slide attacks. In: Preneel, B. (ed.) Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding. Lecture Notes in Computer Science, vol. 1807, pp. 589–606. Springer (2000)
9. Black, J., Rogaway, P.: CBC macs for arbitrary-length messages: The three-key constructions. In: Bellare, M. (ed.) Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1880, pp. 197–215. Springer (2000)
10. Black, J., Rogaway, P.: A block-cipher mode of operation for parallelizable message authentication. In: Knudsen, L.R. (ed.) Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2332, pp. 384–397. Springer (2002)
11. Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., Zhandry, M.: Random oracles in a quantum world. In: Lee, D., Wang, X. (eds.) Advances in Cryptology – ASIACRYPT 2011, Lecture Notes in Computer Science, vol. 7073, pp. 41–69. Springer Berlin Heidelberg (2011)
12. Boneh, D., Zhandry, M.: Quantum-secure message authentication codes. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7881, pp. 592–608. Springer (2013)
13. Boneh, D., Zhandry, M.: Secure signatures and chosen ciphertext security in a quantum computing world. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II. Lecture Notes in Computer Science, vol. 8043, pp. 361–379. Springer (2013)
14. Brassard, G., Høyer, P., Kalach, K., Kaplan, M., Laplante, S., Salvail, L.: Merkle puzzles in a quantum world. In: Advances in Cryptology–CRYPTO 2011, pp. 391–410. Springer (2011)
15. Broadbent, A., Jeffery, S.: Quantum homomorphic encryption for circuits of low T-gate complexity. In: Advances in Cryptology–CRYPTO 2015, pp. 609–629. Springer (2015)
16. Carter, L., Wegman, M.N.: Universal classes of hash functions (extended abstract). In: Hopcroft, J.E., Friedman, E.P., Harrison, M.A. (eds.) Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA. pp. 106–112. ACM (1977)
17. Cogliani, S., Maimut, D., Naccache, D., do Canto, R.P., Reyhanitabar, R., Vaudenay, S., Vizár, D.: OMD: A compression function mode of operation for authenticated encryption. In: Joux, A., Youssef, A.M. (eds.) Selected Areas in Cryptography - SAC 2014 - 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8781, pp. 112–128. Springer (2014)

18. Daemen, J., Rijmen, V.: Probability distributions of correlation and differentials in block ciphers. J. Mathematical Cryptology 1(3), 221–242 (2007)
19. Damgård, I., Funder, J., Nielsen, J.B., Salvail, L.: Superposition attacks on cryptographic protocols. In: Padró, C. (ed.) Information Theoretic Security - 7th International Conference, ICITS 2013, Singapore, November 28-30, 2013, Proceedings. Lecture Notes in Computer Science, vol. 8317, pp. 142–161. Springer (2013)
20. Dworkin, M.: Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. NIST Special Publication 800-38B, National Institute for Standards and Technology (May 2005)
21. Even, S., Mansour, Y.: A construction of a cipher from a single pseudorandom permutation. J. Cryptology 10(3), 151–162 (1997)
22. Gagliardoni, T., Hülsing, A., Schaffner, C.: Semantic security and indistinguishability in the quantum world. arXiv preprint arXiv:1504.05255 (2015)
23. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Miller, G.L. (ed.) Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996. pp. 212–219. ACM (1996)
24. Hoang, V.T., Krovetz, T., Rogaway, P.: Robust authenticated-encryption AEZ and the problem that it solves. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 15–44. Springer (2015)
25. Iwata, T., Kurosawa, K.: OMAC: one-key CBC MAC. In: Johansson, T. (ed.) Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003, Revised Papers. Lecture Notes in Computer Science, vol. 2887, pp. 129–153. Springer (2003)
26. Iwata, T., Minematsu, K., Guo, J., Morioka, S.: CLOC: authenticated encryption for short input. In: Cid, C., Rechberger, C. (eds.) Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers. Lecture Notes in Computer Science, vol. 8540, pp. 149–167. Springer (2014)
27. Kaplan, M.: Quantum attacks against iterated block ciphers. CoRR abs/1410.1434 (2014)
28. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Quantum differential and linear cryptanalysis. CoRR abs/1510.05836 (2015)
29. Krovetz, T., Rogaway, P.: The software performance of authenticated-encryption modes. In: Joux, A. (ed.) Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers. Lecture Notes in Computer Science, vol. 6733, pp. 306–327. Springer (2011)
30. Kuwakado, H., Morii, M.: Quantum distinguisher between the 3-round Feistel cipher and the random permutation. In: Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on. pp. 2682–2685 (June 2010)
31. Kuwakado, H., Morii, M.: Security on the quantum-type Even-Mansour cipher. In: Information Theory and its Applications (ISITA), 2012 International Symposium on. pp. 312–316 (Oct 2012)
32. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. J. Cryptology 24(3), 588–613 (2011)
33. Luby, M., Rackoff, C.: How to construct pseudorandom permutations from pseudorandom functions. SIAM J. Comput. 17(2), 373–386 (1988)

34. Lydersen, L., Wiechers, C., Wittmann, C., Elser, D., Skaar, J., Makarov, V.: Hacking commercial quantum cryptography systems by tailored bright illumination. Nature photonics 4(10), 686–689 (2010)
35. McGrew, D.A., Viega, J.: The security and performance of the galois/counter mode (GCM) of operation. In: Canteaut, A., Viswanathan, K. (eds.) Progress in Cryptology - INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, December 20-22, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3348, pp. 343–355. Springer (2004)
36. Minematsu, K.: Parallelizable rate-1 authenticated encryption from pseudorandom functions. In: Nguyen, P.Q., Oswald, E. (eds.) Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8441, pp. 275–292. Springer (2014)
37. Montanaro, A., de Wolf, R.: A survey of quantum property testing. arXiv preprint arXiv:1310.2035 (2013)
38. Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: Lee, P.J. (ed.) Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3329, pp. 16–31. Springer (2004)
39. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: a block-cipher mode of operation for efficient authenticated encryption. In: Reiter, M.K., Samarati, P. (eds.) CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001. pp. 196–205. ACM (2001)
40. Santoli, T., Schaffner, C.: Using simon's algorithm to attack symmetric-key cryptographic primitives. arXiv preprint arXiv:1603.07856 (2016)
41. Sasaki, Y., Todo, Y., Aoki, K., Naito, Y., Sugawara, T., Murakami, Y., Matsui, M., Hirose, S.: Minalpher v1.1. CAESAR submission (August 2015)
42. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. 26(5), 1484–1509 (1997)
43. Simon, D.R.: On the power of quantum computation. SIAM journal on computing 26(5), 1474–1483 (1997)
44. Unruh, D.: Non-interactive zero-knowledge proofs in the quantum random oracle model. In: Eurocrypt 2015. vol. 9057, pp. 755–784. Springer (2015), preprint on IACR ePrint 2014/587
45. Xu, F., Qi, B., Lo, H.K.: Experimental demonstration of phase-remapping attack in a practical quantum key distribution system. New Journal of Physics 12(11), 113026 (2010)
46. Yuval, G.: Reinventing the travois: Encryption/mac in 30 ROM bytes. In: Biham, E. (ed.) Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings. Lecture Notes in Computer Science, vol. 1267, pp. 205–209. Springer (1997)
47. Zhandry, M.: How to construct quantum random functions. In: 53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012. pp. 679–687. IEEE Computer Society (2012)
48. Zhandry, M.: Secure identity-based encryption in the quantum random oracle model. International Journal of Quantum Information 13(04), 1550014 (2015)

49. Zhao, Y., Fung, C.H.F., Qi, B., Chen, C., Lo, H.K.: Quantum hacking: Experimental demonstration of time-shift attack against practical quantum-key-distribution systems. Physical Review A 78(4), 042333 (2008)

A Proof of Theorem 1

The proof of Theorem 1 is based of the following lemma.

Lemma 1. *For $t \in \{0,1\}^n$, consider the function $g(x) := 2^{-n} \sum_{y \in t^\perp} (-1)^{x \cdot y}$, where $t^\perp = \{y \in \{0,1\}^n \text{ s.t. } y \cdot t = 0\}$. for any x, it satisfies*

$$g(x) = \frac{1}{2}(\delta_{x,0} + \delta_{x,t}). \tag{2}$$

Proof. If $t = 0$ then $g(x) = \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} = \delta(x, 0)$, which proves the claim. From now on, assume that $t \neq 0$. It is straightforward to check that $g(0) = g(t) = \frac{1}{2}$ because all the terms of the sum are equal to 1 and there are 2^{n-1} vectors y orthogonal to t. Since $\sum_{x \in \{0,1\}^n} g(x) = 1$, it is sufficient to prove that $g(x) \geq 0$ to establish the claim in the case $t \neq 0$. For this, decompose $g(x)$ into two terms:

$$g(x) = \sum_{y \in E_0} (-1)^{x \cdot y} - \sum_{y \in E_1} (-1)^{x \cdot y} = |E_0| - |E_1|,$$

where $E_i := \{y \in \{0,1\}^n \text{ s.t. } y \cdot x = i \text{ and } y \cdot y = 0\}$ for $i = 0, 1$. Simple counting shows that:

$$|E_0| = \begin{cases} 2^{n-1} \text{ if } x = 0, \\ 2^{n-1} \text{ if } x = t, \\ 2^{n-2} \text{ otherwise.} \end{cases}$$

In particular, $|E_0| \geq |E_1|$ which implies that $g(x) \geq 0$.

We are now ready to prove Theorem 1. Each call to the main subroutine of Simon's algorithm will return a vector u_i. If cn calls are made, one obtains cn vectors $u_1, \ldots, u_{cn}$. By construction, f is such that $f(x) = f(x \oplus s)$ and consequently, the cn vectors $u_1, \ldots, u_{cn}$ are all orthogonal to s. The algorithm is successful provided one can recover the value of s unambiguously, which is the case if the cn vectors span the $(n-1)$-dimensional space orthogonal to s. (Let us note that if the space is $(n-d)$-dimensional for some constant d, one can still recover s efficiently by testing all the vectors orthogonal to the subspace.)

In other words, the failure probability p_{fail} is

$$\begin{aligned}
p_{\text{fail}} &= \Pr[\dim\big(\text{Span}(u_1,\ldots,u_n)\big) \leq n-2] \\
&\leq \Pr[\exists t \in \{0,1\}^n \setminus \{0,s\} \text{ s.t. } u_1 \cdot t = u_2 \cdot t = \cdots = u_{cn} \cdot t = 0] \\
&\leq \sum_{t \in \{0,1\}^n \setminus \{0,s\}} \Pr[u_1 \cdot t = u_2 \cdot t = \cdots = u_{cn} \cdot t = 0] \\
&\leq \sum_{t \in \{0,1\}^n \setminus \{0,s\}} \big(\Pr[u_1 \cdot t = 0]\big)^{cn} \\
&\leq \max_{t \in \{0,1\}^n \setminus \{0,s\}} \big(2\Pr[u_1 \cdot t = 0]^c\big)^n
\end{aligned}$$

where the second inequality results from the union bound and the third inequality follows from the fact that the results of the cn subroutines are independent.

In order to establish the theorem, it is now sufficient to show that $\Pr[u \cdot t = 0]$ is bounded away from 1 for all t, where u is the vector corresponding to the output of Simon's subroutine. We will prove that for all $t \in \{0,1\}^n \setminus \{0,s\}$, the following inequality holds:

$$\Pr_u[u \cdot t = 0] = \frac{1}{2}\big(1 + \Pr_x[f(x) = f(x \oplus t)]\big) \leq \frac{1}{2}(1 + \varepsilon(f,s)) \leq \frac{1}{2}(1 + p_0). \quad (3)$$

In Simon's algorithm, one can wait until the last step before measuring both registers. The final state before measurement can be decomposed as:

$$\begin{aligned}
2^{-n} \sum_{x \in \{0,1\}^n} \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} |y\rangle |f(x)\rangle = & 2^{-n} \sum_{\substack{y \in \{0,1\}^n \\ \text{s.t. } y \cdot t = 0}} \sum_{x \in \{0,1\}^n} (-1)^{x \cdot y} |y\rangle |f(x)\rangle \\
& + 2^{-n} \sum_{\substack{y \in \{0,1\}^n \\ \text{s.t. } y \cdot t = 1}} \sum_{x \in \{0,1\}^n} (-1)^{x \cdot y} |y\rangle |f(x)\rangle.
\end{aligned}$$

The probability of obtaining u such that $u \cdot t = 0$ is given by

$$\begin{aligned}
\Pr_u[u \cdot t = 0] &= \left\| 2^{-n} \sum_{\substack{y \in \{0,1\}^n \\ \text{s.t. } y \cdot t = 0}} |y\rangle \sum_{x \in \{0,1\}^n} (-1)^{x \cdot y} |f(x)\rangle \right\|^2 \\
&= 2^{-2n} \sum_{\substack{y \in \{0,1\}^n \\ \text{s.t. } y \cdot t = 0}} \sum_{x, x' \in \{0,1\}^n} (-1)^{(x \oplus x') \cdot y} \langle f(x') | f(x) \rangle \\
&= 2^{-2n} \sum_{x, x' \in \{0,1\}^n} \langle f(x') | f(x) \rangle \sum_{\substack{y \in \{0,1\}^n \\ \text{s.t. } y \cdot t = 0}} (-1)^{(x \oplus x') \cdot y} \\
&= 2^{-2n} \sum_{x, x' \in \{0,1\}^n} \langle f(x') | f(x) \rangle 2^{n-1} (\delta_{x,x'} + \delta_{x', x \oplus t}) \qquad (4) \\
&= 2^{-(n+1)} \left[\sum_{x \in \{0,1\}^n} \langle f(x) | f(x) \rangle + \sum_{x \in \{0,1\}^n} \langle f(x \oplus t) | f(x) \rangle \right] \qquad (5) \\
&= \frac{1}{2} \left[1 + \Pr_x[f(x) = f(x \oplus t)] \right] \qquad (6)
\end{aligned}$$

where we used Lemma 1 proven in the appendix in Eq. 4, and $\delta_{x,x'} = 1$ if $x = x'$ and 0 otherwise.

B Proof of Theorem 2

Let t be a fixed value and $p_t = Pr_x[f(x \oplus t = f(t)]$. Following the previous analysis, the probability that the cn vectors u_i are orthogonal to t can be written as $\Pr[u_1 \cdot t = u_2 \cdot t = \cdots = u_{cn} \cdot t = 0] = \left(\frac{1+p_t}{2}\right)^{cn}$.

In particular, we can bound the probability that Simon's algorithm returns a value t with $p_t < p_0$:

$$\Pr[p_t < p_0] = \sum_{t:\, p_t < p_0} \left(\frac{1 + p_t}{2} \right)^{cn} \leq 2^n \times \left(\frac{1 + p_0}{2} \right)^{cn}$$

Cryptanalysis of Full Sprout *

Virginie Lallemand and María Naya-Plasencia

Inria, France

Abstract. A new method for reducing the internal state size of stream cipher registers has been proposed in FSE 2015, allowing to reduce the area in hardware implementations. Along with it, an instantiated proposal of a cipher was also proposed: Sprout. In this paper, we analyze the security of Sprout, and we propose an attack that recovers the whole key more than 2^{10} times faster than exhaustive search and has very low data complexity. The attack can be seen as a divide-and-conquer evolved technique, that exploits the non-linear influence of the key bits on the update function. We have implemented the attack on a toy version of Sprout, that conserves the main properties exploited in the attack. The attack completely matches the expected complexities predicted by our theoretical cryptanalysis, which proves its validity. We believe that our attack shows that a more careful analysis should be done in order to instantiate the proposed design method.
Keywords: Stream cipher, Cryptanalysis, Lightweight, Sprout.

1 Introduction

The need of low-cost cryptosystems for several emerging applications like RFID tags and sensor networks has drawn considerable attention to the area of lightweight primitives over the last years. Indeed, those new applications have very limited resources and necessitate specific algorithms that ensure a perfect balance between security, power consumption, area size and memory needed. The strong demand from the community (for instance, [5]) and from the industry has led to the design of an enormous amount of promising such primitives, with different implementation features. Some examples are PRESENT [6], CLEFIA [26], KATAN/KTANTAN [11], LBlock [28], TWINE [27], LED [17], PRINCE [7], KLEIN [16], Trivium [10] and Grain [18].

The need for clearly recommended lightweight ciphers requires that the large number of these potential candidates be narrowed down. In this context, the need for a significant cryptanalysis effort is obvious. This has been proved by the big number of security analyses of the previous primitives that has appeared (to cite a few: [20,1,19,21,25,13,24,15]).

Stream ciphers are good candidates for lightweight applications. One of the most important limitations to their lightweight properties is the fact that to

*Partially supported by the French Agence Nationale de la Recherche through the BLOC project under Contract ANR-11-INS-011.

resist time-memory-data trade-off attacks, the size of their internal state must be at least twice the security parameter.

In FSE 2015, Armknecht et al. proposed [3,4] a new construction for stream ciphers designed to scale down the area required in hardware. The main intention of their paper is to revisit the common rule to resist against time-memory-data trade-off attacks, and reduce the minimal internal state of stream ciphers. To achieve this goal, the authors decided to involve the secret key not only in the initialization process but also in the keystream generation phase. To support this idea, an instance of this new stream cipher design is specified. This instance is based on the well studied stream cipher Grain128a [2] and as such has been named Sprout. In this paper we analyze the security of this cipher, and present an attack on the full version that allows the attacker to recover the whole 80-bit key with a time complexity of $2^{69.36}$, that is 2^{10} times faster than exhaustive search and needs very few bits of keystream. Our attack exploits an evolved divide-and-conquer idea.

In order to verify our theoretical estimation of the attack, we have implemented it on a toy version of Sprout that maintains all the properties that we exploit during the attack, and we have corroborated our predicted complexities, being able then to validate our cryptanalysis.

This paper is organised as follows: we first recall the specifications of the stream cipher Sprout in Section 2, and then describe our attack in Section 3. We provide the details of the implementation that has verified the validity of our attack in Section 4. Section 5 provides a discussion on how the attack affects the particular instantiation and the general idea.

2 Description of Sprout

In [3] the authors aim at reducing the size of the internal state used in stream ciphers while resisting to time-data-memory trade-off (TMDTO) attacks. They propose to this purpose a new design principle for stream ciphers such that the design paradigm of long states can be avoided. This is done by introducing a state update function that depends on a fixed secret key. The designers expect a minimum time effort equivalent to an exhaustive search of the key for an attacker to lead an attack, since she has to determine the key prior to realise the TMDTO.

Sprout is the concrete instantiation of this new type of stream ciphers developed in [3]. It has an IV and a key size of 80 bits. Based on Grain128a, this keystream generator is composed of two feedback shift registers of 40 bits, one linear (the LFSR) and one non-linear (the NLFSR), an initialization function and an update function, both key-dependent, and of an output function that produces the keystream (see Figure 1). The maximal keystream length that can be produced under the same IV is 2^{40}.

We first recall some notations that will be used in the following:

- t clock-cycle number
- $L^t = (l_0^t, l_1^t, \cdots, l_{39}^t)$ state of the LFSR at clock t

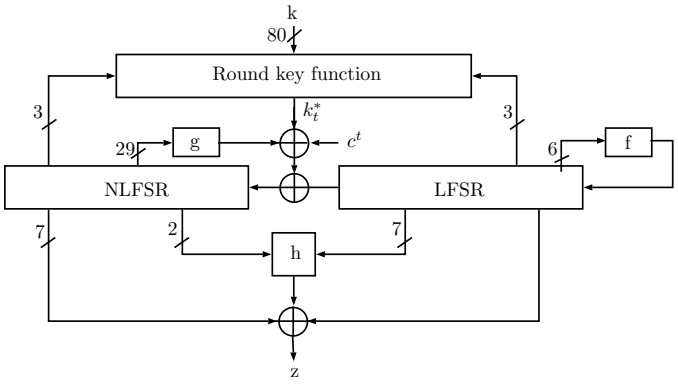


Fig. 1. Sprout KeyStream Generation

- $N^t = (n_0^t, n_1^t, \cdots, n_{39}^t)$ state of the NLFSR at clock t
- $iv = (iv_0, iv_1, \cdots, iv_{69})$ initialisation vector
- $k = (k_0, k_1, \cdots, k_{79})$ secret key
- k_t^* round key bit generated during the clock-cycle t
- z_t keystream bit generated during the clock-cycle t
- c_t round constant at clock t (generated by a counter)

A **counter** is set to determine the key bit to use at each clock and also to update the non linear register. More specifically, the counter is made up of 9 bits that count until 320 in the initialisation phase, and then count in loop from 0 to 79 in the keystream generation phase. The fourth bit (c_4^t) is used in the feedback bit computation of the NLFSR.

The 40-bit **LFSR** uses the following retroaction function, that ensures maximal period: $l_{39}^{t+1} = f(L^t) = l_0^t + l_5^t + l_{15}^t + l_{20}^t + l_{25}^t + l_{34}^t$.

The remaining state is updated as $l_i^{t+1} = l_{i+1}^t$ for i from 0 to 38.

The **NLFSR** is also 40-bit long and uses a feedback computed by:

$$\begin{aligned} n_{39}^{t+1} &= g(N^t) + k_t^* + l_0^t + c^t \\ &= k_t^* + l_0^t + c^t + n_0^t + n_{13}^t + n_{19}^t + n_{35}^t + n_{39}^t + n_2^t n_{25}^t + n_3^t n_5^t + n_7^t n_8^t + n_{14}^t n_{21}^t \\ &\quad + n_{16}^t n_{18}^t + n_{22}^t n_{24}^t + n_{26}^t n_{32}^t + n_{33}^t n_{36}^t n_{37}^t n_{38}^t + n_{10}^t n_{11}^t n_{12}^t + n_{27}^t n_{30}^t n_{31}^t, \end{aligned}$$

where k_t^* is defined as:

$$k_t^* = k_t, 0 \leq t \leq 79$$

$$k_t^* = (k_{t \bmod 80}) \times (l_4^t + l_{21}^t + l_{37}^t + n_9^t + n_{20}^t + n_{29}^t), t \geq 80$$

The remaining state is updated as $n_i^{t+1} = n_{i+1}^t$ for i from 0 to 38.

In the following, we name by $\sum l$ the sum of the LFSR bits that intervene in k_t^* when $t \geq 80$ (i.e. $\sum l \triangleq l_4^t + l_{21}^t + l_{37}^t$) and by $\sum n \triangleq n_9^t + n_{20}^t + n_{29}^t$ it NLFSR counterpart, leading to the following equivalent definition of k_t^* when $t \geq 80$:

$$k_t^* = (k_{t \bmod 80}) \times (\sum l + \sum n)$$

Update and Output Function.- The output of the stream cipher is a boolean function computed from bits of the LFSR and of the NLFSR. The nonlinear part of it is defined as:

$$h(x) = n_4^t l_6^t + l_8^t l_{10}^t + l_{32}^t l_{17}^t + l_{19}^t l_{23}^t + n_4^t n_{38}^t l_{32}^t$$

And the output bit is given by:

$$z_t = n_4^t l_6^t + l_8^t l_{10}^t + l_{32}^t l_{17}^t + l_{19}^t l_{23}^t + n_4^t n_{38}^t l_{32}^t + l_{30}^t + \sum_{j \in B} n_j^t$$

with $B = \{1, 6, 15, 17, 23, 28, 34\}$. Each time a keystream bit is generated, both feedback registers are updated by their retroaction functions.

Initialization.- The IV is loaded in the initial state in the following way: $n_i^0 = iv_i, 0 \leq i \leq 39$, $l_i = iv_{i+40}, 0 \leq i \leq 29$ and $l_i^0 = 1, 30 \leq i \leq 38$, $l_{39}^0 = 0$. The cipher is then clocked 320 times; instead of outputting the keystream bits, these bits are used as feedback in the FSRs:

$$l_{39}^{t+1} = z_t + f(L^t)$$

$$n_{39}^{t+1} = z_t + k_t^* + l^t + c_4^t + g(N^t)$$

Keystream generation.- After the 320 initialisation clocks, the keystream starts being generated according to the previously defined output function; one keystream bit per state update.

3 Key-Recovery Attack on Full Sprout

The attack described in this section and that has allowed us to attack the full version of Sprout, exploits the short sizes of the registers, the little dependency between them when generating the keystream and the non-linear influence of the keybits in the update function. We use an evolved divide-and-conquer attack, combined with a guess-and-determine technique for recovering the key bits, that resembles the analysis applied to the hash function Shabal from [22,9]. It recovers the whole key much faster than an exhaustive search and needs very little data.

Our attack is composed of three steps: in the first one, the attacker builds and arranges two independent lists of possible internal states for the LFSR and for the NLFSR at an instant $r' = 320 + r$. For now on, we will refer to time with respect to the state after initialization, being $t = 0$ the instant where the first keystream bit is output. During the second step, we merge the previous lists with the help of some bits from the keystream that will allow to perform a sieving in order to exclusively keep as candidates the pairs of states that could have generated the known keystream bits. Finally, once a reduced set of possible internal states is kept, we will recover the whole key by using some additional keystream bits. Through all the attack, we consider $r + \sharp z$ keystream bits as known $(z_0, \ldots, z_{r+\sharp z-1})$. The last $1 + \sharp z$ bits are used in the second step of the

attack, for reducing the number of state candidates. The first $r-1$ bits are used in the last step of the attack, for recovering the only one correct state and the whole key. We will use these bits in our attack, and therefore they represent the data complexity. As we show in the following, the parameters r and $\sharp z$ are the ones we adapt to optimize the attack, and in order to mount the best possible attacks, we always have $\sharp z \geq 6$ and $r \geq 1$.

We first describe some useful preliminary remarks. Next we describe the three steps of the attack, and finally we provide a summary of the full attack along with the detailed complexities of each step.

3.1 Preliminary remarks

We present in this subsection some observations on Sprout, that we use in the following sections for mounting our attack.

Let us consider the internal state of the cipher at time t. If we guessed[1] both registers at time t, how could we discard some incorrect guesses by using some known keystream bits?

Linear Register.- First of all, let us remark that the linear register state is totally independent from the rest during the keystream generation phase. Then, once its 40-bit value at time t are guessed, we can compute all of its future and past states during the keystream generation, including all its bits involved in the keystream generation.

We describe now the four sievings that can be performed in order to reduce the set of possible states with the help of the conditions imposed by the keystream bits.

Type I: Direct sieving of 2 bits.- From Section 2 we know that the keystream bit at clock cycle t is given by:

$$z_t = n_4^t l_6^t + l_8^t l_{10}^t + l_{32}^t l_{17}^t + l_{19}^t l_{23}^t + n_4^t n_{38}^t l_{32}^t + l_{30}^t + \sum_{j \in B} n_j^t$$

with $B = \{1, 6, 15, 17, 23, 28, 34\}$. We can see that 9 bits of the NLFSR intervene in the keystream bit computation, 7 linearly and 2 as part of terms of degree 2 and 3, as depicted on Figure 2 (in this figure, instant r corresponds to the generic instant t that we consider in this section). The first observation we can make is that if we know the 80 bits of the internal state at clock t, then we can directly compute the impact of the LFSR and of the NLFSR in the value of z_t and of z_{t+1} (see r and $r+1$ on Figure 2), which will potentially give us a sieving of two bits: as z_t and z_{t+1} are known, the computed values should collide with the known ones. The number of state candidates will then be reduced by a factor of 2^{-2}. For instants positioned after $t+1$, the bit n_{38}^t turns unknown so we cannot exploit the same direct sieving. In the full version of the attack, this sieving involves keystream bits z_r and z_{r+1}.

[1] which cannot be done as it contains 2^{80} possible values and therefore exceeds the exhaustive search complexity

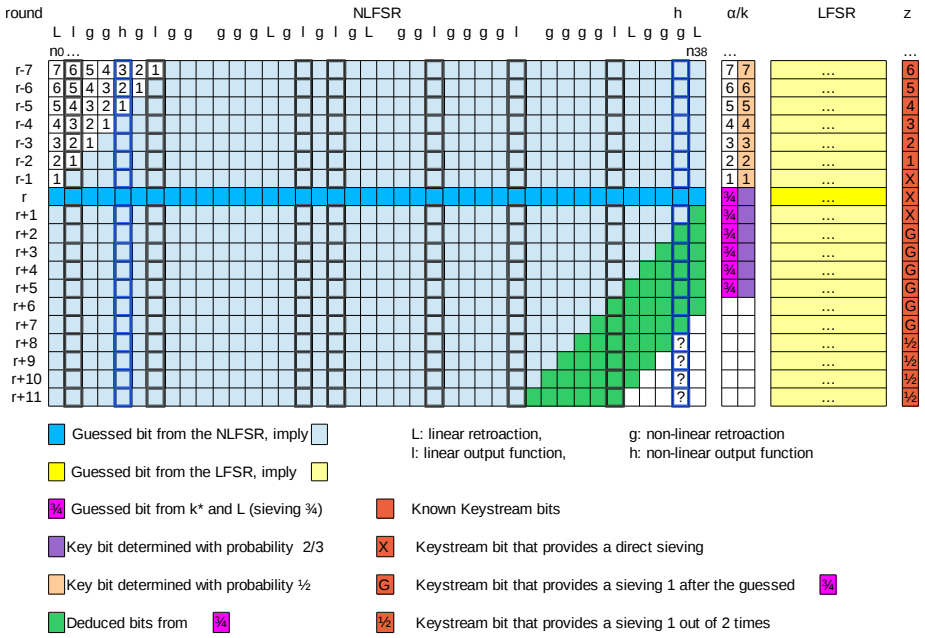


Fig. 2. Representation of the full attack. Each line represents the internal values at a certain instant, and the keystream generated at this same instant is represented in the rightmost column.

Type II: Previous round for sieving.- We consider a situation in which we have guessed a state not at instant 0, but at an instant $t > 0$. This nice idea has the advantage of allowing to additionally exploit the previously generated keystream bits to filter out the wrong states. We can therefore have for free an additional bit of sieving, provided by round $t-1$: indeed, as can be seen in Figure 2, for each possible pair of states (NLFSR, LFSR) at round $(t-1)$ we know all the bits from the NLFSR having an influence on z_{t-1}, as well as all the bits needed from the LFSR, that are also needed to compute z_{t-1}. As this keystream bit is known, we can compare it with the computed value: a match occurs with probability $1/2$, and therefore the number of possible states is reduced by a factor of 2^{-1}. In the full version of the attack, this sieving involves keystream bit z_{r-1}.

Type III: Guessing for sieving.- To obtain a better sieving, we consider one by one the keystream bits generated at time $t+i$ for $i > 1$. On average, one time out of two, n_{38}^{t+i} won't be known, as it would depend on the value of k^*_{t+i-2}.We know that, on average, k^*_{t+i-2} is null one time out of two with no additional guess. In these cases, we have an additional bit sieving, as we can directly check if z_{t+i} is the correct one. Moreover, each time the bit n_{38}^{t+i} is unknown, we can guess the

corresponding k^*_{t+i-2}, and keep as possible candidate the one that verifies the relation with z_{t+i}.In this case not only we reduce the number of possible states, but we also recover some associated key bit candidates 2 out of 3 times, as we show in details in Section 3.3. For each bit that we need to guess ($\times 2$) we obtain a sieving of 2^{-1}, which compensate. The total number of state candidates, when considering the positions that need a bit guessing and the ones that do not, is reduced by a factor of $(3/4) \approx 2^{-0.415}$ per keystream bit considered with the type III conditions. For our full attack this gives $2^{-0.415\times(\sharp z-2-4)}$, as $\sharp z$ is the number of bits considered during conditions of type I, III and IV (the one bit used during type 2 is not included in $\sharp z$). As sieving of type I always uses 2 bits, and conditions of type IV, as we see next, always use 4 bits, sieving of type III remains with $\sharp z - 2 - 4$ keystream bits. In the full version of the attack, this sieving involves keystream bits z_{t+i} for i from 2 to $(\sharp z - 5)$.

Type IV: Probabilistic sieving.- In the full version of the attack, this sieving involves keystream bits z_{t+i} for i from $\sharp z - 4$ to $(\sharp z - 1)$. Now, we do not guess bits anymore, but instead we analyse what more we can say about the states, *i.e.* whether we can reduce the amount of candidates any further. We point out that n_{38}^{t+i} only appears in one term from h.What happens if we consider also the next 4 keystream bits? What information can the next keystream bits provide? In fact, as represented in Figure 2, the next four keystream bits could be computed without any additional guesses with each considered pair of states, but for the bit n_{38}^{t+i}, that is not known. But if we have a look carefully, this bit only affects the corresponding keystream bit one time out of four. Indeed, the partial expression given by h:

$$n_4^{t+i} n_{38}^{t+i} l_{32}^{t+i}$$

is only affected by n_{38}^{t+i} for 1/4 of the values the other two related variables, n_4^{t+i} and l_{32}^{t+i}, can take. Therefore, even without knowing n_{38}^{t+i}, we can perform a sieving of one bit 3/4 of the times. As a first approximation, this can be done for four keystream bits, marked in Figure 2 with 3/4, so we will obtain an additional sieving of $4 \times 3/4 = 3$ bits, *i.e.* the number of state candidates will be additionally reduced by 2^{-3}. If we compute the exact value of this sieving, we obtain $(3/4 * 1/2 + 1/4)^4 = 2^{-2.711}$ for 4 keystream bits.

We can now start describing our attack.

3.2 Building the lists L_L and L_N

We pointed out in the previous section that guessing the whole internal state at once (80 bits) would already be as expensive as the exhaustive key search. Therefore, we start our attack by guessing separately the states of both the NLSFR and the LFSR registers at instant r. For each register we build a list, obtaining two independent lists L_L and L_N, which contain respectively the possible state bit values of the internal states of the LFSR, and respectively of the NLFSR, at a certain clock-cycle $r' = 320 + r$, i.e. r rounds after the first keystream bit is generated.

More precisely, L_L is filled with the 2^{40} possibilities for the 40 bits of the LFSR at time r (which we denoted by l_0 to l_{39}). L_N is a bigger list that contains $2^{40+\sharp z-2-4} = 2^{34+\sharp z}$ elements[2], corresponding to the 40-bit state of the NLFSR (denoted by n_0 to n_{39}), each coupled to the $2^{\sharp z-2-4}$ possible values for $\alpha_r = k_r^* + l_0^r + c_4^r$ to $\alpha_{r+\sharp z-6} = k_{r+\sharp z-6}^* + l_0^{r+\sharp z-6} + c_4^{r+\sharp z-6}$. See Figure 4 for a better description of α.

As detailed next, we also store additional bits deduced from the previous ones to speed up the attack. In L_N, we store for certain instants of time the bits n_4, n_{38}, $t_n \triangleq \sum_{j\in B} n_j$ (the linear contribution of the NLFSR to the output bit z) and $\sum n = n_9 + n_{20} + n_{29}$ (the sum of the NLFSR bits that appear in the key selection process) while in L_L it is l_6, l_{32}, $t_l \triangleq l_{30} + l_8 l_{10} + l_{19} l_{23} + l_{17} l_{32} + z_t$ and $\sum l = l_4 + l_{21} + l_{37}$. These bits are arranged as shown in Figure 3.

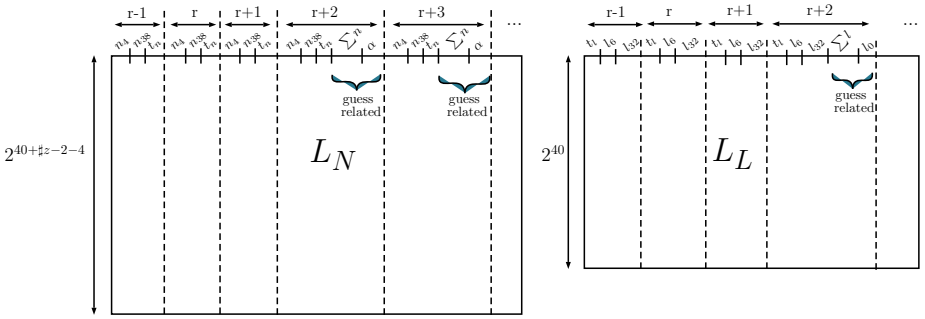


Fig. 3. Lists L_L and L_N before starting the attack. All the values used for the sorting can be computed from the original states, and the α_{r+i} in the case of L_N

3.3 Reducing the Set of Possible States

The main aim of this step is to use the precomputed lists L_L and L_N to combine them and keep only the subset of the crossproduct that corresponds to a full internal state for the registers and that could generate the keystream bits considered. It is easy to see that this problem perfectly corresponds to *merging lists* with respect to a relation, introduced in [23]. Therefore, we will use the algorithms proposed to solve it in [23,14,12] in order to efficiently find the remaining candidate pairs. Let us point out here that in the complexities we take into account for applying these algorithms, we not only take into account the candidates kept on the lists, but also the cost of sorting and comparing the lists.

[2]In the next section we describe how to reduce the state candidates step by step, so if only conditions of type I and II are considered, no guesses are needed and L_N is of size 2^{40}. When sieving conditions of type III are considered, but not of type IV, as in Table 2, the size of L_N is $2^{40+\sharp z-2}$ instead, i.e. the size of the list is $2^{40+\sharp z-2-\sharp IV}$, where $\sharp IV$ are the conditions of type IV considered.

Of course, our aim is to make the number of remaining state candidates shorter than the trivial amount of 2^{80} (the total number of possible internal states for the registers). To achieve this, we use the sieves described in section 3.1 as the relations to consider during the merging of the lists. The sieves were deduced from relations that the known keystream bits and the state bits at time r must satisfy.

For the sake of simplicity, we start by presenting an attack that only uses the sievings of type I and II. Next we will show how to also take into consideration the sieving of type III, and finally we will show how to also take into account the sieving of type IV, and therefore the 4 sievings at once for obtaining a reduced set of possible initial states.

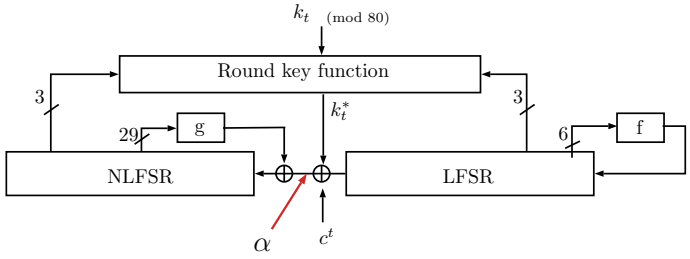


Fig. 4. Position of the Additional Guesses Stored in List L_N

Sievings of Type I and II with z_{r-1}, z_r and z_{r+1}.- Exceptionally, in this simplified version of the attack we consider $\sharp z = 2$, and t is at least one. We therefore know at least three keystream bits: z_{t-1}, z_t and z_{t+1}, that we use for reducing the size of the set of possible internal states at instant t.

We consider the previously built lists L_L and L_N both of size 2^{40} (no guesses are performed for this sievings) and are sorted as follows (see the three first columns of lists in Figure 3):

- L_L is sorted according to $t_l^t = l_{30}^t + l_8^t l_{10}^t + l_{19}^t l_{23}^t + l_{17}^t l_{32}^t + z_t$, l_6^t and l_{32}^t at instants $r-1$, r and $r+1$.
- L_N is sorted according to n_4^t, n_{38}^t and finally $t_n^t = \sum_{j \in B} n_j^t$ at time $r-1, r$ and $r+1$.

Given our new notations, we can rewrite the equation expressing z_t, as:

$$t_l^t + t_n^t + n_4^t(n_{38}^t l_{32}^t + l_6^t) = 0$$

We will use it for t from $r-1$ to $r+1$. The idea is then to use the relations implied by these three equations to deduce the possible initial state values of the LFSR and of the NLFSR in a guess and determine way.

For instance, if we first consider the situations in which the bits n_4^t and n_{38}^t are null, we know that the relation $t_l^t + t_n^t = 0$ must be satisfied so that we can only combine one eighth of L_N ($n_4^t = 0$, $n_{38}^t = 0$ and $t_n^t = 0$, or respectively $n_4 = 0$, $n_{38} = 0$ and $t_n = 1$) with one half of L_L (in which $t_l = 0$, respectively $t_l = 1$). The same way, fixing other values for n_4, n_{38} and t_n we obtain other restricted number of possibilities for the values of t_l^t, l_6^t and l_{32}^t. We reduce the total number of candidate states by 2^{-1} per keystream bit considered. When considering the equations from the three keystream bits z_{t-1}, z_t and z_{t+1}, we therefore obtain 2^{77} possible combinations instead of 2^{80}.

This is a direct application of the gradual matching algorithm from [23], and we provide a detailed description of how the algorithm works and should be implemented in Section 4.2.

Additional Sieving of Type III with $z_{r+2}, \ldots, z_{r+\sharp z-1}$.- [3]

We can easily improve the previous result by taking into account the sieving of type III presented in the previous section. List L_N will have, in this case, a size of $2^{40+\sharp z-2}$, where $\sharp z-2$ is the number of keystream bits that will be treated with sieving of type III, and therefore, the number of α_{t+i} bits that will be guessed (for i from 0 to $\sharp z - 2 - 1$). The attacker is given $(1 + \sharp z)$ bits of keystream $(z_{r-1}, \ldots, z_{r+\sharp z-1})$, and she can directly exploit z_{r-1}, z_r and z_{r+1} with sieving conditions of type I and II. Next arranging the table as showed in Figure 3 will help exploiting the conditions derived from keystream bits $z_{r+2}, \ldots, z_{r+\sharp z-1}$.

It has to be noted that the practical use of filter III slightly differs from the explanation given in the paragraph introducing it. While previously we considered making a guess only when needed, during the attack a guess is associated to every element of L_N. In 1 case out of 4 (as depicted in Table 1, the value of the guess α will be inconsistent with the given state bits, which leads to a merging filter of $2^{-0.415}$. Then, since α allows to compute the value of the retroaction bit of the NLFSR and from that the associated z, we access an additional sieve of average value 2^{-1}.

We recall that, so far (as we have not discussed yet the application of sieving conditions of type IV), the number of keystream bits treated by type III of conditions is $\sharp z - 2$. By repeating this process $\sharp z - 2$ times, we finally obtain $2^{40+\sharp z-2+40-3-(1+0.415)*(\sharp z-2)} = 2^{80-3-0.415*(\sharp z-2)}$ possible internal states. We now detail the cost of obtaining this reduced set. The process of the attack considering sievings of type I, II and III simultaneously, which is done using a gradual matching technique as described in [23], can be broadly summarized as follows and can be visualized in Table 2:

1. Consider the two precomputed lists L_N and L_L of respective sizes $2^{40+\sharp z-2}$ and 2^{40}, containing all the possibilities for the 40-bit long internal states of the NLFSR and the $\sharp z-2$ additional guesses and respectively the 40-bit long possible internal states of the LFSR.

[3] In the full attack, the last keystream bit considered here is $z_{r+\sharp z-1-4}$, as $\sharp z$ is four units bigger when considering sieving conditions of type IV

Table 1. Restrictions obtained from the additional guess, deduced from the formula of n_{39}^{t+1}

α	$\sum n$	$l_0 + c_4$	$\sum l$	information
0	0	0	0	none
			1	$k = 0$
		1	0	impossible
			1	$k = 1$
	1	0	0	$k = 0$
			1	none
		1	0	$k = 1$
			1	impossible
1	0	0	0	impossible
			1	$k = 1$
		1	0	none
			1	$k = 0$
	1	0	0	$k = 1$
			1	impossible
		1	0	$k = 0$
			1	none

2. For i from 0 to $\sharp z$, consider keystream bit z_{r+i}, and:
   (a) if $i \leq 2$, divide the current (sub)list from L_N in 2^3 sublists according to the values of n_4, n_{38} and t_n at time $r + i - 1$ and divide the current (sub)list from L_L into 2^3 sublists according to the values of t_l, l_6 and l_{32} also at time $r + i - 1$. According to the previous discussion, we know that only $2^{3+3-1} = 2^5$ combinations of sublists are possible (for sievings of type I and II). For each one of the 2^5 possible combinations, consider the next value for i.
   (b) if $i > 2$, divide further the current sublist from L_N in 2^5 sublists according to the values of the 5 bits n_4, n_{38}, t_n, $\sum n$ and $\alpha_{r+i-1-2} = (k^*_{r+i-1-2} + l_{r+i-1-2})$ (the additional guess) at time $r + i - 1$ and divide the current sublist from L_L in 2^5 sublists according to the values of the 5 bits t_l, l_6, l_{32}, $\sum l$ and l_0 at time $r + i - 1$. According to the previous discussion, we know that only $2^{5+5-1-0.415} = 2^{8.585}$ combinations of those sublists are possible. For each one of the $2^{8.585}$ possible combinations, consider the next value for i.

For a given value of $\sharp z$, the log of the complexity of recursively obtaining the reduced possibilities for the internal state by this method could be computed as the sum of the right most column of Table 2, as this represents the total number of possible sublist combinations to take into account plus the sum of this column and the log of the *relative sizes* in both remaining sublists, which are given in the last line considered, as, for each possible combination of the sublists, we have to try all the elements remaining in one list with all the elements in the other. In

Table 2.

i	L_N sublists size (log)	L_L sublists size (log)	matching pairs at this step (log)
	$40+\sharp z-2$	40	
0	$35+\sharp z$	37	5
1	$32+\sharp z$	34	5
2	$29+\sharp z$	31	5
3	$24+\sharp z$	26	8.585
4	$19+\sharp z$	21	8.585
5	$14+\sharp z$	16	8.585
6	$9+\sharp z$	11	8.585
7	$4+\sharp z$	6	8.585
8	$\sharp z$-1	1	8.585
9	$\sharp z$-6	'-4'	8.585
10	$\sharp z$-11	'-9'	8.585

the cases where the log is negative ($-h$), we only check the combinations with the other sublists when we find a non empty one, which happens with probability 2^{-h}, and this also corresponds to the described complexity.

Let us consider $\sharp z = 8$. The total time complexity[4] will be

$$2^{3*5+6*8.585} + 2^{3*5+6*8.585+8-1+1} \approx 2^{74.51}$$

If we considered for instance $\sharp z = 9$, we obtain for $i = 9$ a number of possible combinations of $2^{3*5+7*8.585} \approx 2^{75.095}$ for checking if the corresponding sublist is empty or not, and so the attack will be more expensive than when considering $\sharp z = 8$, which seems optimal (without including conditions of type IV).

To compare with exhaustive search (so to give the time complexity on encryption functions), we have to multiply $2^{74.51}$ by $\frac{8}{(320)}$, where $\frac{8}{(320)} = 2^{-5.32}$ is the term comparing our computations with one encryption, i.e. 320 initialization rounds, and we do not take into account the following 80 rounds for recovering one unique key, as with early abort techniques one or two rounds should be enough. This gives $2^{69.19}$ as time complexity, for recovering $2^{74.5}$ possible states.

We can still improve this, by using the sieving of type 4, as we show in the next section.

Additional Sieving of Type IV with $z_{r+2}, \ldots, z_{r+\sharp z-1}$.- Applying the type IV sieving is quite straight forward, as no additional guesses are needed: It just means that on average, we have an additional extra sieving of $2^{-2.71}$ per possible state found after the sievings of type I, II and III. In the end, when considering all the sievings, we recover $2^{71.8}$ possible states with a time

[4] we are not giving here the complexity yet in number of encryptions, which will reduce it when comparing with an exhaustive search

complexity determined by the previous step (applying sieving of type III which is the bottleneck) of $2^{69.19}$ encryption calls.

As previously we have determined that the optimal value for $\sharp z$ when considering sieving conditions of type I, II and III is 8, now, as we consider 4 additional keystream bits, the optimal value is $\sharp z = 8 + 4 = 12$.

The question now is: how to determine, from the $2^{71.8}$ possible states, which one is correct, and whether it is possible or not to recover the whole key. We will see how both things are possible with negligible additional cost.

3.4 Full key recovery attack: guessing a middle state

The main idea that allows us to recover the whole master key with a negligible extra complexity is considering the guessed states of the registers as not the first initial one, obtained right after initialization and generation of z_0, but instead, guessing the state after having generated r keystream bits, with $r > 0$ (for instance, values of r that we will consider are around 100). The data needs will be $r + \sharp z$ keystream bits, which is more than reasonably low (the keystream generation limit provided by the authors is 2^{40} bits). We recall here that the optimal value for $\sharp z$ is 12.

With a complexity equivalent to $2^{69.19}$ encryptions, we have recovered $2^{71.8}$ possible internal states at time r using $\sharp z + 1 = 13$ keystream bits, reducing the initial total amount by $2^{8.2}$. The question now is: how to find the only correct one, out of these $2^{71.8}$ possible states? And can we recover the 80-bit master key? We recall that, on average, we have already recovered $(\sharp z - 2 - 4) * 2/3 = 4$ keybits during the type III procedure described in section 3.3. For the sake of simplicity, and as the final complexity won't be modified (it might be slightly better for the attacker if we consider them in some cases), we will forget about these 4 keybits.

Inverting one round for free.- Using Figure 2, we will describe how to recover the whole key and the correct internal state with a negligible cost. This can be done with a technique inspired by the one for inverting the round function of the Shabal [8] hash function, proposed in [22,9]. The keystream bit from column z, marked with a 1 (at round $(r-2)$) represents z_{r-2}, and implies the value of n_1^{r-2} at this same round[5], which implies the value of n_0^{r-1}, one round later. This last value also completely determines the value of the guessed bit in round $r-1$ (α_{r-1}), which determines the value of this same round k^*_{r-1}, which, with a probability of 1/2, will determine the corresponding key bit and with probability of 1/4 won't be a valid state, corresponding to the case of $k^*_{r-1} = 1$ and $(l_4^{r-1} + l_{21}^{r-1} + l_{37}^{r-1} + n_9^{r-1} + n_{20}^{r-1} + n_{29}^{r-1}) = 0$, producing a sieving of 3/4 (we only keep 3/4 of the states on average).

[5]This result comes from the expression of z_{r-2} that linearly involves n_1^{r-2} while all the other involved terms are known

Inverting many rounds for free.- We can repeat the exact same procedure considering also the keystream bits marqued with 2 and 3 (z_{r-3} and z_{r-4} respectively). When we arrive backwards at round $(r-5)$, we are considering the keystream bit marked with 4, that is actually z_{r-5}, and the bit n_4^{r-5} needed for checking the output equations that wasn't known before, is now known as it is n_1^{r-2}, that was determined when considering the keystream bit z_{r-2}. We can therefore repeat the procedure for keystream bits 4,5,6... and so on. Indeed, in the same way, we can repeat this for as many rounds as we want, with a negligible cost (but for the constant represented by the number of rounds).

Choosing the optimal value for r.- As we have seen, going backwards r rounds (so up to the initialisation state) will determine on average $r/2$ key bits, and for each keystream bit considered we have a probability of $3/4$ of keeping the state as candidate, so we will keep a proportion of $(3/4)^{r-1}$ state candidates.

Additionally, if $r > 80$, because of the definition of k^*, the master key involved bits will start repeating[6]. For the kept state candidates, we have an additional probability of around $2/3 \times 2/3 = 2^{-2}$ of having determined the bit at one round as well as exactly 80 rounds before. The $2/3$ comes from the fact that, for having one key bit at an instant t determined we need $(l_4^t + l_{21}^t + l_{37}^t + n_9^t + n_{20}^t + n_{29}^t) = 1$, and as the case $(l_4^t + l_{21}^t + l_{37}^t + n_9^t + n_{20}^t + n_{29}^t) = 0$ with $k_t^* = 1$ has been eliminated by discarding states, we have that 2 out of the three remaining cases will determine a key bit. Therefore, when this happens, we need the bits to collide in order to keep the tested state as a candidate. This happens with an additional probability of $1/2$ per bit.

We first provide here the equations considering $\boldsymbol{r \leq 80}$. Given $2^{71.5}$ possible states obtained during the second step, the average number of states that we will keep as candidates after inverting r rounds ($\sharp s$) is $\sharp s = 2^{71.8} \times (3/4)^r$. Each one has $\sharp K = r \times 2/3$ determined key bits on average.

For $\boldsymbol{160 > r > 80}$, the average number of states that we will keep as candidates is

$$\sharp s = 2^{71.8} \times (3/4)^r \times 2^{-(r-80)\times(2/3)^2}.$$

Each one has $\sharp K = r \times 2/3 - (r-80) \times (2/3)^2$ determined key bits on average.

For any r, as we can gradually eliminate the candidate states on the fly, we do not need to compute backwards all the 100 bits but for very few of them. The complexity of testing the kept states in encryption function calls in the worst case will be

$$2^{71.8} \times \frac{1}{320} + 2^{71.8-1*0.41} \times \frac{2}{320} + \ldots + 2^{71.8-(r-1)*0.41} \times \frac{r}{320},$$

we can upper bound this complexity by $10 \times 2^{71.8} \times \frac{1}{320} \approx 2^{67.1}$, which is lower than the complexity to perform the previous step, described in Section 3.3, so won't be the bottleneck.

[6] As previously said, for the sake of simplicity we do not take into account the $\sharp z$ bits computed from r forward, and we discuss in the next section on implementation, the very little this changes in the final complexity (any way, it could only help the attacker, so the attack is as least as "good" as explained in our analysis).

As for each final kept state, we have to try all the possibilities for the remaining $80 - \sharp K$ key bits, we can conclude that the final complexity of this last part of the attack in number of encryptions is

$$\sharp s \times 2^{80-\sharp K}, \tag{1}$$

Which will be negligible most of the times (as a small increase in r means a big reduction of this complexity).

The optimal case is obtained for values of r close to 100, so we won't provide the equations when $r > 160$.

For our attack, it would seem enough to choose $r = 80$ in order to have this last step less expensive than the previous one, and therefore, in order not to increase the time complexity. We can choose $r = 100$ so that we are sure that things will behave correctly and the remaining possible key candidates can be very efficiently tested. We recall that the optimal value for $\sharp z$ was $8 + 4$, which means that the data complexity of our attack is $r + \sharp z = 112$ bits of keystream, which is very small. We have $\sharp s = 2^{21.41}$ and $\sharp K = 57.2$. The complexity of this step is therefore $2^{21.41} \times 2^{80-57.2} = 2^{44.2}$, which is much lower than the complexity of the previous steps.

3.5 Full attack summary

We consider $r = 100$ and $\sharp z = 12$. The data complexity of the attack is therefore 112 bits.

First, we have precomputed and carefully arranged the two lists L_L and L_N, of size 2^{40} and $2^{40+12-4-2} = 2^{46}$, and 2^{46} will be the memory needed to perform the attack, as all the remaining steps can be performed on the fly. Next, we merged both lists with respect to the sieving conditions of type I, II, III and IV, obtaining $2^{71.8}$ state candidates with a complexity of $2^{69.19}$ encryptions. For each candidate state, we compute some clocks backwards, in order to perform an additional sieving and to recover some key bits. This can be done with a complexity of $2^{67.1}$. The kept states and associated key bits are tested by completing the remaining key bits, and we only keep the correct one. This is done with a cost of $2^{44.2}$. We recover then the whole master key with a time complexity of $2^{69.24}$ encryptions, *i.e.* around 2^{10} times faster than an exhaustive key search. In the next section we implement the attack on a reduced version of the cipher, being able to proof the validity of our theoretical analysis, and verifying the attack.

4 Implementation and verification of the attack

To prove the validity of our attack, we experimentally test it on a shrinked cipher with similar structure and properties. More specifically, we built a small stream cipher according to the design principles used for Sprout but with a key of 22 bits and two states of 11 bits. We then implemented our attack and checked the returned complexities.

4.1 Toy cipher used

The toy cipher we built is the one represented in Figure 5. It follows the same structure as Sprout but its registers are around 4 times smaller. We have chosen the functions so that the sieving conditions behaved similarly as in our full round attack. We keep the same initialisation principle and set the number of initialisation rounds to $22 \times 4 = 88$ (in Sprout there are $80 \times 4 = 320$ initialisation rounds).

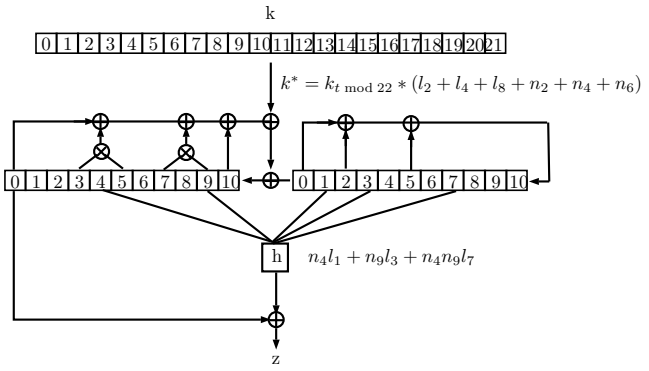


Fig. 5. Toy Cipher

4.2 Algorithm implemented

Steps 1 and 2 of the attack.-

1. Ask for $r + \#z = r + 3$ keystream bits generated from time $t = 0$ to $t = r + 3 - 1$, that we denote by $z^0, z^1, \cdots z^{r+2}$
2. Build a list L_L of size 2^{11} containing all the possible values for the 11 bits of the linear register at time $t = r$, sorted according to:
   - l_1^r, l_3^r and l_7^r at time $t = r$,
   - l_1^{r+1}, l_3^{r+1} and l_7^{r+1} at time $t = r + 1$,
   - l_3^{r+2} and l_7^{r+2} at time $t = r + 2$ and finally
   - l_0^r, $l_2^r + l_4^r + l_8^r$ at time $t = r$
3. Build a list L_N of size $2^{11+1} = 2^{12}$ that contains all the possible state values of the non-linear register at time $t = r$ plus the value of an additional guess and sort it according to:
   - $n_0^r + z^r$, n_4^r and n_9^r at time $t = r$,
   - $n_0^{r+1} + z^{r+1}$, n_4^{r+1} and n_9^{r+1} at time $t = r + 1$,
   - $n_0^{r+2} + z^{r+2}$, n_4^{r+2} and n_9^{r+2} at time $t = r + 2$ and finally
   - α^r (the guessed bit) at time $t = r$
4. Create a new list M containing the possible value of L_L and L_N together:
   (a) Consider the states of L_L and L_N for which the first indexes (l_1^r, l_3^r and l_7^r in L_L and $n_0^r + z^r$, n_4^r and n_9^r in L_N) verify the equation given by the keystream bit at time $t = r$:

$$z^r = n_4^r l_1^r + n_9^r l_3^r + n_4^r n_9^r l_7^r + n_0^r$$

i. Apply a second filter given by the second indexes (l_1^{r+1}, l_3^{r+1} and l_7^{r+1} in L and $n_0^{r+1} + z^{r+1}$, n_4^{r+1} and n_9^{r+1} in G) by checking if the equation given by the keystream bit at time $t = r + 1$ holds:

$$z^{r+1} = n_4^{r+1} l_1^{r+1} + n_9^{r+1} l_3^{r+1} + n_4^{r+1} n_9^{r+1} l_7^{r+1} + n_0^{r+1}$$

A. Similarly, apply a sieving according to the third indexes. Remark here that l_1 at time $t = r + 2$ is equal to the already fixed bit l_3 at time $t = r$. Finally, use the additional information deduced from α at time $t = r$ that must verify

$$\alpha^r = k^r \cdot (l_2^r + l_4^r + l_8^r + n_2^r + n_4^r + n_6^r)$$

so that it implies a contradiction if $l_2^r + l_4^r + l_8^r = n_2^r + n_4^r + n_6^r$ and $\alpha^r \neq l_0$ at the same time.

As discussed in Section 3.3, the resulting filter on the cardinal product of the list is of $2^{-1-1-1-0.415}$ so $2^{23-3.415} = 2^{19.585}$ possible states remain at this point.

Step 3 of the attack.-

1. For each of the $2^{19.585}$ possible states at time $t = r$, create a vector of 22 bits $\tilde{K}$ for the possible value of the key associated to it:
   (a) For time $t = r - 1$ to $t = 0$:
      i. Deduce the values of n_i^t, $i = 1 \cdots 10$ and of l_i^t, $i = 1 \cdots 10$ from the state at time $t + 1$
      ii. Compute the value of n_0^t given by the keystream bit equation as:
      $$n_0^t = z^t + n_4^t l_1^t + n_9^t l_3^t + n_4^t n_9^t l_7^t$$
      and of l_0^t given by the LFSR retroaction equation as:
      $$l_0^t = l_2^t + l_5^t + l_{10}^{t+1}$$
      and deduce from it the value of
      $$k^{*t} = n_0^t + n_3^t n_5^t + n_7^t n_9^t + n_{10}^t + l_0 + n_{10}^{t+1}$$
      (given by the NLFSR retroaction equation)
      iii. Compute the value of $l_2^t + l_4^t + l_8^t + n_2^t + n_4^t + n_6^t$ and combine it with the value of k^{*t} obtained in the previous step:
         A. If $l_2^t + l_4^t + l_8^t + n_2^t + n_4^t + n_6^t = 0$ and $k^{*t} = 1$, there is a contradiction so discard the state and try another one by going back to Step 1.
         B. If $l_2^t + l_4^t + l_8^t + n_2^t + n_4^t + n_6^t = 1$ and $k^{*t} = 0$ check if the bit has already been set in $\tilde{K}$. If no, set it to 0. Else, if there is a contradiction, discard the state and try another one by going back to Step 1.
         C. If $l_2^t + l_4^t + l_8^t + n_2^t + n_4^t + n_6^t = 1$ and $k^{*t} = 1$ check if the bit has already been set in $\tilde{K}$. If no, set it to 1. Else, if there is a contradiction, discard the state and try another one by going back to Step 1.

4.3 Results

The previous algorithm has been implemented and tested for various values of r. At the end of step 2 we recovered indeed $2^{19.5}$ state candidates. In all the cases,

the pair formed by the correct internal state and the partial right key were included amongst the candidates at the end of step 3. The results are displayed in Table 3, together with the values predicted by theory. We recall here that the expected number of states at the end of the key recovery is given by the formula in Section 3.4 which in this case can be simplified by:

$$2^{19.5} \times (3/4)^r = 2^{19.5-0.415r} \quad \text{when} \quad r < |k| \text{ and by}$$
$$2^{19.5} \times (3/4)^r \times 2^{-(r-|k|)\times(2/3)^2} = 2^{29.35-0.859r} \quad \text{when} \quad r \geq |k|.$$

In the same way, we expect the following amount of bits to be determined:

$$r \times (2/3) \quad \text{when} \quad r < |k| \text{ and}$$
$$r \times (2/3) - (r - |k|) \times (2/3)^2 \quad \text{when} \quad r \geq |k|.$$

This leads to the comparison given in Table 3 in which we can remark that theory and practice meet quite well.

Note that given the implementation results, a sensible choice would be to consider a value of r around 26. Indeed, $r = 26$ means that the attacker has to consider all the $2^{7.32}$ states at the end of the key recovery part and for each of them has to exhaust on average the 6.67 unknown bits, leading to an additional complexity of $2^{13.99}$. This number has to be compared to the time complexity of the previous operation. The time complexity for recovering the $2^{19.585}$ candidates at the end of step 2 is the bottleneck of the time complexity. According to Section 3.3, this term can be approximated by $2^{19.585} \times \frac{3}{88} \simeq 2^{14.71}$ encryptions. So recovering the full key is of negligible complexity in comparison, and $r = 26$ leads to an attack of time complexity smaller than 2^{15} encryptions, coinciding with our theoretical complexity.

Table 3. Experimental Results Obtained on Average on 300 Random States and Keys

r	20	21	22	23	24	25	26	27	28	29	30
log of number of states remaining at the end of the key recovery	11.28	10.85	10.47	9.68	8.95	8.01	7.32	6.63	5.75	5.17	4.42
theory	11.3	10.9	10.5	9.6	8.8	7.9	7.0	6.2	5.3	4.4	3.6
unknown bits	8.68	8.02	7.30	7.12	6.96	6.77	6.67	6.32	6.29	6.03	5.94
theory	8.7	8.0	7.3	7.1	6.9	6.7	6.4	6.2	6.0	5.8	5.6

5 Conclusion

In this paper we present a key-recovery attack on the stream cipher Sprout, proposed at FSE 2015, that allows to recover the whole key more than 2^{10} times

faster than exhaustive search. We have implemented our attack on a toy version of the cipher. This implemented attack behaves as predicted, and, therefore, we have been able to verify the correctness of our approach. Our attack exploits the small size of the registers and the non-linear influence of the key in the update function. It shows a security issue on Sprout and suggests that a more careful analysis should be done in order to instantiate the proposed design method.

An interesting direction to look at for repairing this weakness would be to consider the key influence on the update function as linear.

References

1. Abdelraheem, M.A., Blondeau, C., Naya-Plasencia, M., Videau, M., Zenner, E.: Cryptanalysis of ARMADILLO2. In: ASIACRYPT. LNCS, vol. 7073, pp. 308–326. Springer (2011)
2. Ågren, M., Hell, M., Johansson, T., Meier, W.: Grain-128a: a New Version of Grain-128 with Optional Authentication. IJWMC 5(1), 48–59 (2011)
3. Armknecht, F., Mikhalev, V.: On Lightweight Stream Ciphers with Shorter Internal States. In: FSE 2015. LNCS, Springer (2015), to appear.
4. Armknecht, F., Mikhalev, V.: On Lightweight Stream Ciphers with Shorter Internal States. Cryptology ePrint Archive, Report 2015/131 (2015), `http://eprint.iacr.org/2015/131`
5. Bogdanov, A., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y.: Hash Functions and RFID Tags: Mind the Gap. In: Cryptographic Hardware and Embedded Systems - CHES 2008. LNCS, vol. 5154, pp. 283–299. Springer (2008)
6. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Cryptographic Hardware and Embedded Systems - CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer (2007)
7. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçin, T.: PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In: Advances in Cryptology - ASIACRYPT 2012. LNCS, vol. 7658, pp. 208–225. Springer (2012)
8. Bresson, E., Canteaut, A., Chevallier-Mames, B., Clavier, C., Fuhr, T., Gouget, A., Icart, T., J.Misarsky, Naya-Plasencia, M., Paillier, P., Pornin, T., Reinhard, J., Thuillet, C., Videau, M.: Shabal. In: The first SHA-3 Candidate Conference. Leuven, Belgium (2009)
9. Bresson, E., Canteaut, A., Chevallier-Mames, B., Clavier, C., Fuhr, T., Gouget, A., Icart, T., Misarsky, J.F., Naya-Plasencia, M., Paillier, P., Pornin, T., Reinhard, J.R., Thuillet, C., Videau, M.: Indifferentiability with Distinguishers: Why Shabal Does Not Require Ideal Ciphers. Cryptology ePrint Archive, Report 2009/199 (2009), `http://eprint.iacr.org/2009/199`
10. Cannière, C.D.: Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In: Information Security, 9th International Conference, ISC 2006. LNCS, vol. 4176, pp. 171–186. Springer (2006)
11. Cannière, C.D., Dunkelman, O., Knezevic, M.: KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: Cryptographic

Hardware and Embedded Systems - CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer (2009)
12. Canteaut, A., Naya-Plasencia, M., Vayssière, B.: Sieve-in-the-Middle: Improved MITM Techniques. In: CRYPTO 2013 (I). LNCS, vol. 8042, pp. 222–240. Springer (2013)
13. Collard, B., Standaert, F.X.: A Statistical Saturation Attack against the Block Cipher PRESENT. In: Topics in Cryptology - CT-RSA 2009. pp. 195–210. LNCS 5473, Springer (2009)
14. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Efficient Dissection of Composite Problems, with Applications to Cryptanalysis, Knapsacks, and Combinatorial Search Problems. In: Advances in Cryptology - CRYPTO 2012. LNCS, vol. 7417, pp. 719–740. Springer (2012)
15. Dinur, I., Shamir, A.: Cube Attacks on Tweakable Black Box Polynomials. In: Advances in Cryptology - EUROCRYPT 2009. LNCS, vol. 5479, pp. 278–299. Springer (2009)
16. Gong, Z., Nikova, S., Law, Y.W.: KLEIN: A New Family of Lightweight Block Ciphers. In: RFID. Security and Privacy, RFIDSec 2011. LNCS, vol. 7055, pp. 1–18. Springer (2011)
17. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The LED Block Cipher. In: Cryptographic Hardware and Embedded Systems - CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer (2011)
18. Hell, M., Johansson, T., Meier, W.: Grain: a Stream Cipher for Constrained Environments. IJWMC 2(1), 86–93 (2007)
19. Lallemand, V., Naya-Plasencia, M.: Cryptanalysis of KLEIN. In: Fast Software Encryption, FSE 2014. LNCS, vol. 8540, pp. 451–470. Springer (2014)
20. Leander, G., Abdelraheem, M.A., AlKhzaimi, H., Zenner, E.: A Cryptanalysis of PRINTcipher: The Invariant Subspace Attack. In: Advances in Cryptology - CRYPTO 2011. LNCS, vol. 6841, pp. 206–221. Springer (2011)
21. Mendel, F., Rijmen, V., Toz, D., Varici, K.: Differential Analysis of the LED Block Cipher. In: Advances in Cryptology - ASIACRYPT 2012. LNCS, vol. 7658, pp. 190–207. Springer (2012)
22. Naya-Plasencia, M.: Chiffrements à flot et fonctions de hachage : conception et cryptanalyse. Thése, INRIA Paris-Rocquencourt, Project SECRET et Université Pierre et Marie Curie, France (2009)
23. Naya-Plasencia, M.: How to Improve Rebound Attacks. In: CRYPTO 2011. LNCS, vol. 6841, pp. 188–205. Springer (2011)
24. Naya-Plasencia, M., Peyrin, T.: Practical cryptanalysis of ARMADILLO2. In: Fast Software Encryption, FSE 2012. LNCS, vol. 7549, pp. 146–162. Springer (2012)
25. Nikolic, I., Wang, L., Wu, S.: Cryptanalysis of Round-Reduced LED. In: Fast Software Encryption, FSE 2013. LNCS, vol. 8424, pp. 112–129. Springer (2013)
26. Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-Bit Blockcipher CLEFIA (Extended Abstract). In: Fast Software Encryption, FSE 2007. LNCS, vol. 4593, pp. 181–195. Springer (2007)
27. Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: TWINE : A Lightweight Block Cipher for Multiple Platforms. In: In Selected Areas in Cryptography- SAC 2012. LNCS, vol. 7707, pp. 339–354. Springer (2012)
28. Wu, W., Zhang, L.: LBlock: A Lightweight Block Cipher. In: Applied Cryptography and Network Security, ACNS 2011. LNCS, vol. 6715, pp. 327–344 (2011)

Cryptanalysis of ARMADILLO2 *

Mohamed Ahmed Abdelraheem[1], Céline Blondeau[2], María Naya-Plasencia[3] †, Marion Videau[4,5] ‡, and Erik Zenner[6] §

[1] Technical University of Denmark, Department of Mathematics, Denmark
[2] INRIA, project-team SECRET, France
[3] FHNW, Windisch, Switzerland and University of Versailles, France
[4] Agence nationale de la scurit des systmes d'information, France
[5] Université Henri Poincaré-Nancy 1 / LORIA, France
[6] University of Applied Sciences Offenburg, Germany

Abstract. ARMADILLO2 is the recommended variant of a multi-purpose cryptographic primitive dedicated to hardware which has been proposed by Badel et al. in [1]. In this paper, we describe a meet-in-the-middle technique relying on the parallel matching algorithm that allows us to invert the ARMADILLO2 function. This makes it possible to perform a key recovery attack when used as a FIL-MAC. A variant of this attack can also be applied to the stream cipher derived from the PRNG mode. Finally we propose a (second) preimage attack when used as a hash function. We have validated our attacks by implementing cryptanalysis on scaled variants. The experimental results match the theoretical complexities.
In addition to these attacks, we present a generalization of the parallel matching algorithm, which can be applied in a broader context than attacking ARMADILLO2.

Keywords: ARMADILLO2, meet-in-the-middle, key recovery attack, preimage attack, parallel matching algorithm.

1 Introduction

ARMADILLO is a multi-purpose cryptographic primitive dedicated to hardware which was proposed by Badel et al. in [1]. Two variants were presented: ARMADILLO and ARMADILLO2, the latter being the recommended version. In the following, the first variant will be denoted ARMADILLO1 to distinguish it from ARMADILLO2. Both variants comprise several versions, each

*This work was partially supported by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II

†Supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center of the Swiss National Science Foundation under grant number 5005-67322 and by the French Agence Nationale de la Recherche through the SAPHIR2 project under Contract ANR-08-VERS-014

‡Partially supported by the French Agence Nationale de la Recherche under Contract ANR-06-SETI-013-RAPIDE

§This work was produced while at the Technical University of Denmark

one associated to a different set of parameters and to a different security level. For both primitives, several applications are proposed: fixed input-length MAC (FIL-MAC), pseudo-random number generator/pseudo-random function (PRNG/PRF), and hash function. In [6], authors present a polynomial attack on ARMADILLO1. Even if the design of ARMADILLO2 is similar to the design of the first version, authors of [6] claim that this attack can not be applied on ARMADILLO2.

The ARMADILLO family uses a parameterized internal permutation as a building block. This internal permutation is based on two bitwise permutations σ_0 and σ_1. In [1], these permutations are not specified, but some of the properties that they must satisfy are given.

In this paper we provide the first cryptanalysis of ARMADILLO2, the recommended variant. As the bitwise permutations σ_0 and σ_1 are not specified, we have performed our analysis under the reasonable assumption that they behave like random permutations. As a consequence, the results of this paper are independent of the choice for σ_0 and σ_1.

To perform our attack, we use a meet-in-the-middle approach and an evolved variant of the parallel matching algorithm introduced in [2] and generalized in [5, 4]. Our method enables us to invert the building block of ARMADILLO2 for a chosen value of the public part of the input, when a part of the output is known. We can use this step to build key recovery attacks faster than exhaustive search on all versions of ARMADILLO2 used in the FIL-MAC application mode. Besides, we propose several trade-offs for the time and memory needed for these attacks. We also adapt the attack to recover the key when ARMADILLO2 is used as a stream cipher in the PRNG application mode. We further show how to build (second) preimage attacks faster than exhaustive search when using the hashing mode, and propose again several time-memory trade-offs. We have implemented the attacks on a scaled version of ARMADILLO2, and the experimental results confirm the theoretical predictions.

Organization of the paper. We briefly describe ARMADILLO2 in Section 2. In Section 3 we detail our technique for inverting its building block and we explain how to extend the parallel matching algorithm to the case of ARMADILLO2. In Section 4, we explain how to apply this technique to build a key recovery attack on the FIL-MAC application mode. We briefly show how to adapt this attack to the stream cipher scenario in Section 4.2. The (second) preimage attack on the hashing mode is presented in Section 5. In Section 6 we present the experimental results of the verification that we have done on a scaled version of the algorithm. Finally, in Section 7, we propose a general form of the parallel matching algorithm derived from our attacks which can hopefully be used in more general contexts.

2 Description of ARMADILLO2

The core of ARMADILLO is based on the so-called *data-dependent bit transpositions* [3]. We recall the description of ARMADILLO2 given in [1] using the same notations.

2.1 Description

Let C be an initial vector of size c and U be a message block of size m. The size of the register $(C\|U)$ is $k = c + m$. The ARMADILLO2 function transforms the vector

(C, U) into (V_c, V_t) as described in Figure 1:

$$\begin{array}{rccl} \text{ARMADILLO2} & : & \mathbb{F}_2^c \times \mathbb{F}_2^m & \to \mathbb{F}_2^c \times \mathbb{F}_2^m \\ & & (C, U) & \mapsto (V_c, V_t) = \text{ARMADILLO2}(C, U). \end{array}$$

The function ARMADILLO2 relies on an internal bitwise parameterized permutation denoted by Q which is defined by a parameter A of size a and is applied to a vector B of size k:

$$\begin{array}{rccl} Q & : & \mathbb{F}_2^a \times \mathbb{F}_2^k & \to \mathbb{F}_2^k \\ & & (A, B) & \mapsto Q(A, B) = Q_A(B) \end{array}$$

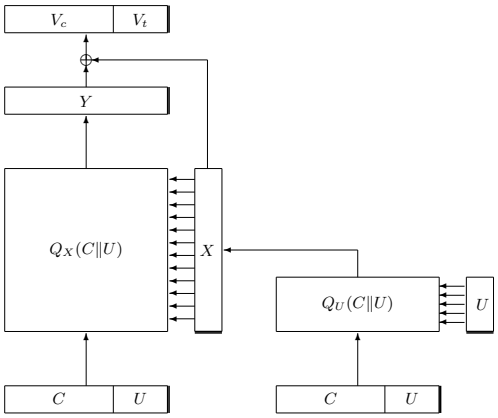


Fig. 1. ARMADILLO2.

Let σ_0 and σ_1 be two fixed bitwise permutations of size k. In [1], the permutations are not defined but some criteria they should fulfil are given. As the attacks presented in this paper are valid for any bitwise permutations, we do not describe these properties. We just stress that in the following, when computing the complexities we assume that these permutations behave like random ones. We denote by γ a constant of size k defined by alternating 0's and 1's: $\gamma = 1010 \cdots 10$.
Using these notations, we can define Q which is used twice in the ARMADILLO2 function. Let A be a parameter and B be the internal state, the parameterized permutation Q (that we denote by Q_A when indicating the parameter is necessary) consists in $a = |A|$ simple steps. The i-th step of Q (reading A from its least significant bit to its most significant one) is defined by:

- an elementary bitwise permutation: $B \leftarrow \sigma_{A_i}(B)$, that is:
  - if the i-th bit of A equals 0 we apply σ_0 to the current state,
  - otherwise (if the i-th bit of A equals 1) we apply σ_1 to the current state,
- a constant addition (bitwise XOR) of γ: $B \leftarrow B \oplus \gamma$.

Using the definition of the permutation Q, we can describe the function ARMADILLO2. Let (C, U) be the input, then ARMADILLO2(C, U) is defined by:

- first compute $X \leftarrow Q_U(C\|U)$
- then compute $Y \leftarrow Q_X(C\|U)$
- finally compute $(V_c\|V_t) \leftarrow Y \oplus X$, the output is (V_c, V_t).

Actually c and m can take different values depending on the required security level. A summary of the sets of parameters for the different versions (A, B, C, D or E) proposed in [1] is given in Table 1.

Version	k	c	m
A	128	80	48
B	192	128	64
C	240	160	80
D	288	192	96
E	384	256	128

Table 1. Sets of parameters for the different versions of ARMADILLO2.

2.2 A Multi-Purpose Cryptographic Primitive

The general-purpose cryptographic function ARMADILLO2 can be used for three types of applications: FIL-MAC, hashing, and PRNG/PRF.

ARMADILLO*2 in FIL-MAC mode.* The secret key is C and the challenge, considered known by the attacker, is U. The response is V_t.

ARMADILLO*2 in hashing mode.* It uses a strengthened Merkle-Damgård construction, where V_c is the chaining value or the hash digest, and U is the message block.

ARMADILLO*2 in PRNG/PRF mode.* The output sequence is obtained by taking the first t bits of (V_c, V_t) after at least r iterations. For ARMADILLO2 the proposed values are $r = 1$ and $t = k$ (see [1, Sec. 6]). When used as a stream cipher, the secret key is C. The keystream is composed of k-bit frames indexed by U which is a public value.

3 Inverting the ARMADILLO2 Function

In [1] a sketch of a meet-in-the-middle (MITM) attack on ARMADILLO1, the first variant of the primitive, is given by the authors to prove *lower bounds* for the complexity and justify the choice of parameters. However, they do not develop further their analysis.

In this section we describe how to invert the ARMADILLO2 function when a part of the output (V_c, V_t) is known and U is chosen in the input $(C\|U)$. Inverting means that we recover C. The method we present can be performed for any arbitrary bitwise permutations σ_0 and σ_1. To conduct our analysis we suppose that they behave like random ones. Indeed, if the permutations σ_0 and σ_1 were not behaving like random ones, one could exploit their distributions to reduce the complexities of the attacks presented in this paper. Therefore, we are considering the worst case scenario for an attacker.

First, we describe the meet-in-the-middle technique we use. It provides two lists of partial states in the middle of the main permutation Q_X. To determine a list of possible values for C, we need to select a subset of the cartesian product of these two lists containing consistent couples of partial states. To build such a subset efficiently, we explain how to use an adaptation of the *parallel matching algorithm* presented in [2, 5]. Then we present and apply the adapted algorithm and compute its time and memory complexities.
All cryptanalysis, we present, on the different applications of ARMADILLO2 relies on the technique for recovering C presented in this section.

3.1 The Meet-in-the-Middle Technique

Whatever mode ARMADILLO2 is embedded in, we use the following facts:

- We can choose the m-bit vector U, in the input vector $(C\|U)$.
- We know part of the output vector $(V_c\|V_t)$: the m-bit vector V_t in the FIL-MAC, the $(c+m)$-bit vector $(V_c\|V_t)$ in the PRNG/PRF and the c-bit vector V_c in the hash function.

We deal with two permutations: the pre-processing Q_U which is known as U is known and the main permutation Q_X which is unknown, and we exploit the three following equations:

- The permutation Q_U used in the pre-processing $X = Q_U(C\|U)$ is known. This implies that all the known bits in the input of the permutation can be traced to their corresponding positions in X. For instance, there are m coordinates of X whose values are determined by choosing U.
- The output of the main permutation $Y = (V_c\|V_t) \oplus X$ implies we know some bits of Y. The amount of known bits of Y is denoted by y and is depending on the mode we are focusing on through $(V_c\|V_t)$.
- In the sequel, we divide X in two parts: $X = (X_{\text{out}}\|X_{\text{in}})$. Then, the main permutation $Y = Q_X(C\|U)$ can be divided in two parts: $Q_{X_{\text{in}}}$ and $Q_{X_{\text{out}}}$ separated by a division line we call the *middle*, hence we perform the meet-in-the-middle technique between $Q_{X_{\text{in}}}$ and $Q^{-1}_{X_{\text{out}}}$.

As $(X_{\text{out}}\|X_{\text{in}}) = Q_U(C\|U)$, we denote by m_{in} (resp. m_{out}) the number of bits of U that are in X_{in} (resp. X_{out}). We have $m_{\text{out}} + m_{\text{in}} = m$. We denote by ℓ_{in} (resp. ℓ_{out}) the number of bits coming from C in X_{in} (resp. X_{out}). We have $\ell_{\text{out}} + \ell_{\text{in}} = c$. The meet-in-the-middle attack is done by guessing the ℓ_{in} unknown bits of X_{in} and the ℓ_{out} unknown bits of X_{out} independently.

First, consider the forward direction. We can trace the ℓ_{in} unknown bits of X_{in} back to C with Q_U^{-1}. Next, for each possible guess of X_{in}, we can trace the corresponding ℓ_{in} bits from C plus the m bits from U to their positions in the middle by computing $Q_{X_{\text{in}}}(C\|U)$. Then consider the backward direction, we can trace the y known bits of Y back to the middle for each possible guess of X_{out}, that is computing $Q^{-1}_{X_{\text{out}}}(Y)$. This way we can obtain two lists $\mathcal{L}_{\text{in}}$ and $\mathcal{L}_{\text{out}}$, of size $2^{\ell_{\text{in}}}$ and $2^{\ell_{\text{out}}}$ respectively, of elements that represent partially known states in the middle of Q_X.

To describe our meet-in-the-middle attack we represent the partial states in the middle of Q_X as ternary vectors with coordinate values from $\{0, 1, -\}$, where $-$ denotes a coordinate (or cell) whose value is unknown. We say that a cell is *active* if it contains 0 or 1 and *inactive* otherwise. The weight of a vector V, denoted by $\text{wt}(V)$, is the number of its active cells. Two partial states are a match if their colliding active cells have the same values.

The list $\mathcal{L}_{\text{in}}$ contains elements $Q_{X_{\text{in}}}(C\|U)$ whose weight is $x = \ell_{\text{in}} + m$. The list $\mathcal{L}_{\text{out}}$ contains elements $Q^{-1}_{X_{\text{out}}}(Y)$ whose weight is y. When taking one element from each list, the probability of finding a match will then depend on the number of collisions of active cells between these two elements.

Consider a vector A in $\{0, 1, -\}^k$ with weight a. We denote by $P_{[k,a,b]}(i)$ the probability over all the vectors $B \in \{0, 1, -\}^k$ with weight b of having i active cells at the same positions in A and B. This event corresponds to the situation where there are i active cells of B among the a active positions in A and the remaining $(b - i)$ active

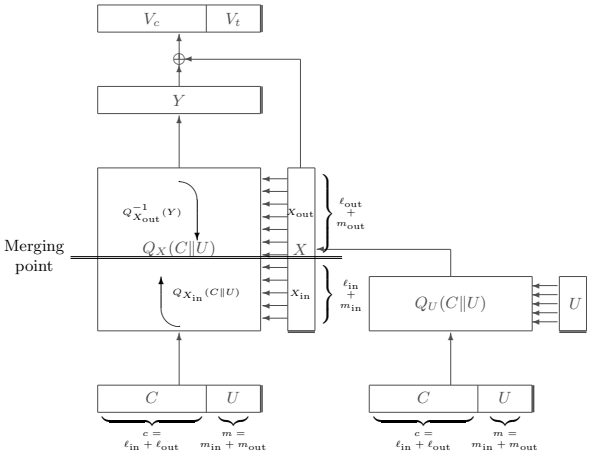


Fig. 2. Overview of the inversion of the ARMADILLO2 core function.

cells of B lie in the $(k-a)$ inactive positions in A. As the number of vectors of length k and weight b is $\binom{k}{b}$, we have:

$$P_{[k,a,b]}(i) = \frac{\binom{a}{i}\binom{k-a}{b-i}}{\binom{k}{b}} = \frac{\binom{b}{i}\binom{k-b}{a-i}}{\binom{k}{a}}.$$

Taking into account the probability of having active cells at the same positions in a pair of elements from $(\mathcal{L}_{\text{in}}, \mathcal{L}_{\text{out}})$ and the probability that these active cells do have the same value, we can compute the *expected probability* of finding a match for a pair of elements, that we will denote $2^{-N_{\text{coll}}}$. We have:

$$2^{-N_{\text{coll}}} = \sum_{i=0}^{y} 2^{-i} P_{[k,x,y]}(i).$$

This means that there will be a possible match with a probability of $2^{-N_{\text{coll}}}$. In total we will find $2^{\ell_{\text{in}}+\ell_{\text{out}}-N_{\text{coll}}}$ pairs of elements that pass this test. Each pair of elements defines a whole C value. Next, we just have to check which of these values is the correct one.

The big question now is that of the cost of checking which elements of the two lists $\mathcal{L}_{\text{in}}$ and $\mathcal{L}_{\text{out}}$ pass the test. The ternary alphabet of the elements and the changing positions of the active cells make it impossible to apply the approach of traditional MITM attacks — having an ordered list $\mathcal{L}_{\text{in}}$ and checking for each element in the list $\mathcal{L}_{\text{out}}$ if a match exists with cost 1 per element. Even more, a priori, for each element in $\mathcal{L}_{\text{in}}$ we would have to try if it matches each of the elements from $\mathcal{L}_{\text{out}}$ independently, which would yield the complexity of exhaustive search.

For solving this problem we adapt the algorithm described in [5, Sec. 2.3] as *parallel matching* to the case of ARMADILLO2. A generalized version of the algorithm is exposed in Section 7 with detailed complexity calculations and the link to our application case.

3.2 ARMADILLO2 Matching Problem: Matching Non-Random Elements

Recently, new algorithms have been proposed in [5] to solve the problem of *merging* several lists of big sizes with respect to a given relation t that can be verified by tuples

of elements. These new algorithms take advantage of the special structures that can be exhibited by t to reduce the complexity of solving this problem. As stated in [5], the problem of merging several lists can be reduced to the problem of merging two lists. Hereafter, we recall the reduced **Problem 1** proposed in [5] that we are interested in.

Problem 1 ([5]). *Let L_1 and L_2 be 2 lists of binary vectors of size 2^{ℓ_1} and 2^{ℓ_2} respectively. We denote by $\boldsymbol{x}$ a vector of L_1 and by $\boldsymbol{y}$ a vector of L_2.*

We assume that vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ can be decomposed into z groups of s bits, i.e. $\boldsymbol{x}, \boldsymbol{y} \in (\{0,1\}^s)^z$ and $\boldsymbol{x} = (x_1, \ldots, x_z)$ (resp. $\boldsymbol{y} = (y_1, \ldots, y_z)$). The vectors in L_1 and L_2 are drawn uniformly and independently at random from $\{0,1\}^{sz}$.

Let t be a Boolean function, $t : \{0,1\}^{sz} \times \{0,1\}^{sz} \to \{0,1\}$ such that there exist some functions $t_j : \{0,1\}^s \times \{0,1\}^s \to \{0,1\}$ which verify:

$$t(\boldsymbol{x}, \boldsymbol{y}) = 1 \quad \Longleftrightarrow \quad \forall j,\ 1 \leq j \leq z, \quad t_j(x_j, y_j) = 1.$$

***Problem 1** consists in computing the set $\mathcal{L}_{sol}$ of all 2-tuples $(\boldsymbol{x}, \boldsymbol{y})$ of $(L_1 \times L_2)$ verifying $t(\boldsymbol{x}, \boldsymbol{y}) = 1$. This operation is called* merging *the lists L_1 and L_2 with respect to t.*

One of the algorithms proposed in [5] to solve **Problem 1** is the *parallel matching* algorithm, which is the one that provides the best time complexity when the number of possible associated elements to one element is bigger than the size of the other list, *i.e.*, when we can associate by t more than $|L_2|$ elements to an element from L_1 as well as more than $|L_1|$ elements to an element from L_2.

In our case, the lists $\mathcal{L}_{\text{in}}$ and $\mathcal{L}_{\text{out}}$ correspond to the lists L_1 and L_2 to merge but the application of this algorithm differs in two aspects. The first one is the alphabet, which is not binary anymore but ternary. The second aspect is the distribution of vectors in the lists. In **Problem 1**, the elements are drawn uniformly and independently at random while in our case the distribution is ruled by the MITM technique we use. For instance, all the elements of $\mathcal{L}_{\text{in}}$ have the same weight x and all the elements of $\mathcal{L}_{\text{out}}$ have the same weight y, which is far from the uniform case.

The function t is the *association rule* we use to select suitable vectors from $\mathcal{L}_{\text{in}}$ and $\mathcal{L}_{\text{out}}$. We say that two elements are *associated* if their colliding active cells have the same values. We can now specify a new **Problem 1** adapted for ARMADILLO2:

ARMADILLO2 Problem 1. *Let $\mathcal{L}_{in}$ and $\mathcal{L}_{out}$ be 2 lists of ternary vectors of size $2^{\ell_{in}}$ and $2^{\ell_{out}}$ respectively. We denote by $\boldsymbol{x}$ a vector of $\mathcal{L}_{in}$ and by $\boldsymbol{y}$ a vector of $\mathcal{L}_{out}$, with $\boldsymbol{x}, \boldsymbol{y} \in \{0,1,-\}^k$*
The lists $\mathcal{L}_{in}$ and $\mathcal{L}_{out}$ are obtained by the MITM technique described in Paragraph 3.1. Let $t : \{0,1,-\}^k \times \{0,1,-\}^k \to \{0,1\}$ be the function defined by $t = t_1 \cdot t_2 \cdots t_{k-1} \cdot t_k$ and:

$$\forall j,\ 1 \leq j \leq k, \quad t_j : \{0,1,-\} \times \{0,1,-\} \to \{0,1\},$$

x_j	0	0	0	1	1	1	−	−	−
y_j	0	1	−	0	1	−	0	1	−
$t_j(x_j, y_j)$	1	0	1	0	1	1	1	1	1

We say that $\boldsymbol{x}$ and $\boldsymbol{y}$ are associated if $t(\boldsymbol{x}, \boldsymbol{y}) = 1$.
***ARMADILLO2 Problem 1** consists in* merging *the lists $\mathcal{L}_{in}$ and $\mathcal{L}_{out}$ with respect to t.*

We can now adapt the parallel matching algorithm to ARMADILLO2 **Problem 1**.

3.3 Applying the Parallel Matching Algorithm to ARMADILLO2

The principle of the parallel matching algorithm is to consider *in parallel* the possible matches for the α first cells and the next β cells in the lists $\mathcal{L}_{\text{in}}$ and $\mathcal{L}_{\text{out}}$. The underlying idea is to improve, when possible, the complexity to find all the elements that are a match for the $(\alpha+\beta)$ first cells. To have a match between a vector in $\mathcal{L}_{\text{in}}$ and a vector in $\mathcal{L}_{\text{out}}$, the vectors should satisfy:

- the vector in $\mathcal{L}_{\text{in}}$ has u of its x active cells among the $(\alpha+\beta)$ first cells;
- the vector in $\mathcal{L}_{\text{out}}$ has v of its y active cells among the $(\alpha+\beta)$ first cells;
- looking at the $(\alpha+\beta)$ first cells, both vectors should have the same value at the same active position.

As x and y are the number of known bits from $(C\|U)$ and from Y resp. (see Fig. 2), the matching probability on the first $(\alpha+\beta)$ cells is:

$$2^{-N_{\text{coll}}^{\alpha+\beta}} = \sum_{u=0}^{x} P_{[k,\alpha+\beta,x]}(u) \cdot \sum_{v=0}^{y} P_{[k,\alpha+\beta,y]}(v) \cdot \sum_{w=0}^{v} 2^{-w} P_{[\alpha+\beta,v,u]}(w).$$

This means that we will find $2^{c-N_{\text{coll}}^{\alpha+\beta}}$ partial solutions. For each pair passing the test we will have to check next if the remaining $k-\alpha-\beta$ cells are verified.

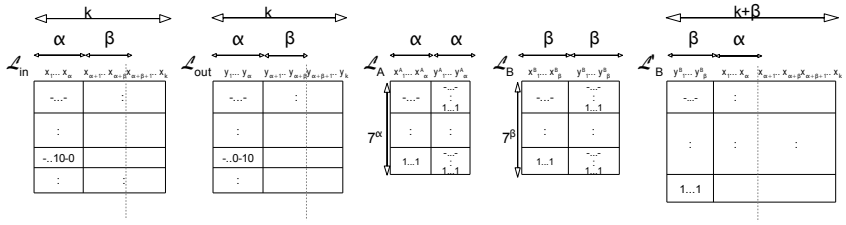


Fig. 3. Lists used in the parallel matching algorithm.

In a pre-processing phase, we first need to build three lists, namely $\mathcal{L}_A$, $\mathcal{L}_B$, $\mathcal{L}'_B$, which are represented in Fig. 3.

List $\mathcal{L}_A$ contains all the elements of the form $(x_1^A \dots x_\alpha^A, y_1^A \dots y_\alpha^A)$ with $(x_1^A \dots x_\alpha^A) \in \{0,1,-\}^\alpha$ and $(y_1^A \dots y_\alpha^A)$ being associated to $(x_1^A \dots x_\alpha^A)$. The size of $\mathcal{L}_A$ is:

$$|\mathcal{L}_A| = \sum_{i=0}^{\alpha} \left(\binom{\alpha}{i} 2^i 3^{\alpha-i} 2^i \right) = 7^\alpha.$$

List $\mathcal{L}_B$ contains all the elements of the form $(x_1^B \dots x_\beta^B, y_1^B \dots y_\beta^B)$ with $(x_1^B \dots x_\beta^B) \in \{0,1,-\}^\beta$ and $(y_1^B, \dots, y_\beta^B)$ being associated to $(x_1^B, \dots, x_\beta^B)$. The size of $\mathcal{L}_B$ is:

$$|\mathcal{L}_B| = \sum_{i=0}^{\beta} \left(\binom{\beta}{i} 2^i 3^{\beta-i} 2^i \right) = 7^\beta.$$

List $\mathcal{L}'_B$ contains for each element $(x_1^B, \dots, x_\beta^B, y_1^B, \dots, y_\beta^B)$ in $\mathcal{L}_B$ all the elements $\boldsymbol{x}$ from $\mathcal{L}_{\text{in}}$ such that $(x_{\alpha+1} \dots, x_{\alpha+\beta}) = (x_1^B, \dots, x_\beta^B)$. Elements in $\mathcal{L}'_B$ are of the form $(y_1^B, \dots, y_\beta^B, x_1, \dots, x_k)$ indexed[7] by $(y_1^B \dots, y_\beta^B, x_1, \dots, x_\alpha)$. The probability

[7] We can use standard hash tables for storage and look up in constant time.

for an element in $\mathcal{L}_{\text{in}}$ to have i active cells in its next β cells is $P_{[k,\beta,x]}(i)$. The size of $\mathcal{L}'_B$ is:

$$|\mathcal{L}'_B| = \sum_{i=0}^{\beta} \binom{\beta}{i} 2^i 3^{\beta-i} 2^i 2^{\ell_{\text{in}}} \frac{P_{[k,\beta,x]}(i)}{2^i \binom{\beta}{i}} = \sum_{i=0}^{\beta} 3^{\beta-i} 2^i 2^{\ell_{\text{in}}} P_{[k,\beta,x]}(i).$$

The cost of building $\mathcal{L}'_B$ is upper bounded by $(|\mathcal{L}'_B| + 3^\beta)$, where 3^β captures the cases where no element in $\mathcal{L}_{\text{in}}$ corresponds to elements in $\mathcal{L}_B$ and is normally negligible.

Next, we do the parallel matching. The probability for an element in $\mathcal{L}_{\text{out}}$ to have i active cells in its α first cells being $P_{[k,\alpha,y]}(i)$, for each element $(x_1^A \ldots x_\alpha^A, y_1^A \ldots y_\alpha^A)$ in $\mathcal{L}_A$ we consider the $2^{\ell_{\text{out}}} \frac{P_{[k,\alpha,y]}(i)}{2^i \binom{\alpha}{i}}$ elements $\boldsymbol{y}$ from $\mathcal{L}_{\text{out}}$ such that $(y_1, \ldots, y_\alpha) = (y_1^A, \ldots, y_\alpha^A)$. Then we check in $\mathcal{L}'_B$ if elements indexed by $(y_{\alpha+1} \ldots y_{\alpha+\beta}, x_1^A \ldots x_\alpha^A)$ exist. If this is the case, we check if each found pair of the form $(\boldsymbol{x}, \boldsymbol{y})$ verifies the remaining $(k - \alpha - \beta)$ cells. As we already noticed, we will find about $2^{c - N_{\text{coll}}^{\alpha+\beta}}$ partial solutions for which we will have to check whether or not they meet the remaining conditions. The time complexity of this algorithm is:

$$\mathcal{O}\left(2^{c-N_{\text{coll}}^{\alpha+\beta}} + 7^\alpha + 7^\beta + \sum_{i=0}^{\beta} 3^{\beta-i} 2^i 2^{\ell_{\text{in}}} P_{[k,\beta,x]}(i) + \sum_{i=0}^{\alpha} 3^{\alpha-i} 2^i 2^{\ell_{\text{out}}} P_{[k,\alpha,y]}(i)\right).$$

The memory complexity is determined by $7^\alpha + 7^\beta + |\mathcal{L}'_B|$. We can notice that if

$$\sum_{i=0}^{\beta} 3^{\beta-i} 2^i 2^{\ell_{\text{in}}} P_{[k,\beta,x]}(i) > \sum_{i=0}^{\alpha} 3^{\alpha-i} 2^i 2^{\ell_{\text{out}}} P_{[k,\alpha,y]}(i),$$

we can exchange the roles of $\mathcal{L}_{\text{in}}$ and $\mathcal{L}_{\text{out}}$, so that the time complexity remains the same but the memory complexity will be reduced. The memory complexity is then:

$$\mathcal{O}\left(7^\alpha + 7^\beta + \min\left\{\sum_{i=0}^{\beta} 3^{\beta-i} 2^i 2^{\ell_{\text{in}}} P_{[k,\beta,x]}(i), \sum_{i=0}^{\alpha} 3^{\alpha-i} 2^i 2^{\ell_{\text{out}}} P_{[k,\alpha,y]}(i)\right\}\right).$$

4 Meet in the Middle Key Recovery attacks

4.1 Key Recovery Attack in the FIL-MAC Setting

In the FIL-MAC usage scenario, C is the secret key and U is the challenge. The response is the m-bit size vector V_t. In order to minimize the complexity of our attack, we want the number of known bits y from Y to be maximal. As $Y = (V_c \| V_t) \oplus X$ and $X = Q_U(C \| U)$ it means that we are interested in having the maximum number of bits from U among the m less significant bits of X.

As we have m bits of freedom in U for choosing the permutation Q_U, we need the probability of having i known bits (from U) among the m first ones (of X), $P_{[k,m,m]}(i)$, to be bigger than 2^{-m}. Then to maximize the number of known bits in Y, we choose y as follows:

$$y = \max_{0 \le i \le m} \left\{ i : P_{[k,m,m]}(i) > 2^{-m} \right\}. \tag{1}$$

For instance for ARMADILLO2-A, we have $y = 38$ with a probability of $2^{-45.19} > 2^{-48}$.

Then, from now on, we assume that we know y among the m bits of the lower part of X and y bits at the same positions of Y.

Now, we can apply our meet-in-the-middle technique which allows us to recover the key. We have computed the optimal parameters for the different versions of

ARMADILLO2, with different trade-offs — the generic attack has a complexity of 2^c. The results appear in Table 2.

For each version of ARMADILLO2 presented in Table 2, the first line corresponds to the ($\log_2$ of the) size of the lists $\mathcal{L}_{\text{in}}$ and $\mathcal{L}_{\text{out}}$ with the smallest time complexity. The second line corresponds to the best parameters when limiting the memory complexity to 2^{45}. In all cases, the complexity is determined by the parallel matching part of the attack. The data complexity of all the attacks is 1, that is, we only need one pair of plaintext/ciphertext to succeed.

Version	c	m	ℓ_{out}	ℓ_{in}	α	β	$\log_2$(Time compl.)	$\log_2$(Mem. compl.)
ARMADILLO2-A	80	48	34	46	24	20	72.54	68.94
			18	62	16	9	75.05	45
ARMADILLO2-B	128	64	58	70	35	35	117.97	108.87
			38	90	2	16	125.15	45
ARMADILLO2-C	160	80	76	84	43	43	148.00	135.90
			35	125	4	16	156.63	45
ARMADILLO2-D	192	96	92	100	50	50	177.98	160.44
			29	163	11	12	187.86	45
ARMADILLO2-E	256	128	125	131	65	65	237.91	209.83
			29	227	11	13	251.55	45

Table 2. Complexities of the meet-in-the-middle key recovery attack on the FIL-MAC application

4.2 Key Recovery Attack in the Stream Cipher Setting

As presented in [1], ARMADILLO2 can be used as a PRNG by taking the t first bits of (V_c, V_t) after at least r iterations. For ARMADILLO2, the authors state in [1, Sc. 6] that $r = 1$ and $t = k$ is a suitable parameter choice. If we want to use it as a stream cipher, the secret key is C. The keystream is composed of k-bit frames indexed by U which is a public value.

In this setting, we can perform an attack which is similar to the one on the FIL-MAC, but with different parameters. As we know more bits of the output of Q_X, $y = m + \ell_{\text{out}}$, complexities of the key recovery attack are lower.

In general, the best time complexity is obtained when $\ell_{\text{in}} = \ell_{\text{out}}$, as the number of known bits at each side is now $x = m + \ell_{\text{in}}$ in the input and $y = m + \ell_{\text{out}}$ in the output. In this context it also appears that the best time complexity occurs when $\alpha = \beta$. There might be a small difference between α and β when the leading term of the time complexity is $2^{c - N_{\text{coll}}^{\alpha+\beta}}$.

We present the best complexities we have computed for this attack in Table 3 — the generic attack has a complexity of 2^c. Other time-memory trade-offs would be possible. As in the previous section, we give as an example the best parameters when limiting the memory complexity to 2^{45}.

5 (Second) Preimage Attack on the Hashing Applications

We recall that the hash function built with ARMADILLO2 as a compression function follows a strengthened Merkle-Damgård construction, where the padding includes the message length. In this case C represents the input chaining value, U the message block

Version	c	m	ℓ_{out}	ℓ_{in}	α	β	$\log_2$(Time compl.)	$\log_2$(Mem. compl.)
ARMADILLO2-A	80	48	40	40	19	19	65.23	62.91
			27	53	11	16	71.62	45
ARMADILLO2-B	128	64	64	64	31	32	104.71	101.75
			29	99	9	16	119.69	45
ARMADILLO2-C	160	80	80	80	39	40	130.53	127.49
			26	134	14	14	151.29	45
ARMADILLO2-D	192	96	96	96	47	48	156.35	153.23
			30	162	8	16	184.37	45
ARMADILLO2-E	256	128	128	128	64	64	207.96	205.93
			30	226	8	16	248.66	45

Table 3. Complexities of the meet-in-the-middle key recovery attack for the stream cipher with various trade-offs.

and V_c the generated new chaining value and the hash digest. In [1] the authors state that (second) preimages are expected with a complexity of 2^c, the one of the generic attack. We show, in this section, how to build (second) preimage attacks with a smaller complexity.

5.1 Meet-in-the-Middle (Second) Preimage Attack

The principle of the attack is represented in Fig. 5.1. We first consider that the ARMADILLO2 function is invertible with a complexity of 2^q, given an output V_c and a message block. In the preimage attack, we choose and fix ℓ, the number of blocks of the preimage. In the second preimage attack, we can consider the length of the given message. Then, given a hash value h:

In the backward direction:

- We invert the insertion of the last block M_{pad} (padding). This step costs 2^q in a preimage scenario and 1 in a second preimage one. We get
$$\text{ARMADILLO2}^{-1}(h, M_{\text{pad}}) = S'.$$
- From state S', we can invert the compression function for 2^b different message blocks M_b with a cost 2^{b+q}, obtaining 2^b different intermediate states: $\text{ARMADILLO2}^{-1}(S', M_b) = S''$.

In the forward direction: From the initial chaining value, we insert 2^a messages of length $(\ell - 2)$ blocks, $\mathcal{M} = M_1 \| M_2 \| \dots \| M_{\ell-2}$, obtaining 2^a intermediate states S. This can be done with a complexity of $\mathcal{O}((\ell - 2)2^a)$.

If we find a collision between one of the 2^a states S and one of the 2^b states S'', we have obtained a (second) preimage that is $\mathcal{M} \| M_b \| M_{\text{pad}}$.

A collision occurs if $a + b \geq c$. The complexity of this attack is $2^a + 2^q + 2^{b+q}$ in time, where the middle term appears only in the case of a preimage attack and is negligible. The memory complexity is about 2^b (plus the memory needed for inverting the compression function). So if $2^q < 2^c$, we can find a and b so that $2^a + 2^{b+q} < 2^c$.

5.2 Inverting the Compression Function

In the previous section we showed that inverting the compression function for a chosen message block and for a given output can be done with a cost of $2^q < 2^c$. In this section we show how this complexity depends on the chosen message block, as the inversion

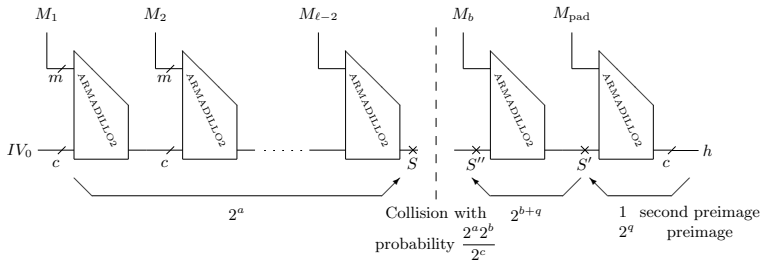


Fig. 4. Representation of the meet-in-the-middle (second) preimage attack.

can be seen as a key recovery similar to the one done in Section 4. In this case we know U (the message block) and V_c, and we want to find C. When inverting the function with the blocks M_b, we choose message blocks (U) that define permutations Q_U which put most of the m bits from U among the c most significant bits of X. This will result in better attacks, as the bits in Y known from U do not cost anything and this gives us more freedom when choosing the parameters ℓ_{in} and ℓ_{out}.

As before, we have 2^m possibilities for Q_U. We denote by n the number of bits of U in the c most significant bits of X. The number of message blocks (U) verifying this condition is:

$$N_{\text{block}}(n) = 2^m P_{[k,c,m]}(n).$$

In fact we are interested in the values of n which are the greatest possible (to lower the complexity) that still leaves enough message blocks to invert in order to obtain S''. It means that these values belong to a set $\{n_i\}$ such that:

$$\sum_{\{n_i\}} N_{\text{block}}(n_i) \geq 2^b.$$

As the output is V_c, the ℓ_{out} bits guessed from X are also known bits from the output of Q_X. The number of known bits of the output of Q_X is then defined by:

$$y = \min(c, \ell_{\text{out}} + n)$$

Compared to the key recovery attack, the number of known bits at the end of the permutation Q_X is significantly bigger, as we may know up to c bits, while in the previous case the maximal number for y was $y = \max_i \left\{i : P_{[k,m,m]}(i) > 2^{-m}\right\}$. To simplify the explanations, we concentrate on the case of ARMADILLO2-A, that can be directly adapted to any of the other versions. For $n = 48$ we have a probability $P_{[128,80,48]} = 2^{-44.171}$. This leaves $2^{48-44.171} = 2^{3.829}$ message blocks to invert which allow us to know $y = \min(80, \ell_{\text{out}} + 48)$ bits from the output of Q_X. As we need to invert 2^b message blocks, if b is bigger than 3.829, we have to consider next the message blocks with $n = 47$, that allow us to know $y = \min(80, \ell_{\text{out}} + 47)$ bits, and so on. For each n considered, the best time complexity (2^{q_n}) for inverting ARMADILLO2 might be different, but in practice, with at most two consecutive values of n we have enough message blocks for building the attack, and the complexity of inverting the compression function for these two different types of messages is very similar.

For instance, in ARMADILLO2-A, we consider $n = 48, 47$, associated each to $2^{3.829}$ and $2^{9.96}$ possible message blocks respectively. The best time complexity for inverting the compression function in both cases is $2^{q_{48}} = 2^{q_{47}} = 2^{65.9}$, as we can see from Table 4. If we want to find the best parameters for a and b in the preimage attack, we can consider that $a+b = c$ and $2^b = 2^{b_{48}} + 2^{b_{47}}$, and we want that $2^a = 2^{b_{48}} 2^{65.9} + 2^{b_{47}} 2^{65.9} = 2^{65.9}(2^{b_{48}} + 2^{b_{47}})$, as the complexity of the attack is $\mathcal{O}(2^a + 2^{65.9}(2^{b_{48}} + 2^{b_{47}}))$. So if we choose the parameters correctly, the best time complexity will be $\mathcal{O}(2^{a+1})$.

Version	c	m	ℓ_{out}	ℓ_{in}	n	$\log_2(N_{\text{block}}(n))$	α	β	$\log_2$(Time compl.)	$\log_2$(Mem. compl.)
ARMADILLO2-A	80	48	35	45	47	9.95	22	16	65.90	63.08
			35	45	48	3.83	22	16	65.90	63.08
			20	60	47	9.95	16	8	71.36	45
			27	53	48	3.83	11	16	71.62	45
ARMADILLO2-B	128	64	62	66	64	15.89	33	30	104.67	102.35
			33	95	64	15.89	6	16	120.41	45
ARMADILLO2-C	160	80	78	82	80	19.82	41	38	130.48	128.08
			26	134	80	19.82	11	16	152.24	45
ARMADILLO2-D	192	96	94	98	96	23.74	49	46	156.31	153.82
			30	162	96	23.74	8	16	184.37	45
ARMADILLO2-E	256	128	126	130	128	31.58	65	62	207.96	205.30
			34	222	128	31.58	5	16	249.47	45

Table 4. Complexities for inverting the compression function.

In this particular case the time complexity for $n = 48$ and for $n = 47$ is the same, so finding the best b and a can be simplified by $b = \frac{c-q}{2}$ and $a = c - b$. We obtain $b = 7.275$, $a = 72.95$. We see that we do not have enough elements with $n = 48$ for inverting 2^b blocks, but we have enough with $n = 47$ alone. As the complexities are the same in both cases, we can just consider $b = b_{47}$. The best time complexity for the preimage attack that we can obtain is then $2^{73.95}$, with a memory complexity of $2^{63.08}$. Other trade-offs are possible by using other parameters for inverting the function, as shown in Table 5.

For the other versions of ARMADILLO2, the number of message blocks associated to $y = m$ is big enough for performing the 2^b inversions, so we do not consider other n's for computing the (second) preimage complexity. Then, $b = b_m = \frac{c-q_{\{n=m\}}}{2}$ and $a = c - b_m$.

Complexities for preimage attacks on the different versions of ARMADILLO2 are given in Table 5, where we can see two different complexities with different trade-offs for each version.

Version	c	m	Best time		Time-memory trade-off	
			$\log_2$(Time compl.)	$\log_2$(Mem. compl.)	$\log_2$(Time compl.)	$\log_2$(Mem. compl.)
ARMADILLO2-A	80	48	73.95	63.08	76.81	45
ARMADILLO2-B	128	64	117.34	102.35	125.21	45
ARMADILLO2-C	160	80	146.24	128.08	157.12	45
ARMADILLO2-D	192	96	175.16	153.82	191.19	45
ARMADILLO2-E	256	128	232.98	205.30	253.74	45

Table 5. Complexities of the (second) preimages attacks.

6 Experimental Verifications

To verify the above theoretical results, we implemented the proposed key recovery attacks in the FIL-MAC and stream cipher settings against a scaled version of ARMADILLO2 that uses a 30-bit key and processes 18-bit messages, i.e. $c = 30$ and $m = 18$. We performed the attack 10 times for both the FIL-MAC and the PRNG settings where at each time we chose random permutations for both σ_0 and σ_1 and random messages U (in the FIL-MAC case U was chosen so that we got y bits from U among the m least significant bits of X).

As for each application the key is a 30-bit key, the generic attack requires a time complexity of 2^{30}. Using the parallel matching algorithm we decrease this complexity. Table 6 shows that the implementation results are very close to the theoretical estimates, confirming our analysis. We can also mention that we exchanged the role of $\mathcal{L}_{\text{in}}$ and $\mathcal{L}_{\text{out}}$ in our implementation of the attacks to minimize the memory needs.

		c	m	ℓ_{out}	ℓ_{in}	α	β	y	$\log_2(\lvert\mathcal{L}'_B\rvert)$	$\log_2(c - N_{\text{coll}}^{\alpha+\beta})$	$\log_2$(Time compl.)	$\log_2$(Mem. compl.)
FIL-MAC	Impl.	30	18	12	18	8	6	14	23.477	27.537	27.874	24.066
	Theory	30	18	12	18	8	6	14	23.475	27.538	27.874	24.064
PRNG	Impl.	30	18	14	16	7	6	32	22.530	24.728	25.396	22.738
	Theory	30	18	14	16	7	6	32	22.530	24.735	25.401	22.738

Table 6. Key recovery attacks against a scaled version of ARMADILLO2 in the FIL-MAC and PRNG modes.

7 Generalization of the Parallel Matching Algorithm

In Section 3, we managed to apply the parallel matching algorithm to invert the ARMADILLO2 function by modifying the merging **Problem 1** of [5].

When the number of possible associated elements to one element is bigger than the other list as it is the case for ARMADILLO2, we cannot apply a basic algorithm like the *instant matching* algorithm proposed in [5]. Instead, we can use either the *gradual matching* or the *parallel matching* algorithms also proposed in [5]. We are going to concentrate on the parallel matching algorithm which allows a significant reduction of the time complexity of solving **Problem 1**, while allowing several time-memory tradeoffs.

We can state the generalized problem that also covers our attack on ARMADILLO2 and give the corresponding parallel matching algorithm. We believe that this more general problem will be useful for recognizing situations where the parallel matching can be applied, and solving them in an automatized way.

7.1 The Generalized Problem 1

As stated in [5], **Problem 1** for N lists can be reduced to 2 lists, therefore we will only consider the problem of merging 2 lists in the sequel.

Generalized Problem 1. *We are given 2 lists, L_1 and L_2 of size 2^{ℓ_1} and 2^{ℓ_2} respectively. We denote by $\boldsymbol{x}$ a vector of L_1 and by $\boldsymbol{y}$ a vector of L_2. Coordinates of $\boldsymbol{x}$ and $\boldsymbol{y}$ belong to a general alphabet $\mathcal{A}$.*

We assume that vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ can be decomposed into z groups of s coordinates, i.e. $\boldsymbol{x}, \boldsymbol{y} \in (\mathcal{A}^s)^z$ and $\boldsymbol{x} = (x_1, \ldots, x_z)$ (resp. $\boldsymbol{y} = (y_1, \ldots, y_z)$).

We want to keep pairs of vectors verifying a given relation t: $t(\boldsymbol{x}, \boldsymbol{y}) = 1$. The relation t is group-wise,and is defined by $t : (\mathcal{A}^s)^z \times (\mathcal{A}^s)^z \to \{0,1\}$ such that there exist some functions $t_j : \mathcal{A}^s \times \mathcal{A}^s \to \{0,1\}$, verifying:

$$t(\boldsymbol{x}, \boldsymbol{y}) = 1 \iff \forall j,\ 1 \le j \le z, \quad t_j(x_j, y_j) = 1.$$

Generalized Problem 1 *consists in* merging *these* 2 *lists to obtain the set $\mathcal{L}_{sol}$ of all* 2*-tuples of $(L_1 \times L_2)$ verifying $t(\boldsymbol{x}, \boldsymbol{y}) = 1$. We say that $\boldsymbol{x}$ and $\boldsymbol{y}$ are associated in this case.*

In order to analyze the time and memory complexities of the attack we need to compute the size of $\mathcal{L}_{\text{sol}}$. This quantity depends on the probability that $t(\boldsymbol{x}, \boldsymbol{y}) = 1$. More precisely the complexities of the generalized parallel matching algorithm depends on the conditional probabilities: $\mathbf{Pr}_{y_j}[t_j(x_j, y_j) = 1 | x_j = a]$, $a \in \mathcal{A}^s$. We will denote these probabilities by $p_{j,a}$, $a \in \mathcal{A}^s$.

In [5] the elements of the lists L_1 and L_2 were binary (i.e. $\mathcal{A} = \{0,1\}$) and random, and the probability of each t_j of being verified did not depend on the elements x_j or y_j. Let us consider as an example the case where $s = 1$ and t_j tests the equality of x_j and y_j. We have:

$$\forall j,\ 1 \le j \le z, \quad p_{j,0} = p_{j,1} = \frac{1}{2}.$$

In the case of the ARMADILLO2 cryptanalysis that we present in this paper, the alphabet is ternary (i.e. $\mathcal{A} = \{0,1,-\}$) and the association rule (*see. ARMADILLO2 Problem 1*) gives:

$$\forall j,\ 1 \le j \le z, \quad p_{j,0} = \frac{2}{3},\ p_{j,1} = \frac{2}{3} \text{ and } p_{j,-} = 1$$

7.2 Generalized Parallel Matching Algorithm

First we need to build the three following lists:

List $\mathcal{L}_A$, of all the elements of the form $(x_1^A, \ldots, x_\alpha^A, y_1^A, \ldots, y_\alpha^A)$ with $(x_1^A, \ldots, x_\alpha^A) \in (\mathcal{A}^s)^\alpha$ and $(y_1^A, \ldots, y_\alpha^A)$ being associated by t to $(x_1^A, \ldots, x_\alpha^A)$. The size of $\mathcal{L}_A$ is:

$$|\mathcal{L}_A| = \sum_{\mathbf{a} \in (\mathcal{A}^s)^\alpha} \prod_{j=1}^{\alpha} |\mathcal{A}|^s \, p_{j,\mathbf{a}_j}, \tag{2}$$

where $\mathbf{a}_j$ is the j-th coordinate of $\mathbf{a} \in (\mathcal{A}^s)^\alpha$.

List $\mathcal{L}_B$, of all the elements of the form $(x_1^B, \ldots, x_\beta^B, y_1^B, \ldots, y_\beta^B)$ with $(x_1^B, \ldots, x_\beta^B) \in (\mathcal{A}^s)^\beta$ and $(y_1^B, \ldots, y_\beta^B)$ being associated by t to $(x_1^B, \ldots, x_\beta^B)$. The size of $\mathcal{L}_B$ is

$$|\mathcal{L}_B| = \sum_{\mathbf{b} \in (\mathcal{A}^s)^\beta} \prod_{j=1}^{\beta} |\mathcal{A}|^s \, p_{j,\mathbf{b}_j},$$

where $\mathbf{b}_j$ is the j-th coordinate of $\mathbf{b} \in (\mathcal{A}^s)^\beta$.

List $\mathcal{L}'_B$, containing for each element $(x_1^B, \ldots, x_\beta^B, y_1^B, \ldots, y_\beta^B)$ in $\mathcal{L}_B$ all the elements $\boldsymbol{x}$ from L_1 such that $(x_{\alpha+1} \ldots, x_{\alpha+\beta}) = (x_1^B, \ldots, x_\beta^B)$. Elements in $\mathcal{L}'_B$ are of the form $(y_1^B, \ldots, y_\beta^B, x_1, \ldots, x_z)$ indexed[8] by $(y_1^B \ldots, y_\beta^B, x_1, \ldots, x_\alpha)$. If we denote by $\mathrm{P}_{\mathbf{b},[\alpha+1,\alpha+\beta],L_1}$ the probability of having an element $\boldsymbol{x}$ from L_1 such that $(x_{\alpha+1}, \ldots, x_{\alpha+\beta}) = \mathbf{b}$, the size of $\mathcal{L}'_B$ is:

$$|\mathcal{L}'_B| = \sum_{\mathbf{b} \in (\mathcal{A}^s)^\beta} \left(\prod_{j=1}^{\beta} |\mathcal{A}|^s \, p_{j,\mathbf{b}_j} \right) 2^{\ell_1} \mathrm{P}_{\mathbf{b},[\alpha+1,\alpha+\beta],L_1}.$$

[8] We can use standard hash tables for storage and look up in constant time.

The cost of building this list is upper-bounded by $(|\mathcal{L}'_B|+(|\mathcal{A}|)^\beta)$, where the second term captures the cases where no element in L_1 corresponds to elements in $\mathcal{L}_B$ and should be negligible.

In the case where

$$\sum_{\mathbf{a}\in(\mathcal{A}^s)^\alpha}\left(\prod_{j=1}^{\alpha}|\mathcal{A}|^s\, p_{j,\mathbf{a}_j}\right)2^{\ell_2}\mathrm{P}_{\mathbf{a},[\beta+1,\alpha+\beta],L_2} < \sum_{\mathbf{b}\in(\mathcal{A}^s)^\beta}\left(\prod_{j=1}^{\beta}|\mathcal{A}|^s\, p_{j,\mathbf{b}_j}\right)2^{\ell_1}\mathrm{P}_{\mathbf{b},[\alpha+1,\alpha+\beta],L_1}$$

we can swap L_1 and L_2, to reduce the memory complexity of the attack.

Next, we do the parallel matching. For each element $(x_1^A,\ldots,x_\alpha^A,y_1^A,\ldots,y_\alpha^A)$ in $\mathcal{L}_A$ we consider the $2^{\ell_2}\mathrm{P}_{(y_1^A,\ldots,y_\alpha^A),[1,\alpha],L_2}$ elements $\boldsymbol{y}$ from L_2 such that $(y_1\ldots y_\alpha)=(y_1^A,\ldots,y_\alpha^A)$ and we check in $\mathcal{L}'_B$ if elements indexed by $(y_{\alpha+1}\ldots y_{\alpha+\beta},x_1^A\ldots x_\alpha^A)$ exist. If this is the case, we check if each found pair of the form $(\boldsymbol{x},\boldsymbol{y})$ verifies the remaining $(k-\alpha-\beta)$ cells. We denote by Ω the number of partial solutions for which we will have to check whether or not they meet the remaining conditions:

$$\Omega = 2^{\ell_1+\ell_2}\sum_{\mathbf{b}\in(\mathcal{A}^s)^{\alpha+\beta}}\left(\prod_{j=1}^{\alpha+\beta}p_{j,\mathbf{b}_j}\right)\mathrm{P}_{\mathbf{b},[1,\alpha+\beta],L_1}$$

The time complexity of this algorithm is:

$$\mathcal{O}\left(\Omega+|\mathcal{L}_A|+|\mathcal{L}_B|+|\mathcal{L}'_B|+\sum_{\mathbf{a}\in(\mathcal{A}^s)^\alpha}\left(\prod_{j=1}^{\alpha}|\mathcal{A}|^s\, p_{j,\mathbf{a}_j}\right)2^{\ell_2}\mathrm{P}_{\mathbf{a},[\beta+1,\alpha+\beta],L_2}\right)$$

The memory complexity is determined by the size of the lists $\mathcal{L}_A$, $\mathcal{L}_B$ and $\mathcal{L}'_B$. Therefore the memory complexity is:

$$\sum_{\mathbf{a}\in(\mathcal{A}^s)^\alpha}\prod_{j=1}^{\alpha}|\mathcal{A}|^s\, p_{j,\mathbf{a}_j}+\sum_{\mathbf{b}\in(\mathcal{A}^s)^\beta}\prod_{j=1}^{\beta}|\mathcal{A}|^s\, p_{j,\mathbf{b}_j}+\sum_{\mathbf{b}\in(\mathcal{A}^s)^\beta}\left(\prod_{j=1}^{\beta}|\mathcal{A}|^s\, p_{j,\mathbf{b}_j}\right)2^{\ell_1}\mathrm{P}_{\mathbf{b},[\alpha+1,\alpha+\beta],L_1}$$

7.3 Link with Formulas in the Case of ARMADILLO

Using the previous formulas for the time and memory complexities, we can rediscover formulas of the time and memory complexities we have computed for ARMADILLO2 (see. Section 3.3). As these formulas depend essentially on the size of the different lists, we simply expose how to find the size of the list $|\mathcal{L}_A|$ using equation (2).

For ARMADILLO2, the probabilities $p_{j,a}$ are independent of the position j and $p_{j,a}=2/3$ if and only if a is an active cell. Moreover, in this case, each cell is composed of one letter of the alphabet which means that $s=1$. And we have:

$$\begin{aligned}|\mathcal{L}_A| &= \sum_{\mathbf{a}\in(\mathcal{A}^s)^\alpha}\prod_{j=1}^{\alpha}|\mathcal{A}|^s\, p_{j,\mathbf{a}_j} = \sum_{\mathbf{a}\in\{0,1,-\}^\alpha}\prod_{j=1}^{\alpha}3\left(\frac{2}{3}\right)^{\mathrm{wt}(\mathbf{a})}\\ &= \sum_{i=0}^{\alpha}\#\left\{\mathbf{a}:\mathrm{wt}(\mathbf{a})=i\right\}3^\alpha\left(\frac{2}{3}\right)^i = \sum_{i=0}^{\alpha}\binom{\alpha}{i}2^i\left(\frac{2}{3}\right)^i 3^\alpha\end{aligned}$$

The same method can be applied to find the size of the list $\mathcal{L}_B$ and $\mathcal{L}'_B$. Here we have $\Omega = 2^{c-N_{\mathrm{coll}}^{\alpha+\beta}}$.

8 Conclusion

In this paper, we have presented the first cryptanalysis of ARMADILLO2, the recommended variant of the ARMADILLO family. We propose a key recovery attack on all its versions for the FIL-MAC and the stream cipher mode, which works for any bitwise permutations σ_0 and σ_1. We give several time-memory trade-offs for its complexity. We also show how to build (second) preimage attacks when using the hashing mode.

Besides the results on ARMADILLO2, we have generalized the parallel matching algorithm presented in [5] for solving a wider **Problem 1** which includes the cases where the lists to merge do not have random elements. We believe that new types of meet-in-the-middle attacks might appear now given this algorithm that is cheaper than exhaustive search.

References

1. Badel, S., Dagtekin, N., Nakahara, J., Ouafi, K., Reffé, N., Sepehrdad, P., Susil, P., Vaudenay, S.: ARMADILLO: A Multi-purpose Cryptographic Primitive Dedicated to Hardware. In: Workshop on Cryptographic Hardware and Embedded Systems, CHES 2010. Lecture Notes in Computer Science, vol. 6225, pp. 398–412 (2010)
2. Khovratovich, D., Naya-Plasencia, M., Röck, A., Schläffer, M.: Cryptanalysis of Luffa v2 components. In: Selected Areas in Cryptography 17th International Workshop, SAC 2010. Lecture Notes in Computer Science, vol. 6544, pp. 388–409 (2010)
3. Moldovyan, A.A., Moldovyan, N.A.: A Cipher Based on Data-Dependent Permutations. Journal of Cryptology 15(1), 61–72 (2002)
4. Naya-Plasencia, M.: How to Improve Rebound Attacks. Tech. Rep. Report 2010/607, Cryptology ePrint Archive (2010), (extended version). http://eprint.iacr.org/2010/607.pdf
5. Naya-Plasencia, M.: How to Improve Rebound Attacks. In: Advances in Crypology: CRYPTO 2011. Lecture Notes in Computer Science, vol. 6841, pp. 188–205 (2011)
6. Sepehrdad, P., Sušil, P., Vaudenay, S.: Fast Key Recovery Attack on ARMADILLO1 and Variants. In: Tenth Smart Card Research and Advanced Application Conference, CARDIS 2011. Lecture Notes in Computer Science (2011), to appear