Improved Techniques in Differential Cryptanalysis

Nicolas David

THÈSE DE DOCTORAT DE SORBONNE UNIVERSITÉ

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Nicolas DAVID

Pour obtenir le grade de

DOCTEUR de SORBONNE UNIVERSITÉ

Improved Techniques in Differential Cryptanalysis

soutenue publiquement le 8 Novembre 2023

devant le jury composé de :

María Naya-Plasencia	Inria de Paris	Directrice
Willi Meier	FHNW	Rapporteur
Marine Minier	LORIA	Rapportrice
Christina Boura	Université de Versailles	Examinatrice
Patrick Derbez	Université de Rennes	Examinateur
Henri Gilbert	ANSSI	Examinateur
Thomas Peyrin	Nanyang Technological University	Président du Jury
Sondre Rønjom	University of Bergen	Examinateur

Remerciements

Avant d'entamer toute discussion scientifique, ce manuscrit débutera par des remerciements. En effet, la réalisation de cette thèse n'a été possible que grâce à l'indispensable soutien de nombreuses personnes.

Tout d'abord, je voudrais exprimer ma gratitude envers Maria qui m'a ouvert au monde de la cryptographie symétrique et qui m'a encadré lors de ces trois années de thèse tout en faisant preuve de disponibilité et d'enthousiasme.

J'aimerais également remercier les autres membres du jury : Christina Boura, Patrick Derbez, Henri Gilbert Willi Meier, Marine Minier, Thomas Peyrin et Sondre Rønjom avec une attention particulière envers Marine et Willi qui ont relu le manuscrit.

Merci à tous les membres de l'équipe COSMIQ avec qui j'ai eu la chance de partager un espace de travail à l'ambiance saine et positive: André.S, André.C, Anne, Anthony, Antonio, Augustin, Aurélie, Aurélien, Agathe, Axel, Charles, Christina, Clara, Clémence, Daniel, Dounia, Ferdinand, Gaëtan, Jean-Pierre, Johanna, Jules, Kévin, Léo, Maria, Maxime, Nicholas, Nicolas, Paul, Rachelle, Ritam, Rocco, Simona, Thomas, Valentin, Virgile, Xavier et Yann. Je souhaite remercier tout particulièrement Christelle sans qui aucun voyage ni aucune soutenance ne pourrait avoir lieu.

Je remercie David Lefranc pour avoir retenu ma candidature pour le poste d'ingénieur cryptologue au sein du groupe Thales. J'ai hâte de rejoindre l'équipe de cryptographie à Gennevilliers.

Merci à André, Augustin, Clara, Dounia et Paul pour avoir partagé avec moi le bureau "symétrique" placé stratégiquement proche de la machine a café et des mots croisés.

Merci à l'Agos de fournir le baby-foot, synonyme de parties endiablées qui révèlent des facettes insoupçonnées des personnalités de certains membres de l'équipe.

Je voudrais remercier tous mes coauteurs qui m'ont appris beaucoup tant sur le plan scientifique que sur le plan humain: Akinori, André, Antonio, Augustin, Christina, Christof, Frederico, Gaëtan, Gregor, Marek, Maria, Patrick, Rachelle, Thomas et Yosuke.

J'aimerais exprimer ma gratitude envers les nombreuses personnes que j'ai rencontrées lors de mon voyage au Japon, qui a précédé ma thèse: Akinori, Alexandre, Ky, Mehdi, Miguel, Thomas et Yu. Ces rencontres m'ont permis de traverser la période du confinement d'une manière plus apaisée.

Pour finir, j'aimerais exprimer ma gratitude envers mes parents et mes frères pour leur soutien inestimable tout au long de ces trois années de thèse, et des 23 autres qui ont précédées.

Contents

List of Figures

List of Tables

Présentation des travaux

Le manuscrit de thèse présenté s'intitule "Techniques améliorées en cryptanalyse différentielle". Cette thèse en informatique se concentre sur le domaine de la cryptographie, en particulier sur la cryptanalyse différentielle et s'étend sur un total de six chapitres. Dans un premier temps, deux chapitres d'introductions sont présentés: Préliminaires et Cryptanalyse différentielle, suivis de quatre chapitres constitués de mes contributions scientifiques : Cryptanalyse différentielle-linéaire améliorée, application sur Chaskey; Attaque différentielle de la version complète de SPEEDY-7-192; Cryptanalyse différentielle-MITM[1]; Cryptanalyse différentielle impossible quantique.

Chapitre 1 : Préliminaires

Dans ce chapitre, je vais commencer par fournir une description historique de la cryptographie, en commençant par la cryptographie classique, puis en passant à la cryptographie moderne, et enfin en abordant la cryptographie de demain. Après cette introduction, je vais décrire quelques constructions importantes en cryptographie symétrique. Bien que je présente les constructions de chiffrement de flux, de fonction de hachage et de code d'authentification de message, je fournirai surtout une description approfondie ainsi qu'une définition formelle du chiffrement par bloc puisqu' il s'agit de la construction qui sera le plus étudiée pendant cette thèse. Ce dernier peut être décrit comme une fonction qui prend en entrée une clef secrète et renvoie une permutation sur un espace de message de taille fixé. La sécurité de cette construction réside entre autres sur la difficulté qu'un attaquant a à distinguer cette permutation d'une permutation aléatoire.

Ensuite, je vais aborder des éléments de cryptanalyse symétrique. Je commencerai par présenter les modèles dans lesquels l'adversaire se place pour tenter d'attaquer les chiffrements par bloc en décrivant notamment les principes de distingueur de récupération de clef. Cette partie sera suivie d'une description des techniques de cryptanalyse symétrique bien connues, telles que la cryptanalyse linéaire et la cryptanalyse MITM (Meet-in-the-Middle). La cryptanalyse différentielle, qui est également une technique très importante de cryptanalyse symétrique, sera abordée dans le chapitre suivant.

[1]MITM vient de Meet-in-the middle qui se traduit par rencontre au milieu

Pour terminer cette section introductive, je fournirai quelques éléments d'informatique quantique qui se révéleront utiles pour le chapitre 6, approfondissant ainsi notre compréhension des concepts nécessaires à l'étude de la cryptographie quantique.

Chapitre 2 : Cryptanalyse Différentielle

Ce second chapitre introductif se focalise sur la description de la cryptanalyse différentielle. Cette technique consiste à exploiter la propagation de différences à travers un chiffrement par bloc. Dans un premier temps, je décrirai les éléments qui constituent un distingueur différentiel puis nous décrirons le déroulement d'une attaque différentielle. Cette attaque se déroule en trois temps :

- *Création des données.* Dans cette étape, le but est d'engendrer des données qui ont une différence en entrée appartenant à un certain ensemble décrit par la propagation du distingueur différentiel : les structures. Dans cette partie, je décris en détail les équations qui permettent de calculer la complexité en données et en mémoire de l'attaque.
- *Criblage des paires.* Cette étape débute avec les paires de messages clairs/chiffrés obtenues dans l'étape précédente. L'objectif est maintenant de créer un ensemble de paires différentielles qui pourraient potentiellement suivre le chemin différentiel en entrée et en sortie du chiffrement par bloc. Il s'agit donc de cribler l'ensemble des paires différentielles pour conserver uniquement celles qui nous intéressent.
- *Récupération de clef.* Dans cette dernière étape, nous utiliserons les paires différentielles gardées précédemment pour retrouver certains bits de la clef secrète en utilisant le distingueur différentiel. Bien qu'il existe différentes méthodes pour effectuer cette étape, je présenterai la technique de l'arrêt anticipé (early abort) qui permet souvent de retrouver la clef efficacement.

À la suite de cette description, je détaille les complexités en temps, mémoire et données associées à cette attaque. Dans la suite de ce chapitre, je donnerai la description de différentes variations de la cryptanalyse différentielle : cryptanalyse différentielle impossible, cryptanalyse différentielle linéaire.

La cryptanalyse différentielle impossible se construit de manière similaire à la cryptanalyse différentielle classique, à la différence notable que le distingueur différentiel a désormais une probabilité associée nulle plutôt qu'anormalement élevée. Le but sera dorénavant d'éliminer les mauvaises clefs afin de conserver seulement la clef utilisée lors du chiffrement. Si cette méthode peut aussi être décrite suivant les mêmes trois étapes que la cryptanalyse différentielle, la description de l'algorithme d'early abort se fait plus naturellement avec un point de vue différent. Ce second point de vue permet une traduction plus naturelle vers la version quantique des attaques différentielles impossibles décrite dans le chapitre 6.

La cryptanalyse différentielle-linéaire consiste au regroupement de la cryptanalyse linéaire et de la cryptanalyse différentielle. En effet, en combinant des approximations différentielles et linéaires, il est possible de construire une nouvelle forme de distingueur. Dans cette section, je donnerai une description précise de la construction de ce distingueur et de l'exploitation que l'on peut en tirer. Un exemple de cryptanalyse différentielle linéaire est donné dans le chapitre suivant sur le chiffrement Chaskey.

Chapitre 3 : Cryptanalyse différentielle-linéaire améliorée, application sur Chaskey

Dans ce chapitre, je vais présenter un résumé de notre travail collaboratif avec Marek Broll, Frederico Canale, Antonio Flórez-Gutiérrez, Gregor Leander, María Naya-Plasencia et Yosuke Todo, portant sur les optimisations de la cryptanalyse différentielle-linéaire des constructions ARX, ainsi que son application à Chaskey [Bei+22]. Notre article explore différentes optimisations indépendantes les unes des autres. Dans ce chapitre, je mettrai particulièrement l'accent sur la partie de l'article traitant des améliorations apportées à la composante différentielle, étant donné que c'est ma contribution spécifique à cette recherche.

Dans l'article [BLT20], les auteurs ont présenté une technique visant à améliorer les attaques différentielles-linéaires et à les appliquer à Chaskey et Chacha. À la suite de cette étude, nous avons continué à perfectionner les attaques différentielles-linéaires dans le cadre de notre travail décrit dans [Bei+22]. Dans la première section de ce chapitre, nous allons détailler la technique introduite dans l'article [BLT20], qui permet d'améliorer la corrélation d'un distingueur différentiel-linéaire et on montrera comment. Cette amélioration est possible grâce à l'étude d'espace neutre. Il s'agit d'espace affine qui vérifie une certaine équation différentielle. L'utilisation de ces espaces permet d'améliorer la corrélation du distingueur différentiel linéaire considéré. Ainsi, il est possible de faire des attaques plus performantes qu'auparavant. Pour calculer ces espaces neutres, je décrirai deux approches. La première approche est algorithmique et consiste à tester des vecteurs de poids faible afin de construire empiriquement l'espace neutre. La seconde approche s'inspire de la cryptanalyse conditionnelle différentielle et donne des méthodes pour retrouver des vecteurs neutre. Bien que laborieuse, cette méthode permet de retrouver des vecteurs de poids élevés qui ne peuvent pas être détéctés par la méthode algorithmique.

La seconde partie de ce chapitre se concentre sur l'application de ces améliorations au chiffrement par bloc Chaskey. Tout d'abord, je fournirai une description détaillée de Chaskey lui-même. Ensuite, j'exposerai l'espace neutre obtenu grâce à l'algorithme ainsi que les techniques de conditions présentées dans la section qui précède. Grâce à ces applications, de nouvelles attaques plus efficaces ont été proposées pour différentes versions de Chaskey, offrant ainsi des améliorations significatives dans la cryptanalyse

de ce chiffrement.

Chapitre 4 : Attaque différentielle de la version complète de SPEEDY-7-192

Ce chapitre tire son inspiration de l'article [Bou+23a], fruit d'une collaboration avec Christina Boura, Rachelle Heim Boissier et María Naya-Plasencia. Dans cette publication, nous avons présenté une attaque complète contre la version principale du chiffrement par bloc Speedy [Lea+21] en utilisant la cryptanalyse différentielle. Cette étude a permis de découvrir une vulnérabilité majeure dans le système de chiffrement Speedy, remettant en question sa sécurité et soulignant l'importance de la cryptanalyse différentielle dans l'évaluation de la résistance des nouveaux chiffrements par bloc.

Dans un premier temps, je donnerai une description de la famille de chiffrement par bloc `SPEEDY` en décrivant notamment la représentation de l'état interne, la fonction de tour ainsi que l'algorithme d'expansion de clef[2]. De plus, je donnerai les revendications de sécurité données par les créateurs de `SPEEDY` pour chacune des versions du chiffrement. L'attaque présentée dans la suite de ce chapitre sera en mesure de contredire ces revendications pour le chiffrement principal de cette famille : `SPEEDY-7-192`.

Le chapitre se poursuit avec une section qui traite de la méthode utilisée pour trouver un distingueur différentiel permettant de monter une attaque intéressante. Dans un premier temps, je mettrai en évidence les propriétés différentielles que présente la SBox de `SPEEDY`. Ensuite, je donnerai une procédure algorithmique qui permet de trouver un distingueur différentiel à haute probabilité en concaténant des distingueur sur un tour. Cela nous a permis d'obtenir un distingueur sur 5.5 tours avec une probabilité $2^{-183.59}$ qui permettra de faire une attaque différentielle sur `SPEEDY-7-192`.

La dernière section de ce chapitre décrit l'attaque différentielle sur `SPEEDY-7-192`. La complexité de cette attaque est $2^{187.28}$ en temps, $2^{187.84}$ en données, et 2^{42} en mémoire, les revendications des créateurs annonçaient 2^{192} en temps et données. Cette attaque contredit donc ces revendications. De plus, les créateurs de `SPEEDY` ont indiqué : l'attaquant ne peut pas ajouter plus d'un tour pour prolonger le distingueur différentiel, ceci est aussi contredit par notre attaque qui étend le distingueur de 1.5 tour. La description de l'attaque différentielle se fait en suivant les trois étapes décrites dans le chapitre 2. Cependant, quelques optimisations sont aussi présentées

- *Compromis entre étapes.* La complexité totale de l'attaque est la somme des complexités de chaque étape. Pour minimiser le tout, il est souvent intéressant de faire des compromis. Ceci est possible notamment en contrôlant la propagation de la différence en entrée du distingueur vers le message clair. En effet, un surcout en

[2]key schedule en anglais

probabilité sera nécessaire, donc une augmentation du coût de l'étape de *création des données*. En revanche, la dimension de l'espace en entrée sera réduite, ainsi, la complexité de l'étape *criblage des paires* sera réduite.

- *Mise à jour des espaces différentiels.* Si la première étape nécessite que les espaces différentiels en entrée et en sortie soient des espaces vectoriels afin d'utiliser les *structures*, il n'est plus nécessaire de considérer ce genre d'espace à partir de la seconde étape. En raffinant ces espaces en ensemble de différences autorisés (qui n'ont plus à être des structures), on obtient un criblage plus puissant, par conséquent la quantité de paires retenues baisse et la complexité se réduit.

- *Association parallèle* Cette dernière optimisation intervient pendant l'exécution de l'algorithme de l'*early abort*. Cette technique décrite et améliorée dans [CNV13a] permet d'accélérer certaines étapes de cet algorithme lorsque certaines conditions sur les relations de propagations sont vérifiées. En effet, cette technique permet de calculer les paires (a, b) qui vérifient une relation R lorsque l'on dispose des elements a dans une liste L_A et des elements b dans une liste L_B en un temps comparable a $|L_A| + |L_B|$ plutôt que $|L_A| \cdot |L_B|$.

Ce chapitre se termine en concluant que les attaques différentielles sont certes très connues, mais menacent toujours les chiffrements par bloc même les plus récents car cette famille d'attaque reste très techniques et difficile à évaluer automatiquement.

Chapitre 5 : Cryptanalyse différentielle-MITM

Dans ce chapitre, je présenterai une nouvelle technique de cryptanalyse en cryptographie symétrique : *cryptanalyse différentielle meet-in-the-middle*. Cette approche consiste à combiner des éléments de cryptanalyse différentielle avec des éléments de cryptanalyse meet-in-the-middle. Du point de vue différentiel, cette nouvelle technique peut être interprétée comme une nouvelle méthode pour effectuer l'étape de récupération de clef. Du point de vue du meet-in-the-middle, les attaques différentielles-MITM consistent en une méthode pour étendre les attaques MITM en ajoutant quelques tours au milieu (similaire à sieve-in-the-middle [CNV13b; CNV13c]).

Dans une première section, je donnerai une description générale de cette technique. Plus précisément, je fournirai un cadre général qui décrit comment monter une attaque différentielle-MITM de manière générique et simple. Ensuite, je montrerai comment combiner cette méthode générique avec deux techniques : le traitement parallèle des partitions de données afin d'en ajouter un tour gratuitement (cette idée hérite des attaques MITM, comme les bicliques, et qui ne s'applique pas aux attaques différentielles classiques), ainsi qu'une technique pour réduire la complexité des données. Enfin nous ferons une comparaison de cette technique avec les attaques différentielles traditionelles et les attaques MITM.

La seconde section de ce chapitre présente une application de la cryptanalyse différentielle MITM au chiifrement par bloc `SKINNY-128-384`. Je commencerai cette section par donner une description de la famille de chiffrement par bloc `SKINNY` puis je décrirai une succession d'attaques différentielles MITM sur `SKINNY-128-384`. La première attaque (23 tours) correspond à l'application de la technique décrite dans la première section. La deuxième (24 tours) reprend l'attaque sur 23 tours et intègre l'optimisation du traitement parallèle des données afin de gagner un tour gratuitement. La troisième attaque sur 25 tours est présentée avec un distingueur différentiel différent de celui utilisé pour les deux premières attaques. Les deux optimisations sur la repartitions des données et sur la complexité des données sont untilisées ici permettant d'ajouter deux tours à la meilleure attaque sur `SKINNY-128-384` en clef simple (en opposition aux clefs reliées).

Pour finir ce chapitre, je donnerai des éléments qui démontrent la puissance des attaques différentielles MITM. En effet, si l'on tente d'effectuer une attaque différentielle traditionelle en utilisant le même distingueur, on ne peut pas espérer attaquer plus de 22 tours si l'on utilise un algorithme de récupération de clef sequentiel et 24 tours dans le cas d'un récupération de clef parallel. Cela soulève alors la question suivante: Dans quel scénarios la cryptanalyse différentielle MITM est elle plus efficace que la cryptanalyse différentielle et MITM ?

Chapitre 6 : Cryptanalyse différentielle impossible quantique

Dans ce chapitre, je présenterai des résultats issus de l'article [DNS22], fruit d'une collaboration avec María Naya-Plasencia et André Schrottenloher. Cet article présente la version quantique de la cryptanalyse différentielle impossible : cryptanalyse différentielle quantique. Cette article réponds à un problème ouvert émis dans [BNS19] qui questionnait l'efficacité des attaques différentielles impossible en quantique.

Dans un premier temps, je présenterai les différents algorithmes quantiques utiles pour la cryptanalyse différentielle impossible. Ces algorithmes de recherche de collisions sont primordiaux pour l'étape de l'attaque qui vise à créer des paires différentielles. Utilisant ces algorithmes, je serai en mesure d'exhiber une formule précise pour la complexité de l'étape de création de paires dans le cas quantique.

Dans un deuxième temps, je présenterai la version quantique de l'algorithme de l'*early abort*. Pour cela, je présenterai d'abord le modèle quantique dans lequel j'évoluerai pour décrire l'attaque. Enfin, je décrirai en détail l'early abort quantique, cet algorithme consiste en une imbrication de recherche de Grover. Aussi, je présenterai une version améliorée lorsque certaines conditions sont sur les propagations sont réunies. Tous les algorithmes seront décrits avec leurs complexités en temps et en mémoire.

Ce chapitre se poursuit avec deux applications : `SKINNY` et `AES`. Pour chacune des applications, il est possible d'observer une amélioration des meilleures attaques quantiques (bien que l'attaque différentielle sur AES présentée dans [BNS19] propose

des complexités similaires).

Conclusion

Pour résumer, ce manuscrit présente un éventail de techniques de cryptanalyse de type différentielle agrémentées d'applications de ces dernières sur des chiffrements par bloc parfois très récents. Cela met donc en exergue l'importance que représente cette famille de cryptanalyse symétrique et impose à tout créateur de primitive de s'assurer de sa résistance vis-à-vis de ces attaques.

Publications

[BDL20] Augustin Bariant, Nicolas David, and Gaëtan Leurent. "Cryptanalysis of Forkciphers". In: *IACR Trans. Symmetric Cryptol.* 2020.1 (2020), pp. 233–265.

[Bei+22] Christof Beierle, Marek Broll, Federico Canale, Nicolas David, Antonio Flórez-Gutiérrez, Gregor Leander, María Naya-Plasencia, and Yosuke Todo. "Improved Differential-Linear Attacks with Applications to ARX Ciphers". In: *J. Cryptol.* 35.4 (2022), p. 29 (cit. on pp. xiii, 31, 36, 38).

[Bou+23a] Christina Boura, Nicolas David, Rachelle Heim Boissier, and María Naya-Plasencia. "Better Steady than Speedy: Full Break of SPEEDY-7-192". In: *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part IV*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14007. Lecture Notes in Computer Science. Springer, 2023, pp. 36–66 (cit. on pp. xiv, 43, 47, 58).

[Bou+23b] Christina Boura, Nicolas David, Patrick Derbez, Gregor Leander, and María Naya-Plasencia. "Differential Meet-In-The-Middle Cryptanalysis". In: *Crypto 2023* (2023), p. 1640 (cit. on pp. 67, 84).

[Bro+22b] Marek Broll, Federico Canale, Nicolas David, Antonio Flórez-Gutiérrez, Gregor Leander, María Naya-Plasencia, and Yosuke Todo. "New Attacks from Old Distinguishers Improved Attacks on Serpent". In: *CT-RSA 2022*. Vol. 13161. LNCS. Springer, 2022, pp. 484–510 (cit. on p. 55).

[DNS22] Nicolas David, María Naya-Plasencia, and André Schrottenloher. "Quantum impossible differential attacks: Applications to AES and SKINNY". In: *Des. Codes Cryptogr.* (2022), p. 754 (cit. on pp. xvi, 87).

Chapter 1 Preliminaries

This chapter's goal is to introduce cryptography, more specifically symmetric cryptography. We will deliberately emphasize the concepts that are relevant to the contributions discussed later in this thesis, except for differential cryptanalysis that is presented later in Chapter 2.

Contents

This chapter is organized as follows: first, we will present cryptology from a historical point of view, then we will describe some constructions from symmetric cryptography followed by cryptanalytic techniques developed to analyze their security. Finally, we will give some elements useful for quantum computing, more precisely we will introduce elements required for quantum cryptanalysis that will be used in Chapter 6.

1.1 History of Cryptology

Cryptology has a rich and ancient history that dates back thousands of years. The earliest known instances of cryptology can be traced to ancient civilizations such as Egypt and Mesopotamia, where methods like transposition and substitution were used to conceal messages. A well-known example of substitution cipher is the *Caesar cipher* that was named after and used by the Roman emperor Julius Ceasar. This type of cipher is now outdated and shows critical weakness to statistical attacks. Throughout

history, cryptologic designs have evolved alongside techniques used to break them. As cryptographers devised new ways to secure information, adversaries countered with innovative approaches to extract information from encrypted messages. This dynamic resulted in the development of increasingly sophisticated methods and the emergence of modern cryptography.

Modern cryptography is built on the foundation of several key principles and techniques. One of these principles is Kerckhoffs' principles [Ker83], which state that the security of a cryptographic system should rely solely on the secrecy of the key, not on the secrecy of the algorithm. This principle emphasizes the importance of key management and assumes that the cryptographic algorithm will be widely known and studied. To answer the need for secure communication between parties, cryptographers design cryptographic protocols that can ensure confidentiality and integrity. These protocols are based on the usage of cryptographic primitives that can be separated into two families: *symmetric cryptology* and *assymetric cryptography*. Symmetric cryptography, also known as secret-key cryptography, involves the use of a single shared secret key for both encryption and decryption. The same key is used by both the sender and the receiver to transform plaintext into ciphertext and vice versa. This type of cryptography is efficient and fast, making it suitable for secure data transmission and storage but requires first to set up a secure key exchange protocol. Asymmetric cryptography, also known as public-key cryptography, employs a pair of mathematically related keys: a public key and a private key. The public key is freely distributed, while the private key is kept secret. Messages encrypted with the public key can only be decrypted with the corresponding private key. This approach enables secure key exchange for symmetric cryptography, digital signatures, and encryption of confidential information.

The development of quantum computers has had a significant impact on cryptography by rendering many of the current asymmetric cryptographic algorithms vulnerable to quantum attacks. Quantum computers can break commonly used encryption methods, such as RSA and elliptic curve cryptography, which rely on the difficulty of certain mathematical problems that can be efficiently solved by quantum algorithms. As a result, the field of post-quantum cryptography is actively exploring new classical algorithms that can resist attacks from (future) quantum computers, ensuring the continued security of sensitive information in the future.

1.2 Symmetric Cryptography

Symmetric cryptography is an approach to encryption that involves the use of a single shared secret key for both encryption and decryption. In this section, we will present the main constructions from symmetric cryptography with a particular focus on *block ciphers*.

1.2.1 Block Ciphers

Block ciphers are one of the main constructions in symmetric cryptography used for encryption, where plaintexts have fixed sizes. To encrypt messages of arbitrary and longer length, it suffices to divide them into fixed-size blocks and encrypted them independently with a special treatment for the last block with padding. Common block cipher algorithms include the Advanced Encryption Standard (AES [DR99]), which operates on 128-bit blocks, and the Data Encryption Standard (DES [Sta+99]), which operates on 64-bit blocks.

Definition 1.1 (Block Cipher)**.** A *block cipher* is a deterministic algorithm that acts on a fixed-length group of bits. It is often used to *encrypt* a plaintext into a ciphertext using a *secret key*. More precisely, a block cipher can be represented as a function $E : \mathbb{F}_2^n \times \mathbb{F}_2^k \to \mathbb{F}_2^n$ such that for any given key $\kappa \in \mathbb{F}_2^k$ the function $E(\cdot, \kappa) : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is a permutation (we will note $E(\cdot, \kappa)$ as E_κ).

Simply meeting the definition above is not enough to make a block cipher secure and interesting. Indeed, some additional requirements must be satisfied regarding both efficiency and security. When the key κ is known, both encryption and decryption operations should be performed very efficiently at a very low cost, hence allowing a large volume of exchange. When the key κ is unknown, the attacker should not be able to extract information from the pairs of plaintext ciphertext. Ideally, the attacker should not be able to distinguish E_κ from a random permutation.

1.2.2 Block Cipher Construction

The construction of a block cipher often consists in the iteration of *rounds* that act on the n-bit blocks that compose the *internal state*. Each round usually applies some key-related transformation to ensure the security of the block cipher. The key material injected in each round is called a *round key* and is computed from the master key κ through a *key schedule* algorithm. This algorithm is expected to be very efficient, in this regard it is oftentimes linear.

Many block ciphers can be described as *key-alternating block ciphers* [DR02]. This construction consists in alternating a key addition with a key independent operation.

Definition 1.2 (Key-alternating block cipher)**.** A block cipher E is a *key-alternating block ciphers* if it can be written as a composition of the following form:

$$E_\kappa = ARK_{\kappa_r} \circ F_r \circ \cdots \circ ARK_{\kappa_1} \circ F_1 \circ ARK_{\kappa_0}$$

where $(F_i)_{1 \leq i \leq r}$ is a family of permutations: *round functions*, ARK_{κ_i} corresponds to the addition of the subkey κ_i to the internal state and $(\kappa_i)_{0 \leq i \leq r}$ is the family of round keys obtained from key schedule applied on the master key κ.

Block ciphers can be constructed in various ways. In practical applications, the block cipher often falls into one of the following categories:

- *Feistel networks.* This family of ciphers was born at IBM in the early 70s and designed by Feistel and Coppersmith. It will later become the basis of the Data Encryption Standard [DH77a]. The internal state of a Feistel network is decomposed into two parts of $n/2$ bits $X = (L, R)$. The round function is based on a *keyed* function $f : \mathbb{F}_2^{n/2} \times \mathbb{F}_2^k \to \mathbb{F}_2^{n/2}$ and consists in applying the following transformation : $(L_{i+1}, R_{i+1}) \leftarrow (R_i, f(R_i, \kappa_i) \oplus L_i$. In essence, the Feistel network is a process that allows the building of a n-bits permutation from a n/2-bits permutation. Some generalizations of this construction exist such as Lay Massey constructions [LM91] and generalized Feistel Networks.

- *Substitution permutation networks (SPN).* The SPN block ciphers have a round function that consists of two operations. The first one is the *substitution layer* and consists of the parallel application of a non-linear function called Sbox. The second one is the *linear layer* and consists of the application of a linear map to the internal state. In the end, the round function consists of the combination of a local complex transformation (substitution layer) with a simpler but global transformation (linear layer). The Advanced Encryption Standard [FIP01] is the most well-known example of an SPN cipher.

- *Add-rotate-XOR (ARX) constructions.* In ARX ciphers, the internal state is represented by dividing the n-bits into n/l registers of the same length l. Three basic operations can be performed: modular addition modulo 2^l, cyclic rotation, and XOR. Since these operations are very simple, the ARX constructions tend to be very efficient. SPECK [Bea+15] is an example of an ARX cipher.

Later in this thesis, we will give precise descriptions of the block ciphers studied in applications (Section 6.4.1, Section 4.1, Section 5.2.1).

1.2.3 Other Constructions

Block cipher is not the only construction in symmetric cryptography, indeed *stream ciphers*, *hash functions* and *Message authentication codes* fulfill different tasks.

Stream Ciphers. Stream ciphers are another class of constructions in symmetric cryptography that encrypt data in a continuous stream, typically one bit or one-word at a time. These ciphers generate a pseudorandom keystream, which is then combined with the plaintext to produce the ciphertext. Stream cipher algorithms are known for their efficiency, making them suitable for applications that require real-time encryption. They are often used when the size of the input cannot be determined in advance.

Hash Functions. Hash functions are cryptographic constructions that transform variable-sized input data into fixed-sized output, commonly referred to as a hash value or message digest. These functions play a crucial role in various cryptographic applications, including integrity checks, password hashing, and digital signatures. Hash functions

have to provide collision resistance and preimage resistance, making them suitable for secure data integrity verification.

Message Authentication Codes (MAC). Message Authentication Codes are constructions that ensure the integrity and the authenticated encryption of a message using a secret key. These codes generate a tag that is appended to the message and can be verified by the recipient using the same key.

1.3 Symmetric Cryptanalysis

In this section, we will present some basic elements of symmetric cryptanalysis. First, we will describe the models for block ciphers in which the attacker is considered to act and then we will present some well-known cryptanalysis families used to break symmetric constructions.

1.3.1 Cryptanalysis Models for Block Ciphers

In cryptanalysis, there are different models in which that characterize the possible actions of the adversary. In the following, we present some of them sorted from the weakest to the strongest.

- *Ciphertext only.* In cyphertext-only attacks, the attacker has access to several encrypted messages without any additional information on the associated plaintext or the encryption key.
- *Known plaintext.* In known-plaintext attacks, the attacker has access to pairs of messages that correspond to a ciphertext and its associated plaintext.
- *Chosen plaintext.* In a chosen-plaintext attack, the attacker has access to an encryption oracle that turns arbitrary plaintext into its corresponding ciphertext.
- *Chosen ciphertext.* In a chosen-ciphertext attack, the attacker has access to a decryption oracle that turns arbitrary ciphertext into its corresponding plaintext.

Adversary objectives can be very different from one case to another. In this section, we will consider the cases where the attacker aims either to build a *distinguisher* algorithm or a *key recovery* algorithm.

1.3.1.1 Distinguisher

Let us consider a block cipher E and suppose that the attacker can query an oracle that takes as input an n-bit message and returns a n-bit ciphertext. The goal of the attacker is to decide whether the oracle consists of the block cipher E using a secret key k (hence a permutation of the form E_k) or if it is a random permutation.

Definition 1.3 (distinguisher)**.** In cryptanalysis, a distinguisher for a cipher E is an algorithm that solves the following problem:

Data: A data set $\mathcal{D}$ composed of pairs $(P_i, C_i)_i$ obtained through oracle calls
Question: Do the pairs correspond to plaintext-ciphertext pairs generated by E or randomly?

1.3.1.2 Key recovery

In this setting, the attacker has access to an encryption oracle using the secret key: E_k. Hence, the attacker can build chosen pairs of plaintext-ciphertext (P, C) and will attempt to extract information on the secret key using them. The goal of the attacker is to recover this key k significantly faster than the *exhaustive search*.

Exhaustive Search. For a given pair of plaintext-ciphertext (P, C), one can perform an exhaustive search to recover the key(Algorithm 1.1). This procedure exists for all ciphers and has a time complexity of $|K|$. In symmetric cryptanalysis an algorithm that manages to recover the key faster than the exhaustive search is an attack.

Algorithm 1.1 Exhaustive Search

$\mathcal{D} \leftarrow \{(P, C) : C = E_k(P)\}$ ▷ Data setup
for $\kappa \in K$ **do**
  if $E(P, \kappa) = C$ for all $(P, C) \in \mathcal{D}$ **then**
    return κ

The efficiency of an attack is measured by three types of complexities:

- *Time complexity.* The time complexity of an attack is the time complexity of the algorithm that describes the attack. Oftentimes, the time complexity unit of measure is the encryption time, since the exhaustive search consists in performing $|K| \cdot |\mathcal{D}|$ encryptions, it facilitates the comparison.
- *Memory complexity.* The memory complexity of an attack is the memory complexity of the algorithm that describes the attack.
- *Data complexity.* The data complexity of an attack is the number of times the attack queries the encryption oracle.

From distinguisher to key recovery. Using distinguishers, it is possible to build key recovery attacks. Indeed, by surrounding a distinguisher for a reduced-round version of the cipher with key recovery rounds, we can recover the key as the good key will be the one implying the non-random property exhibited by the distinguisher as depicted in Section 1.3.1.2. More formally, let us assume we have a data set $\mathcal{D}$ of pairs of plaintext-ciphertext. For each key guess κ (note that we only need to guess the key bits involved in the key recovery rounds), we can compute the value of the internal state at the input and the output of the distinguisher for all the pairs of data set $\mathcal{D}$,

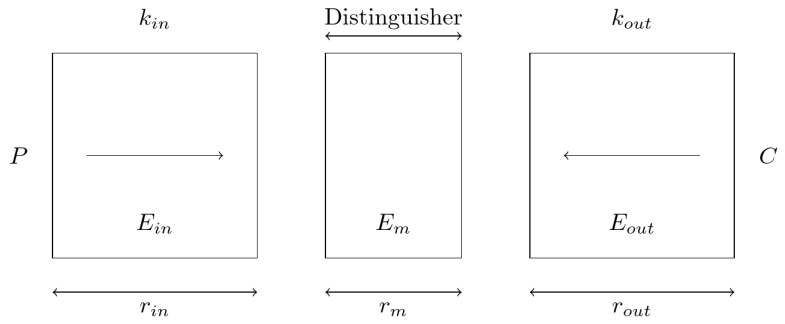


Figure 1.1: Building a key recovery from a distinguisher

we would obtain a data set $\mathcal{D}_\kappa$. Then we can extract the right key guess by applying our distinguisher algorithm on $\mathcal{D}_\kappa$ since only the right key guess implies a non-random property for $\mathcal{D}_\kappa$.

Related key cryptanalysis. Related-key cryptanalysis is a method used to break cryptographic systems by exploiting relationships between keys. It involves analyzing the behavior of a cryptographic algorithm when different keys are used, especially when these keys are related in some known way. By studying the patterns and vulnerabilities that emerge from these relationships, an attacker can potentially recover the secret key or weaken the security of the system though this is a model where attacks have more freedom. This model is particularly useful while analyzing the security of a hash function built on a block cipher (using Merkle Damgard construction [Mer79] for instance).

1.3.2 Cryptanalysis Families

In this section, we will present some well-known cryptanalysis families: Linear cryptanalysis and Meet-the-Middle cryptanalysis. Differential cryptanalysis will be studied later in Chapter 2.

1.3.2.1 Linear Cryptanalysis

Linear cryptanalysis is a powerful method used to analyze and exploit the linearity properties of cryptographic algorithms. It relies on finding linear approximations that hold with high probability, that can be derived into a linear distinguisher and thus allowing attackers to deduce information about the secret key by performing a key recovery attack. Linear cryptanalysis has been successfully applied to various symmetric key algorithms, highlighting the importance of designing ciphers resistant to this type of attack. The two fundamental algorithms in linear cryptanalysis were introduced by Matsui [Mat93] who described an attack on the well-known DES.

Definition 1.4 (Keyed Linear Approximation)**.** Let E be a block cipher, a *keyed linear approximation* of E is a map:

$$\nu : \langle \alpha, P \rangle + \langle \beta, C \rangle + \langle \gamma, k \rangle$$

where P, C denotes a pair of plaintext ciphertext and k the key. The vectors $\alpha \in \mathbb{F}_2^n$ (respectively $\beta \in \mathbb{F}_2^n$, $\gamma \in \mathbb{F}_2^k$) is called the input (respectively output and key) mask.

Definition 1.5 (Bias and Correlation)**.** Let ν be a linear approximation of the block cipher E, its *bias* is the quantity:

$$\epsilon = \mathbb{P}_{P,k}(\nu(P, E_k(P), k) = 0) - \frac{1}{2}.$$

The *correlation* is

$$c = \mathbb{P}_{P,k}(\nu(P, E_k(P), k) = 0) - \mathbb{P}_{P,k}(\nu(P, E_k(P), k) = 1).$$

From the definition, it is possible to see that $c = 2\epsilon$.

Linear Distinguisher. A linear distinguisher consists in exhibiting a linear approximation whose underlying correlation (or bias) is non-negligible. Matsui described some procedures in [Mat93] that allow to construct linear approximations. Indeed, the *piling-up lemma* describes how to combine linear approximations over a small part of a cipher (for instance one round) into an approximation over a larger part of the cipher as explained in Corollary 1.1.

Corollary 1.1. *Let E be a R rounds key alternating block cipher where $(F_i)_{1 \leq i \leq r}$ denotes the family of round functions, $(\kappa_i)_{0 \leq i \leq r}$ denotes the family of round keys and $(X_i)_{0 \leq i \leq r}$ denotes the internal state after round ARK_i (hence X_0 corresponds to the plaintext). For each round, if we have a linear approximation of the form:*

$$\nu_i : \langle \alpha_{i-1}, X_{i-1} \rangle + \langle \alpha_i, X_i \rangle + \langle \alpha_i, \kappa_i \rangle$$

with bias ϵ_i. Then by adding all the linear approximations, we obtain the following approximation on E:

$$\nu : \langle \alpha_0, X_0 \rangle + \langle \alpha_r, X_r \rangle + \sum_{i=1}^{r} \langle \alpha_i, \kappa_i \rangle$$

If the rounds are statistically independent then the bias of ν is $\epsilon = 2^{r-1} \prod_{i=1}^{r} \epsilon_i$.

Later in [Mat94], Matsui also described the *branch and bound algorithm* that looks for a linear approximation with a high correlation. It consists in recursively building linear approximations while incrementing the number of rounds explored.

Linear Key recovery. In [Mat93], Matsui proposed two different algorithms (Algorithm 1.2 and Algorithm 1.3) to perform key recovery based on a linear distinguisher. In both cases, we suppose that the attacker has access to a collection of pairs of plaintext-ciphertext $\mathcal{D} = \{P, C : E_k(P) = C\}$ and knows the existence of a linear approximation $\nu : \langle \alpha, P \rangle + \langle \beta, C \rangle + \langle \gamma, k \rangle$.

In Algorithm 1.2, the attacker computes the counter $T = |\{(P, C) \in \mathcal{D} : \langle \alpha, P \rangle + \langle \beta, C \rangle = 0\}|$ and deduces the value of $\langle \gamma, k \rangle$ based on the value of T and ϵ. This method recovers only one key bit unlike the method developed in Algorithm 1.3 that can recover many more key bits. This second method consists in building a key recovery attack based on a linear distinguisher as explained in the previous section. Hence, we start by splitting the cipher into two parts $E = E_{out} \circ E_{in}$. Without loss of generality, the linear distinguisher ν considered affects the top part: E_{in}. The idea here is to compute $\mathcal{D}_\kappa = \{(P, E_{out}^{-1}(C, \kappa)) : P, C) \in \mathcal{D}\}$ for any possible guess of key κ and check which key guess has the largest empirical bias concerning ν. This key guess is the key used during encryption.

Algorithm 1.2 Matsui's Algorithm 1

$T \leftarrow 0$ ▷ Counter Initialisation
for $(P, C) \in \mathcal{D}$ **do**
  if $\langle \alpha, P \rangle + \langle \beta, C \rangle = 0$ **then**
    $T \leftarrow T + 1$
if $T > N/2$ **then**
  return $\epsilon < 0$ ▷ if true 1 else 0
else
  return $\epsilon > 0$ ▷ if true 1 else 0

Algorithm 1.3 Matsui's Algorithm 2

for $\kappa \in K$ **do**
  $T_\kappa \leftarrow 0$ ▷ Counter Initialisation
  for $(P, C) \in \mathcal{D}$ **do**
    if $\langle \alpha, P \rangle + \langle \beta, E_{out}^{-1}(C, \kappa) \rangle = 0$ **then**
      $T_\kappa \leftarrow T_\kappa + 1$
return $\texttt{argmax}_\kappa(|T_\kappa - N/2|)$

Matsui introduced linear cryptanalysis but this field has a lot of potential and new techniques in linear cryptanalysis still emerge, such as linear hull [Nyb94], multi-dimensional linear cryptanalysis [KR94; BDQ04] and Walsh transform key recovery [Fló22] only to cite a few.

1.3.2.2 Meet-in-the-middle Cryptanalysis

Meet-In-The-Middle attacks were introduced by Diffie and Hellman in 1977 [DH77a]. This widely used technique consists in computing some internal state in the middle of

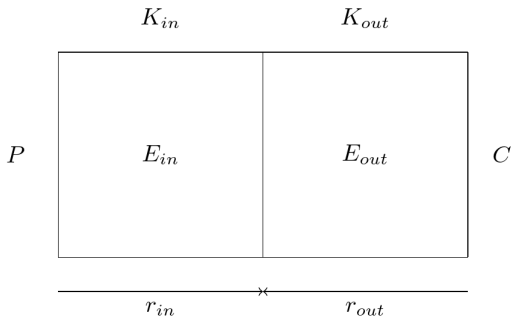


Figure 1.2: Meet-In-The-Middle framework

the cipher from both the input and the output without needing the whole key to perform either of these computations. To recover key bits, the attacker keeps track of the keys leading to a collision in the middle state.

More precisely, let E be an n-bit block cipher with r rounds and let (P, C) denote a pair of plaintext/ciphertexts. We write $E = E_{out} \circ E_{in}$, as in Figure 1.2, where E_{out} and E_{in} have r_{out} and r_{in} rounds respectively ($r = r_{in} + r_{out}$). We denote by K_{in} (respectively K_{out}) the set of key bits required in the computation of E_{in} (respectively E_{out}).

Algorithm 1.4 Meet-In-The-Middle attack

$T \leftarrow$ Empty table ▷ Table Initialisation
$S \leftarrow \emptyset$ ▷ Solution Set Initialisation
for $k_{in} \in K_{in}$ **do**
  $M_{in} \leftarrow E_{in}(P, k_{in})$
  $T[\text{hash}(M_{in})] \leftarrow k_{in}$
for $k_{out} \in K_{out}$ **do**
  $M_{out} \leftarrow E_{out}(C, k_{out})$
  if $T[\text{hash}(M_{out})]$ is not empty **then**
    $S \leftarrow S \cup \{k_{in}, k_{out}\}$
return S

Algorithm 1.4 describes the Meet-In-The-Middle procedure. Since this attack can be performed in the encryption or the decryption direction, without loss of generality, we can suppose that $|K_{in}| \leq |K_{out}|$. The size of the table is set to $|T| = |K_{in}|$, therefore the memory complexity is $\mathcal{M} = |K_{in}|$. Since the algorithm consists of a first loop over K_{in} and a second loop over K_{out}, the time complexity is $\mathcal{T} = |K_{in}| + |K_{out}|$.

The cryptographic community proposed many improvements to this generic attack, such as the technique of guessing some bits of the internal state [DSP07], the all-subkeys approach [Iso11], splice-and-cut [AS08; AS09; Guo+10] and bicliques [KRS12]. In the following, we will present the improvements proposed in [BR10] and [CNV13b]: `partial matching, sieve-in-the-middle`.

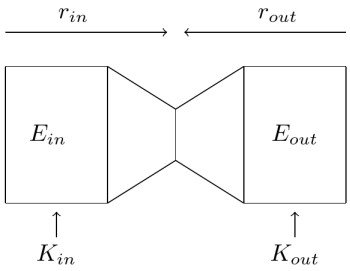


Figure 1.3: Partial Matching framework

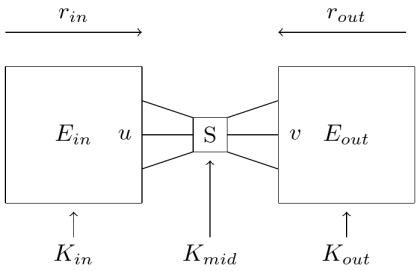


Figure 1.4: Sieve-in-the-middle framework

Partial Matching. Partial matching is a technique that generalizes the meet-in-the-middle technique. Indeed, instead of matching on the whole state in the middle, we will now allow matching on a substate composed of fewer bits (see Figure 1.3). Computing this subspace might involve fewer key bits, hence reducing the size of K_{in} and K_{out}, therefore reducing the overall complexity of the meet-in-the-middle procedure.

Sieve-in-the-middle Sieve-In-The-Middle is a technique that takes partial matching a step further and matches through a non-linear function by sieving noncompatible pairs.

As represented in Figure 1.4, this technique consists first in splitting the cipher into three parts: `input part, middle part, and output part`. The `input part` (respectively `output part`) consists in a subcipher E_{in} (respectively E_{out}). We denote by $\mathcal{U}$ (respectively $\mathcal{V}$) a subspace of $E_{in}(K, P)$ (respectively $E_{out}(K, C)$ and K_{in} (respectively K_{out}) the key bits involved in the computation of the restricted function $E_{in}|_{\mathcal{U}}$ (respectively $E_{out}|_{\mathcal{V}}$). The `middle part` consists of a keyed function S such that $S : K_{mid} \times \mathcal{U} \to \mathcal{V}$.

Now, it is possible to discard every guess (k_{in}, k_{out}) that produces a pair (u, v) that can not match through S. Hence, by precomputing the relation R as follows

$$R(u, v) \Leftrightarrow \exists k_{mid} \in K_{mid} \text{ s.t. } S(k_{mid}, u) = v$$

we can sieve the pairs (u, v), therefore the pairs (k_{in}, k_{out}). The Algorithm 1.5 summarizes the procedure of a Sieve-In-The-Middle attack.

Algorithm 1.5 Sieve-In-The-Middle attack

$L_{in} \leftarrow \emptyset$ ▷ Input Table Initialisation
$L_{out} \leftarrow \emptyset$ ▷ Output Table Initialisation
$L_{sol} \leftarrow \emptyset$ ▷ Soltion Set Initialisation
for $k_{in} \in K_{in}$ **do** ▷ Forward computation
  $u \leftarrow E_{in}|_{\mathcal{U}}(k_{in}, P)$
  $L_{in} \leftarrow L_{in} \cup \{(u, k_{in})\}$
for $k_{out} \in K_{out}$ **do** ▷ Backward computation
  $v \leftarrow E_{out}|_{\mathcal{V}}(k_{out}, C)$
  $L_{out} \leftarrow L_{out} \cup \{(v, k_{out})\}$
Merge L_{in} and L_{out} with respect to R and return the merged list L_{sol} ▷ Merge step
for k such that $(k_{in}, k_{out}) \in L_{sol}$ **do**
  if $E_k(C) = P$ **then**
    return k

1.4 Preliminaries of Quantum Computing

Some asymmetric cryptographic algorithms such as RSA happen to be vulnerable to quantum attacks. It raises the following question: does the development of quantum computers threaten the security of symmetric cryptography? To answer this question, we first need to understand the functioning of a quantum computer and then try to perform cryptanalysis using quantum algorithms to attack the current symmetric primitives.

In this section, we first provide some notions of quantum computing useful in this thesis (Chapter 6): we introduce the quantum circuit model, notions of quantum complexity, quantum-accessible memory, and some of our quantum computing tools. We also define the two quantum attacker scenarios commonly found in the literature.

1.4.1 Quantum Computing Notions

In this thesis, we use the abstract model of *quantum circuits* which is the quantum analogous to the Turing machine.

Definition 1.6 (Quantum State)**.** The *state* of a quantum system is an element of an N-dimensional Hilbert space of norm 1. The states that compose the orthonormal basis of the Hilbert space are referred to as *pure states*: $(|e_i\rangle)_{0 \leq i \leq N-1}$. Therefore, each quantum state can be represented as a superposition over the pure states:

$$|\psi\rangle = \sum_{i=0}^{N-1} \alpha_i |e_i\rangle$$

where $(\alpha_i)_{0 \leq i \leq N-1}$ is a family of complex numbers called *amplitude* such that $\sum_i |\alpha_i|^2 = 1$. The smallest element of quantum information is called *qubit* and corresponds to a quantum state in dimension 2. Any quantum state can be described by a set of qubits.

Proposition 1.1 (Measurement)**.** *Measuring a quantum system* $|psi\rangle = \sum_j \alpha_j |e_j\rangle$ *yields* $|e_j\rangle$ *with probability* $|\alpha_j|^2$*. This action makes the system collapse to* $|e_j\rangle$*.*

If the measurement of a quantum $|\psi\rangle$ state has an effect on another quantum state $|\phi\rangle$, we say that $|\psi\rangle$ and $|\phi\rangle$ are *entangled*. To perform computation on quantum states, we can use *unitary operators*.

Definition 1.7 (Unitary operators)**.** In quantum computing, a *unitary operator* is a matrix $U \in \mathcal{M}^{N,N}(\mathbb{C})$ such that $U^\dagger \times U = I$ where $U^\dagger$ is the conjugate transpose of U. The action of an operator U on a quantum state $|\psi\rangle = \sum_{i=0}^{N-1} \alpha_i |e_i\rangle$ results in a quantum state whose amplitude corresponds to the vector $U \cdot (\alpha_0 \cdots \alpha_{N-1})^t$.

In practice, the operators considered are built from *quantum gates*, analogous to classical logic gates, we refer to [De 19] for a precise description of quantum gates. A quantum circuit consists in applying elementary *quantum gates* to a quantum state. The width of the circuit (number of qubits) is the memory complexity of the algorithm. The depth of the circuit can be thought of as its wall-clock time, and the number of gates as the time complexity since it represents the total number of operations applied.

Quantum attacker scenarios. Following a standard terminology [Kap+16], we define two scenarios. In the *Q1* setting, the attacker has access to a quantum machine for accelerating his computations but has only access to a classical cryptographic oracle: the black box E can only be queried classically. In the *Q2* setting, the attacker can, in addition, encrypt arbitrary quantum states using a *quantum embedding* O_E of E which operates: $|x\rangle |0\rangle \mapsto |x\rangle |E(x)\rangle$.

Quantum memory. We will consider three types of memory: (1) classical memories with classical random access; (2) quantum circuits with large amounts of qubits; (3) large amounts of qubits with *quantum random access*. The latter is named QRAQM in [Kup13]. It can be modeled as the addition, to the quantum circuit model, of a "qRAM gate" (see *e.g.* [Amb07]) which performs the equivalent of a classical memory access, read or write, but in superposition. Implementing a qRAM gate with only standard quantum gates would require a time proportional to the number of qubits in the circuit. The QRAQM model assumes that the qRAM gate is given as an additional basic gate (thus costing quantum time 1, if "time" is taken as the number of gates). This powerful memory model is used to obtain optimal time complexities in many advanced quantum algorithms, e.g., collision finding for the limited birthday problem [Amb07; Bon+23]. The results of this paper, notably the *quantum pair filtering* algorithm of Section 6.2, require large amounts of qubits, but not necessarily QRAQM. We will indicate whether this is the case.

1.4.2 Quantum Search

In Chapter 6, we will use Grover's algorithm [Gro96] and its generalization from [Bra+02], *amplitude amplification*, that we will regroup under the name *quantum search*. In this

Algorithm 1.6 Quantum search (Grover's algorithm).

Input: access to O_f

Output: an n-qubit state such that when measured, the state collapses to x_g with high probability

1: Initialize n qubits in the state $|0\rangle$
2: Apply a Hadamard transform $H^{\otimes n}$
3: **Repeat** t **times**
4: Apply O_f
5: Apply $H^{\otimes n}$
6: Apply O_0
7: Apply $H^{\otimes n}$
8: **EndRepeat**

section, we will describe Grover's search algorithm.

We mostly use the simple setting of Grover search, where we are given a function $f : \{0,1\}^n \to \{0,1\}$ and we want to find y such that $f(y) = 1$. If we have an implementation of the unitary O_f: $|x\rangle \mapsto (-1)^{f(x)} |x\rangle$, and assuming that there is a single solution x_0, we can find it in $\mathcal{O}\left(\sqrt{N}\right)$ calls to O_f instead of $\mathcal{O}(N)$ calls to f classically. In this algorithm, we consider the sets $X = \{x : f(x) = 0\}$ and $Y = \{y : f(y) = 1\}$ and we will work in the space spawned by $|Good\rangle = \frac{1}{\sqrt{|Y|}} \sum_{y \in Y} |y\rangle$ and $|Bad\rangle = \frac{1}{\sqrt{|X|}} \sum_{x \in X} |x\rangle$. The procedure consists of iterating rotations until we obtain a state that is close to $|Good\rangle$ to finally measure it. The quantum operations used here are: the oracle O_f, the Hadamard gate: $H^{\otimes n} : |i\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{j=0}^{n-1} (-1)^{\langle i,j \rangle} |j\rangle$ and the inversion arround 0: $O_0 |x\rangle \mapsto (-1)^{x \neq 0} |x\rangle$.

As depicted in Algorithm 1.6, Grover's algorithm consists of the iteration of the operation $H^{\otimes n} O_0 H^{\otimes n} O_f$ on the initial state $H^{\otimes n} |0\rangle$. This iteration can be interpreted as the composition of $H^{\otimes n} O_0 H^{\otimes n}$ and O_f. Both of these operations are reflections, $H^{\otimes n} O_0 H^{\otimes n}$ is a reflection arround the average since $H^{\otimes n} O_0 H^{\otimes n} = |\psi_0\rangle \langle\psi_0| - I$ (with $|\psi_0\rangle = H^{\otimes n} |0\rangle$) and O_f is a reflection arround $|Bad\rangle$, since $O_f = |Bad\rangle \langle Bad| - I$. Since we are composing two reflections, we obtain a rotation as explained in Lemma 1.1.

Lemma 1.1. *The operation $H^{\otimes n} O_0 H^{\otimes n} O_f$ is a rotation of angle $2\theta = 2 \arcsin \sqrt{\frac{|Y|}{N}}$ in the plane spawn by $|Good\rangle$ and $|Bad\rangle$*

Proof. We can first remark that $|\psi_0\rangle$ can be decomposed on the basis $|Good\rangle$, $|Bad\rangle$ as follows:

$$|\psi_0\rangle = \frac{\sqrt{|Y|}}{\sqrt{N}} |Good\rangle + \frac{\sqrt{|X|}}{\sqrt{N}} |Bad\rangle$$

hence the angle θ between $|\psi_0\rangle$ and $|Bad\rangle$ satisfy $\sin\theta = \frac{\sqrt{|Y|}}{\sqrt{N}}$. As a consequence, the combination of the reflection around $|Bad\rangle$ then $|\psi_0\rangle$ results in a rotation of angle 2θ. □

Theorem 1.1 (Grover's complexity)**.** *A measurement of the state after* $t = \lfloor \frac{\pi}{4 \arcsin \sqrt{\frac{|Y|}{N}}} \rfloor$ *iterations has an outcome* $y \in Y$ *with probability greater that* $1 - \frac{|Y|}{N}$

Proof. The initial state before the iteration of the rotation is $|\psi_0\rangle$ and has an angle θ with $|Bad\rangle$. Hence after t iteration, the quantum state is:

$$|\psi_t\rangle = \sin(2t+1)\theta\,|Good\rangle \cos(2t+1)\theta\,|Bad\rangle\,.$$

Therefore the probability to obtain a bad element (from X) is $\cos^2(2t+1)\theta$. On the other hand we know that $\frac{\pi}{4} - \theta \leq t\theta < \frac{\pi}{4}$, hence $\frac{\pi}{2} - \theta \leq (2t+1)\theta < \frac{\pi}{2} + \theta$ and finally $\cos^2(2t+1)\theta \leq \cos^2 \frac{\pi}{2} - \theta \leq \sin^2\theta = \frac{|Y|}{N}$ □

As a consequence, when $|Y| <<< N$, it is possible to retreive a good element in time $\mathcal{O}(\sqrt{N})$ with an error probability in $\mathcal{O}(1/N)$.

The *amplitude amplification* generalizes Grover since it allows the search space to be defined by a quantum circuit C. Hence, operations $H^{\otimes n}$ are replaced by C and $C^\dagger$. In practice, if we are given the implementation of an algorithm A that succeeds with probability p, the amplitude amplification algorithm allows us to retrieve a successful element in time $\mathcal{O}(\frac{1}{\sqrt{p}})$ as opposed to $\mathcal{O}(\frac{1}{p})$ in the classical setting.

Chapter 2
Differential cryptanalysis

In this chapter, we aim to present the most commonly-used concepts in differential cryptanalysis. All the following chapters are based on elements introduced in the following sections.

Contents

This chapter is organized as follows: first, we will introduce the main cryptanalysis technique studied in this thesis: *Differential Cryptanalysis*. Then, we will describe two of its variants: *Impossible Differential Cryptanalysis* and *Differential-linear Cryptanalysis*.

2.1 Differential Cryptanalysis

Differential cryptanalysis is arguably the most well-known and studied technique in symmetric cryptography. Indeed, in the last 30 years, differential attacks have been applied to analyze a high number of primitives: [BS91a; BS91b; BS92; Bih+01; Wan08; BG11; Can+14; Leu16b], to cite only a few. This technique consists in exploiting the propagation of differences through some part of a cryptographic construction. In the context of block ciphers, the attacker can use differential cryptanalysis to build both distinguishers and key recoveries.

Several refinements and generalizations of the basic technique were introduced together with some new dedicated methods. One can for example mention the technique of truncated differentials [Knu94], the use of structures to reduce data complexity (a technique already introduced in [BS92]), the technique of probabilistic neutral bits [CM16] or the conditional differential attacks [KMN10a]. In addition, differential

cryptanalysis is the basis of newer cryptanalysis techniques such as: boomerang and rectangle cryptanalysis [Wag99], impossible differential cryptanalysis [Knu98; BBS99], differential-linear cryptanalysis [LH94] etc.

2.1.1 Differential Dinstinguisher

The purpose of this section is to provide a definition for differential distinguishers as well as give insight on the process constructing them. To complete this task, we will begin by presenting some preliminary definitions needed for describing differential cryptanalysis.

Definition 2.1 (Differential)**.** Let $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a boolean function. A *differential* is a pair of input and output differences $(\Delta_{in} \xrightarrow{f} \Delta_{out}) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$. We call differential probability of $(\Delta_{in} \xrightarrow{f} \Delta_{out})$ the following quantity:

$$DP(\Delta_{in} \xrightarrow{f} \Delta_{out}) = \mathbb{P}_{x \in \mathbb{F}_2^n} \left[f(x \oplus \Delta_{in}) \oplus f(x) = \Delta_{out} \right].$$

For a given block cipher E, any differential $(\Delta_{in} \xrightarrow{E} \Delta_{out})$ with an underlying probability significantly higher than what we would get with a random permutation can be exploited to build a differential distinguisher. Indeed we can distinguish the cipher from a random permutation by empirically computing the differential $(\Delta_{in} \xrightarrow{E} \Delta_{out})$ as depicted in Algorithm 2.1.

Algorithm 2.1 Differential distinguisher

Require: $(\Delta_{in}, \Delta_{out})$ a differential for f, $\mathcal{X} \subset \mathbb{F}_2^n$ set of $(DP(\Delta_{in} \xrightarrow{f} \Delta_{out}))^{-1}$ elements

```
for x ∈ 𝒳 do
    Δ ← f(x) ⊕ f(x ⊕ Δ_in)
    if Δ = Δ_out then
        return Cipher
return Random permutation
```

In practice, directly computing a differential for a whole cipher often is computationally expensive. However, a differential can be computed for smaller boolean functions that compose the round function of the cipher by exhaustively computing its differential table.

Definition 2.2 (Differential table)**.** Let $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a boolean function, we call *differential table* of f the table $\mathcal{T}$ that verifies:

$$T[i][j] = |\{x : f(x) \oplus f(x \oplus i) = j\}|$$

To understand how we can derive a differential for a cipher using a differential of its inner parts, we introduce the following definition.

Definition 2.3 (Differential Characteristic)**.** Let's consider a family of r functions $(f_i)_{1\leq i\leq r} : \mathbb{F}_2^n \to \mathbb{F}_2^n$ and let F denote the concatenation of this family: $F = f_r \circ \cdots \circ f_1$. A differential characteristic is a family of $r+1$ differences $(\delta_0 \xrightarrow{f_1} \delta_1 \xrightarrow{f_2} \cdots \xrightarrow{f_r} \delta_r) \in (\mathbb{F}_2^n)^{r+1}$. We call differential probability of the differential characteristic the following quantity::

$$DP((\delta_0 \xrightarrow{f_1} \delta_1 \xrightarrow{f_2} \cdots \xrightarrow{f_r} \delta_r)) = \mathbb{P}_{x\in\mathbb{F}_2^n}\left(\cap_{i=1}^{r}\left[f_i \circ \cdots \circ f_1(x \oplus \delta_0) \oplus f_i \circ \cdots \circ f_1(x) = \delta_i\right]\right).$$

Under some independence assumption between each function f_i, the overall differential probability of the differential characteristic can be decomposed as the product of the differential probability of all its composing functions:

$$DP(\delta_0 \xrightarrow{f_1} \delta_1 \xrightarrow{f_2} \cdots \xrightarrow{f_r} \delta_r) = \prod_{i=0}^{r-1} DP(\delta_i \xrightarrow{f_{i+1}} \delta_{i+1})$$

In the general case, there is no reason that the random variable X_i (that represents the internal state associated to δ_i) is independent to some X_j for $j \neq i$. However in the case of key-alternating block cipher, the round function features a key addition layer that brings randomness to the internal state and oftentimes, we can derive the independent hypothesis from this addition. Nonetheless, if the key addition brings does not randomness on the whole state (for instance the key addition does not all concern the bits of the internal state) then the independence hypothesis might not hold.

In practice, it is possible to derive a differential characteristic for some rounds of a block cipher by concatenating 1-round differentials (as it is the case in Chapter 4). With a large collection of differential characteristics that coincide both in input and output difference, we can derive the probability of the differential for the whole cipher.

Property 2.1. *Let's consider a family of r functions $(f_i)_{1\leq i\leq r} : \mathbb{F}_2^n \to \mathbb{F}_2^n$ and $F = f_r \circ \cdots \circ f_1$. For a differential $(\Delta_{in} \xrightarrow{F} \Delta_{out})$, the differential probability is:*

$$DP(\Delta_{in} \xrightarrow{F} \Delta_{out}) = \sum_{\delta_1,\cdots,\delta_{r-1}\in\mathbb{F}_2^n} DP(\Delta_{in} \xrightarrow{f_1} \delta_1 \xrightarrow{f_2} \cdots \xrightarrow{f_r} \Delta_{out})$$

In practice, computing all the differential characteristics associated to a differential is infeasible as soon as more than a few rounds are considered. However, most of them have negligible probability compared to the main one. Hence by considering a reasonable subset of differential characteristics, we can expect to get a reasonable approximation of the value of the differential probability.

2.1.2 Differential Attack

In this section, we follow the framework described in Section 1.3.1.2 and show how to build a key recovery attack using a differential distinguisher. We will begin by extending the differential considered here into key recovery rounds by using truncated differentials.

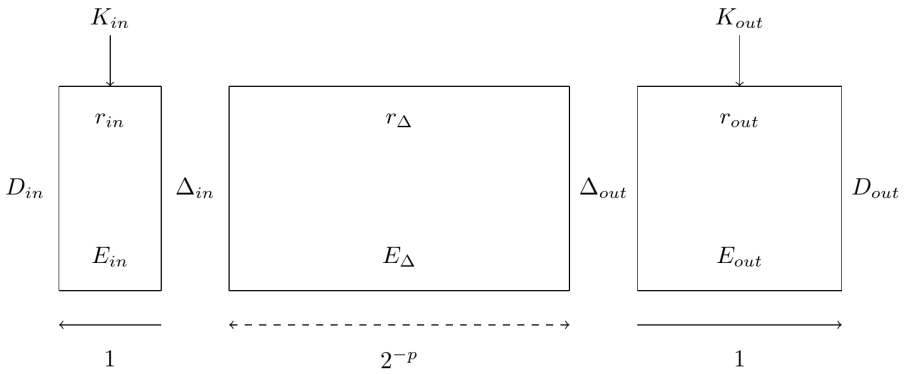


Figure 2.1: Extended differential distinguisher with key recovery rounds

Definition 2.4 (Truncated Differential)**.** Let $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a boolean function. A truncated differential is a pair of differences subset $(\mathcal{D}_{in}, \mathcal{D}_{out}) \subset \mathbb{F}_2^n \times \mathbb{F}_2^n$. The differential probability associated to this truncated differential is:

$$DP(\mathcal{D}_{in} \xrightarrow{f} \mathcal{D}_{out}) = \mathbb{P}_{x \oplus y \in \mathcal{D}_{in}} \left[f(x) \oplus f(y) \in \mathcal{D}_{out} \right]$$

We start by considering a cipher E that is the composition of three subciphers $E = E_{out} \circ E_\Delta \circ E_{in}$ such that E_{in} (respectively E_Δ, E_{out}) consists of r_{in} rounds (respectively r_Δ, r_{out}). Let's now suppose that $(\Delta_{in} \xrightarrow{E_\Delta} \Delta_{out})$ is a differential of probability 2^{-p}. Extending the differential with truncated probability 1 in both directions consists in considering two truncated differentials: $(\{\Delta_{in}\} \xrightarrow{E_{in}^{-1}} D_{in})$ and $(\{\Delta_{out}\} \xrightarrow{E_{out}} D_{out})$. D_{in} and D_{out} often consist of two linear subspaces of $\mathbb{F}_2^n$ derived from subsets of active bits. We denote by d_{in} and d_{out} the quantities such that $|D_{in}| = 2^{d_{in}}$ and $|D_{out}| = 2^{d_{out}}$. Let K_{in} denote the space of subkeys involved in E_{in}, typically a subset of information bits of the key, and K_{out} be the space of subkeys involved in E_{out}. Finally, let $K_{in \cup out}$ be the space of subkeys involved both in E_{in} and E_{out}. In general $K_{in \cup out}$ is smaller than $K_{in} \times K_{out}$ due to key-schedule relations.

2.1.2.1 Data Generation

The first step of the attack is called: Data Generation and consists in making calls to the encryption oracle. Without loss of generality, we can assume that the attack is done in the encryption way (rather than the decryption way) since it suffices to consider E^{-1} instead of E. In this section, we will assume that D_{in} is a subspace of $\mathbb{F}_2^n$ of dimension d_{in}.

Definition 2.5 (Structure)**.** A structure $S \subset \mathbb{F}_2^n$ associated to D_{in} is a set of plaintexts such that:

$$x, y \in S \times S \Rightarrow x \oplus y \in D_{in}$$

A structure is an affine subset of $\mathbb{F}_2^n$ of dimension $\dim(S)$. Therefore, it is possible to build $2^{2\dim(S)}$ pairs within a structure. When omitting the repetitions, we obtain a total of $2^{2\dim(S)-1}$ pairs.

To estimate the amount of data required to mount the attack, we consider the overall differential probability.

$$DP(D_{in} \xrightarrow{E_{in}} \{\Delta_{in}\}) \cdot DP(\Delta_{in} \xrightarrow{E_{\Delta}} \Delta_{out}) \cdot DP(\Delta_{out} \xrightarrow{E_{out}} \{D_{out}\})$$

The probability $DP(D_{in} \xrightarrow{E_{in}} \{\Delta_{in}\}) = 2^{-c_{in}}$ can be characterized by the number of conditions c_{in} that we need to verify (the number of conditions c_{out} can also be defined similarly). In practice, we often remark that $c_{in} = d_{in}$, in this case, the overall probability is $2^{-p-d_{in}}$. Therefore, we need to generate $2^{p+d_{in}}$ pairs by using 2^s structures of size at most $2^{d_{in}}$. Therefore,

$$\begin{aligned} p + d_{in} =& s + 2\dim(S) - 1 \\ \Leftrightarrow s =& p + d_{in} - 2\dim(S) + 1. \end{aligned}$$

At that point, we need to consider two scenarios:

- $p \geq d_{in} - 1$. In this case, one structure does not contain enough data to mount our attack. Hence, we need to consider $2^{p-d_{in}+1}$ different structures of size $2^{d_{in}}$ to mount our attack. The complexity of this first step consists in the product of the number of structures considered times the size of these structures: $\mathcal{C}_{Data} = 2^{p+1}$.
- $p < d_{in} - 1$. In this case, considering a structure of size $2^{d_{in}}$ would generate more data than what is needed. Hence, it is enough to consider one resized structure of size $2^{\frac{p+d_{in}+1}{2}}$ to mount the attack. The complexity of this first step consists of the size of the structure considered: $\mathcal{C}_{Data} = 2^{\frac{p+d_{in}+1}{2}}$.

Theorem 2.1 ([BS91a])**.** *The complexity of the Data Generation step of a differential attack is given by the following formula:*

$$\mathcal{C}_{Data} = \max\left(2^{p+1}, \min_{d \in \{d_{in}, d_{out}\}} 2^{\frac{p+1+d}{2}}\right)$$

This theorem can be derived from the study of the two scenarios depicted above. The min over d_{in} and d_{out} appears since it is possible to consider the cipher in the encryption or the decryption direction depending on what is the most interesting. The memory complexity of a differential attack consists of the size of the structure considered. Indeed, all the rest of the attack can be performed on the fly.

2.1.2.2 Pair Sieving

We now describe the second step of differential cryptanalysis: Pair Sieving. The goal of this step is to build differential pairs whose plaintext difference is in D_{in} and ciphertext

difference is in D_{out}. This step follows the Data generation step in which data was generated using structures following D_{in}. Hence we already know that two plaintexts will follow D_{in} if and only if they belong to the same structure. Therefore, we store the data in a table and sort them according to their ciphertext value. Hence, we can efficiently check if two elements only differ on the d_{out} bits by checking that there is no difference on the $n - d_{out}$ other bits.

To estimate the number of differential pairs that we will be able to build, we consider the restriction of E to the complementary set of D_{out}: $\mathbb{F}_2^n \setminus D_{out}$. Since we expect this restriction to behave like a random function, we can state that:

$$\begin{aligned}\mathbb{P}\left[E(x) \oplus E(x \oplus \delta) \in D_{out}\right] &= \mathbb{P}\left[E|_{\mathbb{F}_2^n \setminus D_{out}}(x) = E|_{\mathbb{F}_2^n \setminus D_{out}}(x \oplus \delta)\right] \\ &= 2^{-(n-d_{out})}\end{aligned}$$

Therefore, from the $2^{p+d_{in}}$ possible pairs generated from the first step, we keep a $2^{-(n-d_{out})}$ proportion of them. Hence, we are left with $N = 2^{p+d_{in}+d_{out}-n}$ differential pairs, that we store in a table $\mathcal{T}_0$.

Some more optimizations could be performed at this stage of the attack, an example is given in Section 4. Indeed, we could refine the sets D_{in} and D_{out} to reduce their size, hence reducing the number of differential pairs. Note that this size update cannot be done earlier since we want our sets D_{in} and D_{out} to follow a certain shape so that we can use structures based on them to efficiently perform the data generation step.

2.1.2.3 Key Recovery

We now move on to the final part of the differential cryptanalysis: the Key Recovery. The main idea of this step is to perform some key guesses (sequentially or in parallel) to perform a partial encryption through E_{in} and E_{out} by performing key guesses on $K_{in \cup out}$. Then we expect to witness a pair that satisfies the differential if and only if the key guesses performed are right. By following the framework presented in Section 1.3.1.2 to build a key recovery attack from a distinguisher, we would obtain Algorithm 2.2. In the context of differential cryptanalysis, we can recover the key more efficient techniques as shown in []. Such a technique is called the early abort and was introduced in [].

Early abort. In this technique, we want to consider all propagation conditions (ex: S-box conditions, MixColumns conditions, etc.) that allow a pair to propagate through the truncated differential from the plaintext (or ciphertext) to the differential in the middle. We will denote by $\mathcal{L}_c$ the list of these conditions. For each condition $c \in \mathcal{L}_c$, we denote by $\mathcal{K}(c)$ the set of keys required to verify the propagation of this condition. In Algorithm 2.3, we give a high-level description of how early abort should proceed.

In essence, we want to consider sequentially all the conditions $c \in \mathcal{L}_c$. Then, we guess the key bits that remain unknown to check whether the condition c is verified for each element stored in data. Then we keep the elements that verify the condition together with the associated key guess, the other elements are discarded. In the end, we

Algorithm 2.2 Naive Key Recovery

Require: $\mathcal{P}$ a set of differential pairs

for $\kappa_\cap \in K_\cap$ **do**
  for $\kappa_{in} \in K_{in} \setminus K_\cap$ **do**
    for $\kappa_{out} \in K_{out} \setminus K_\cap$ **do**
      for $(p_1, p_2) \in \mathcal{P}$ **do**
        $\delta_{in} \leftarrow E_{in}(p_1, \kappa_\cap, \kappa_{in}) \oplus E_{in}(p_2, \kappa_\cap, \kappa_{in})$
        $\delta_{out} \leftarrow E^{-1}_{out}(c_1, \kappa_\cap, \kappa_{output}) \oplus E^{-1}_{out}(c_2, \kappa_\cap, \kappa_{out})$
        if $\delta_{in} = \Delta_{in}$ and $\delta_{out} = \Delta_{out}$ **then**
          return $\kappa_\cap, \kappa_{in}, \kappa_{out}$

Algorithm 2.3 Sequential Early Abort

Require: $\mathcal{P}$ a set of differential pairs, $\mathcal{L}_c$ a list of condition

$L_k \leftarrow \emptyset$ ▷ Set of known keys
$\mathcal{D} \leftarrow \mathcal{P}$ ▷ Initialization of our data
for $c \in \mathcal{L}_c$ **do**
  $\mathcal{K} \leftarrow \mathcal{K}(c) \setminus L_k$ ▷ Set of key bits we do not know yet
  $\mathcal{D}' \leftarrow \emptyset$ ▷ Initialization of our data
  for $d \in \mathcal{D}$ **do**
    for κ a guess of $\mathcal{K}$ **do**
      if (d, κ) verify the condition c **then**
        $\mathcal{D}' \leftarrow \mathcal{D}' \cup \{(d, \kappa)\}$
  $\mathcal{D} \leftarrow \mathcal{D}'$
return $\mathcal{D}$

expect $\mathcal{D}$ to be a singleton of the form (p, κ) where $\kappa \in K_{in} \cup K_{out}$ is the subkey used for the encryption.

Regarding the complexity, let g_i denote the number of bits that remain to be guessed for the i-th condition and 2^{f_i} denote the proportion of data discarded when verifying the i-th condition.

$$\begin{aligned}\mathcal{C}_{KR} &= |\mathcal{P}| \cdot \left[2^{g_0} + 2^{g_0 - f_0}(2^{g_1} + 2^{g_1 - f_1}(\cdots(2^{g_{|\mathcal{L}_c|-1}})\cdots))\right] \\ &= |\mathcal{P}| \cdot \sum_{i=0}^{|\mathcal{L}_c|-1} 2^{g_i} \cdot 2^{\sum_{j=0}^{i-1} g_i - f_i}\end{aligned}$$

When looking at the formula for the complexity of early abort, it is possible to notice that the order in which the conditions of $\mathcal{L}_c$ are considered has an impact on this complexity. Finding the optimal order is a challenging task and is still an active domain of research. In addition, some other techniques allow to perform parallel key guessing, hence resulting in a speedup for the complexity of the key recovery (see example in Section 4.3).

To summarize this section, we gather in the following theorem the complexities associated to differential cryptanalysis

Theorem 2.2 (Differential Cryptanalysis Complexity)**.** *The complexities of a differential attack are:*

- Data. *The data complexity corresponds to the number of calls to the encryption oracle done in the* pair generation *step.*

$$\mathcal{C}_{Data} = \max\left(2^{p+1}, \min_{d\in\{d_{in},d_{out}\}} 2^{\frac{p+1+d}{2}}\right)$$

- Memory. *The memory complexity corresponds to the size of the structure used in the attack.*

$$\mathcal{C}_{Memory} = \min\left(\min_{d\in\{d_{in},d_{out}\}} 2^{d}, \min_{d\in\{d_{in},d_{out}\}} 2^{\frac{p+1+d}{2}}\right)$$

- Time. *The time complexity of the attack is the sum of the time complexities of the three steps.*

$$\mathcal{C}_{Time} = \mathcal{C}_{Data} + 2^{p+d_{in}+d_{out}-n} + 2^{p+d_{in}+d_{out}-n} \cdot \mathcal{C}_{KR}$$

where C_{KR} denotes the complexity of the key recovery for one *pair.*

2.2 Impossible Differential Cryptanalysis

In this section, we provide a generic depiction of impossible attacks, a variation of differential attacks that features a differential distinguisher with probability zero.

2.2.1 Principle

Impossible differential attacks were independently introduced by Knudsen [Knu98] and Biham, Biryukov, and Shamir [BBS05]. The goal of this cryptanalysis technique is to recover some bits of the secret key of a black-box encryption oracle. This is done by discarding all the wrong key guesses, with the help of a pair of plaintexts that leads to an impossible pattern under its partial encryption with the wrong key guesses.

An impossible differential attack is built similarly to a differential attack. The differential distinguisher is replaced by an impossible differential, that is, a pair of differentials $\Delta_{in}, \Delta_{out}$ such that the probability that Δ_{in} propagates to Δ_{out} after r_Δ rounds is 0. Then, using truncated differentials, we will append r_{in} and r_{out} rounds of the cipher respectively before and after the impossible differential.

Assume that, we found a pair of plaintexts $p = (x, y)$ such that $x \oplus y \in D_{in}$ and $E_k(x) \oplus E_k(y) \in D_{out}$. Then, any key guess that leads to the impossible differential can be discarded, that is any key candidate κ that verifies:

$$\left(E_{in}(\kappa)(x) \oplus E_{in}(\kappa)(y) = \Delta_{in}) \wedge (E^{-1}_{out}(\kappa)(E_k(x)) \oplus E^{-1}_{out}(\kappa)(E_k(y)) = \Delta_{out}\right) \ .$$

The goal of the attack is to discard as many keys as possible using many plaintext-ciphertext pairs. Similarly to differential attacks, impossible differential attacks can be represented in a three-step procedure: *Data generation, Pair Sieving, Key Recovery.*

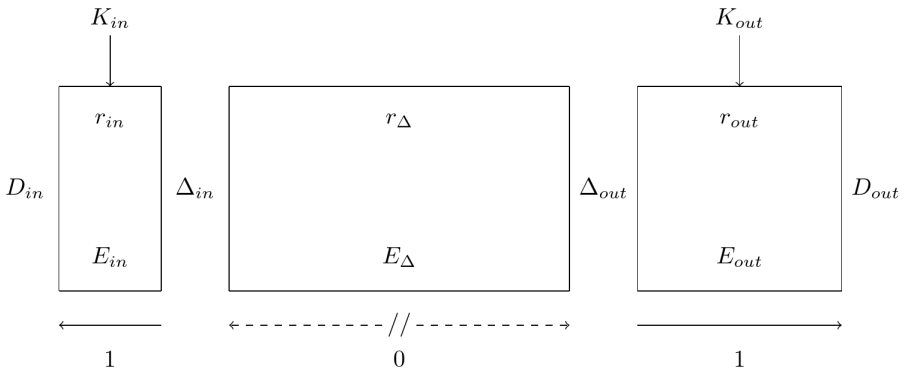


Figure 2.2: Generic presentation of an impossible differential attack, with the notations used in this paper. The differential $\Delta_{in} \leftrightarrow \Delta_{out}$ through the middle rounds E_{imp} is impossible.

2.2.2 Complexity

In this section, we provide both the data complexity and the time complexity of the impossible attack previously described in [BNS14].

Number of Pairs. In impossible differential attacks, the data complexity is determined by the number N of pairs in the table $\mathcal{T}_0$.

Considering a differential pair with input difference in D_{in} and output difference in D_{out}, we denote by c_{in} the number of bit conditions for a pair to propagate to a difference of Δ_{in} at round r_{in}, and c_{out} the number of bit conditions to reach a difference of Δ_{out} at round $r_{in} + r_\Delta$. Hence, a pair propagates to the middle part both from the plaintext and the ciphertext with probability $2^{-(c_{in}+c_{out})}$. A given trial key $k \in K_{in \cup out}$ is kept among the candidate keys with probability

$$P = \left(1 - 2^{-(c_{in}+c_{out})}\right)^N \quad .$$

By applying log to both sides, we obtain

$$N = \mathcal{O}\left(\log \frac{1}{P} \cdot 2^{c_{in}+c_{out}}\right) \quad .$$

Different trade-offs are therefore possible between P and N. A popular strategy, generally used by default is to choose N such that only the right key is left after the sieving procedure. This means $P < \frac{1}{|K_{in \cup out}|}$ therefore $N = \mathcal{O}(2^{c_{in}+c_{out}} \cdot |K_{in \cup out}|)$. Another way consists in taking $P = \frac{1}{2}$ which removes half of the candidate keys.

Time Complexity. The time complexity can be decomposed as the sum of the time complexity of the three steps. For each of these steps, we describe de time complexity associated.

Data Generation. In this step, we aim at generating enough data using *structures* to recover N pairs after the Pair Sieving step. The complexity of this problem was studied in [GP10] for the special case where $N = 1$:

$$C_1 = \max\left(\min_{d\in\{d_{in},d_{out}\}} \sqrt{2^{n+1-d}}, 2^{n+1-(d_{in}+d_{out})}\right) .$$

This is optimal if E behaves like a random permutation, as shown in [HNS20]. A naive way to extend to N pairs would be to use this technique N times and get a complexity of $N \cdot C_1$. However, as shown in [BNS14], with access to encryption and decryption oracles, it is possible to reduce this complexity to:

$$C_N = \max\left(\min_{d\in\{d_{in},d_{out}\}} \sqrt{N2^{n+1-d}}, N2^{n+1-(d_{in}+d_{out})}\right) .$$

Pair Sieving. This step is strait forward in this type of attack since it consists in gathering the N pairs using the data generated before. Hence its complexity both in time and memory is N.

Key Recovery. The key recovery of impossible differential attacks can also be performed using *early abort*. However, in the context of impossible differential attacks, we will present another description of the key recovery algorithm since it suits better the transition towards quantum impossible differential attacks presented in Chapter 6.

We formalize the key recovery step using *test functions*. Let us assume that $K_{in\cup out}$ can be decomposed as $K_{in\cup out} = K_1 \times K_2 \times \cdots \times K_\ell$, where K_1, ..., K_ℓ typically represent choices for some bits of the subkeyspace. Together with this decomposition, we will have ℓ test functions:

$$T_i : \begin{cases} \mathcal{T}_0 \times K_1 \times \ldots \times K_i \to & \{0,1\} \\ (p \ , \ k_1 \ , \ \ldots \ , \ k_i) \mapsto & T_i(p, k_1, \ldots, k_i) \end{cases} .$$

The test functions formalize the concept of condition presented in Section 2.1.2.3. Indeed each test function T_i takes some part of the subkey, some part of the pair, and checks whether they meet some condition. Typically, we start from the differences D_{in} and D_{out} and compute partially the first and last rounds; the successive T_i check that the partial encryption and decryption of the pair satisfies a truncated differential pattern that ultimately leads to the differential $(\Delta_{in} \xrightarrow{E_\Delta} \Delta_{out})$ at rounds r_{in} and r_{out}.

Next, we define the set of pairs satisfying all the T_i:

$$\mathcal{T}_\ell(k) = \{p \in \mathcal{T}_0 \mid \forall i, T_i(p, k_1, \ldots, k_i) = 1\} .$$

Thus, the test functions are defined so that: $\mathcal{T}_\ell(k) \neq \emptyset$ if and only if, there exists a given pair $p \in \mathcal{T}_0$ such that k makes the differential appear for p. The computation of $\mathcal{T}_\ell(k)$ thus yields a procedure exhibiting the encryption key.

Impossible differential early abort. We describe generically the *early-abort* procedure in the context of impossible differential cryptanalysis. Given an ordering of the test functions, the early-abort strategy relies on the definition of *intermediate tables*: we do not compute directly $\mathcal{T}_\ell(k)$ for all key guesses k, and instead, we compute tables $\mathcal{T}_i$ which depend on some partial key guess and are partially filtered with respect to a subset of the test functions.

Definition 2.6 (Intermediate Tables)**.** Given a partial key guess $(k_1, \ldots, k_i) \in K_1 \times \ldots \times K_i$, given the initial table $\mathcal{T}_0$ of pairs, the *intermediate table of* $(k_1, \ldots, k_i)$ contains all the pairs satisfying the test functions which can be computed from $(k_1, \ldots, k_i)$:

$$\mathcal{T}_i(k_1, \ldots, k_i) := \{p \in \mathcal{T}_0, T_1(p, k_1) = 1, T_2(p, k_1, k_2) = 1, \ldots, T_i(p, k_1, \ldots, k_i) = 1\} \ .$$

The early-abort technique (Algorithm 2.4) relies on the inclusion of intermediate tables: $\forall i \geq 1, \mathcal{T}_i(k_1, \ldots, k_i) \subseteq \mathcal{T}_{i-1}(k_1, \ldots, k_{i-1})$. This allows us to exhibit the complete guesses $(k_1, \ldots, k_\ell)$ such that $\mathcal{T}_\ell(k_1, \ldots, k_\ell) \neq \emptyset$ by looping over the guesses k_i and backtracking to previously computed tables, instead of recomputing them from the start.

Algorithm 2.4 Early Abort for Impossible Differential Attacks.

Input: a table $\mathcal{T}_0$ for the pairs
Output: finds the only key k for which there is no invalidating pair

1: **for all** $k_1 \in K_1$ **do**
2: Build $\mathcal{T}_1(k_1) = \{p \in \mathcal{T}_0, T_1(p, k_1) = 1\}$
3: **for all** $k_2 \in K_2$ **do**
4: Build $\mathcal{T}_1(k_1) = \{p \in \mathcal{T}_1, T_2(p, k_1, k_2) = 1\}$
...
5: **for all** $k_\ell \in K_\ell$ **do**
6: Sieve the table $\mathcal{T}_{\ell-1}(k_1, \ldots, k_{\ell-1})$ according to T_ℓ, obtain $\mathcal{T}_\ell$
7: If $\mathcal{T}_\ell = \emptyset$, then return $(k_1, \ldots, k_\ell)$
...

For a given sequence of keys $k_1, \ldots, k_i$, we define:

$$\sigma_i = \mathbb{P}_{p \in \mathcal{T}_0} \left[T_i(p, k_1, \ldots, k_i) = 1\right] \ .$$

This quantity σ_i corresponds to the proportion of data that is filtered at each step, it corresponds to the exponential version of f_i presented in Section 2.1.2.3 (i.e. $\sigma_i = 2^{-f_i}$). In general, σ_i can depend on the choice of the key, but in practice, the values are similar, and we can reason with the expectancy of σ_i over all key choices ($\sigma_i \simeq \mathbb{E}[k](\sigma_i(k))$). Then the sizes of the tables $\mathcal{T}_0, \ldots, \mathcal{T}_{\ell-1}$ are $N, \sigma_1 N, \ldots, (\prod_{i=1}^{\ell-1} \sigma_i)N$. Assuming that each new table is built by enumerating the previous table, the time complexity of Algorithm 2.4,

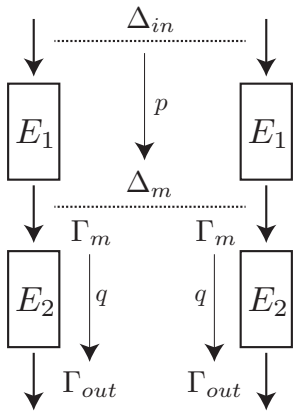


Figure 2.3: The structure of a differential-linear distinguisher.

hence the complexity of the key recovery step can be written as:

$$|K_1|\left(\underbrace{N}_{\text{Build } \mathcal{T}_1} + |K_2|\left(\sigma_1 N + |K_3|\left(\sigma_1\sigma_2 N + \ldots + |K_\ell|\left(\prod_{i=1}^{\ell-1}\sigma_i\right)N\right)\right)\right)$$
$$= N\left(|K_1| + \sigma_1|K_1||K_2| + \sigma_1\sigma_2|K_1||K_2||K_3| + \ldots + \prod_{i=1}^{\ell-1}\sigma_i\prod_{i=1}^{\ell}|K_i|\right)$$

2.3 Differential-linear Cryptanalysis

Differential-Linear cryptanalysis was introduced in 1994 by Langford and Hellman [LH94]. This technique consists in building a distinguisher by combining a differential and a linear approximation. In the original work, the differential part was reduced to the cases of probability 1, and a generalization to other probabilities was given by Biham, Dunkelman, and Keller in [BDK02]. This technique is particularly useful when both correlations of linear approximations and probabilities of differential approximation decrease very fast when the number of round increase. Indeed, by combining a linear and a differential approximation allow, it is possible to derive a distinguisher on some large amount of rounds without witnessing a drastic reduction of the underlying correlation.

Differential-linear cryptanalysis is built using both differential and linear properties of the cipher. More precisely, suppose that the considered cipher E can be split into $E = E_2 \circ E_1$ (as depicted in Figure 2.3) such that:

- a differential distinguisher is applied to E_1

$$\mathbb{P}_{x\in\mathbb{F}_2^n}\left[E_1(x) \oplus E_1(x \oplus \Delta_{in}) = \Delta_m\right] = p$$

- a linear distinguisher is applied to E_2

$$\mathrm{cor}_{y\in\mathbb{F}_2^n}\left[\langle\Gamma_m, y\rangle \oplus \langle\Gamma_{out}, E_2(y)\rangle\right] = q$$

Then, under the assumption that $E_1(X)$ and $E_2(X)$ are independent random variables, the following differential-linear distinguisher holds

$$cor_{x\in\mathbb{F}_2^n}\left[\langle\Gamma_{out}, E(x)\rangle \oplus \langle\Gamma_{out}, E(x\oplus\Delta_{in})\rangle\right] = pq^2$$

Proof. If we assume that $\langle\Gamma_m, y\rangle \oplus \langle\Gamma_{out}, E_2(y)\rangle$ and $\langle\Gamma_m, y\oplus\Delta_m\rangle \oplus \langle\Gamma_{out}, E_2(y+\oplus\Delta_m)\rangle$ are independent random variables and are 0 with probability $\frac{1+q}{2}$ then

$$\begin{aligned}
&\mathbb{P}_y\left[\langle\Gamma_m, y\rangle \oplus \langle\Gamma_{out}, E_2(y)\rangle \oplus \langle\Gamma_m, y\oplus\Delta_m\rangle \oplus \langle\Gamma_{out}, E_2(y+\oplus\Delta_m)\rangle = 0\right]\\
&=\mathbb{P}_y\left[\langle\Gamma_m, y\rangle \oplus \langle\Gamma_{out}, E_2(y)\rangle = 0\right]\mathbb{P}_y\left[\langle\Gamma_m, y\oplus\Delta_m\rangle \oplus \langle\Gamma_{out}, E_2(y+\oplus\Delta_m)\rangle = 0\right]\\
&+\mathbb{P}_y\left[\langle\Gamma_m, y\rangle \oplus \langle\Gamma_{out}, E_2(y)\rangle = 1\right]\mathbb{P}_y\left[\langle\Gamma_m, y\oplus\Delta_m\rangle \oplus \langle\Gamma_{out}, E_2(y+\oplus\Delta_m)\rangle = 1\right]\\
&=\left(\frac{1+q}{2}\right)^2 + \left(\frac{1-q}{2}\right)^2\\
&=\frac{1+q^2}{2}
\end{aligned}$$

On the other hand, we can simplify the equation as follows

$$\langle\Gamma_m, y\rangle\oplus\langle\Gamma_{out}, E_2(y)\rangle\oplus\langle\Gamma_m, y\oplus\Delta_m\rangle\oplus\langle\Gamma_{out}, E_2(y+\oplus\Delta_m)\rangle = \langle\Gamma_m, \Delta_m\rangle\oplus\langle\Gamma_{out}, E_2(y)\rangle\oplus\langle\Gamma_{out}, E_2(y+\oplus\Delta_m)\rangle$$

Hence, depending on the sign of $\langle\Gamma_m, \Delta_m\rangle$,

$$\mathbb{P}_y\left[\langle\Gamma_{out}, E_2(y)\rangle \oplus \langle\Gamma_{out}, E_2(y+\oplus\Delta_m)\rangle = 0\right] = \frac{1\pm q^2}{2}$$

Assuming the independence between the differential approximation on E_1 and the linear approximation on E_2, we obtain

$$\begin{aligned}
&\mathbb{P}_x\left[\langle\Gamma_{out}, E(x)\rangle \oplus \langle\Gamma_{out}, E(x\oplus\Delta_{in})\rangle = 0\right]\\
&=\mathbb{P}\left[E_1(x\oplus\Delta_{in}) = E_1(x)\oplus\Delta_m\right]\mathbb{P}\left[\frac{\langle\Gamma_{out}, E(x)\rangle \oplus \langle\Gamma_{out}, E(x\oplus\Delta_{in})\rangle = 0}{E_1(x\oplus\Delta_{in}) = E_1(x)\oplus\Delta_m}\right]\\
&+\mathbb{P}\left[E_1(x\oplus\Delta_{in}) \neq E_1(x)\oplus\Delta_m\right]\mathbb{P}\left[\frac{\langle\Gamma_{out}, E(x)\rangle \oplus \langle\Gamma_{out}, E(x\oplus\Delta_{in})\rangle = 0}{E_1(x\oplus\Delta_{in}) \neq E_1(x)\oplus\Delta_m}\right]\\
&=p\left(\frac{1\pm q^2}{2}\right) + (1-p)\cdot\frac{1}{2}\\
&=\frac{1\pm pq^2}{2}
\end{aligned}$$

Therefore we can deduce $cor_{x\in\mathbb{F}_2^n}\left[\langle\Gamma_{out}, E(x)\rangle \oplus \langle\Gamma_{out}, E(x\oplus\Delta_{in})\rangle\right] = pq^2$. □

As a consequence, an attacker can distinguish the cipher from a random permutation by preparing $\mathcal{O}(p^{-2}q^{-4})$ pairs of chosen plaintexts. This type of attack tends to be effective on ciphers in which the optimal differential probability and the optimal correlation decrease too fast with the number of rounds to be able to mount either of these attacks on their own.
In practice, the independence assumption between E_1 and E_2 is sometimes not true, resulting in wrong estimates for the correlation. A solution to this independence assumption is to split E in three sub ciphers $E = E_2 \circ E_m \circ E_1$ and to experimentally compute the correlation of the middle cipher E_m. In this case, if we denote by r the experimental correlation of the middle part, then the differential-linear distinguisher would have a correlation of pr^2q^2.

Chapter 3
Improving Differential-Linear Cryptanalysis, Application to Chaskey

This chapter consists of a summary of my joint work with Marek Broll, Frederico Canale, Antonio Flórez-Gutiérrez, Gregor Leander, María Naya-Plasencia, and Yosuke Todo about the optimizations of differential-linear cryptanalysis of ARX constructions and its application to Chaskey [Bei+22]. This paper features different types of optimizations that are independent of one another. In this chapter, we will highlight the part of the paper that concerns the improvements on the differential part, since it matches my contribution in this paper.

Contents

This chapter organizes as follows: first, based on ideas from [BLT20], we describe some first elements of optimization for the differential part of differential-linear attacks. Then, we will describe the improvement from [Bei+22]. Finally, we will give an example of an application on Chaskey.

3.1 Improving the differential-linear distinguisher

In [BLT20], the authors described a technique to improve differential-linear attacks and apply it to Chaskey and Chacha. Following this work, we further improved differential-linear attacks in [Bei+22]. In this section, we will describe the technique introduced in

[BLT20] that improves the correlation of a differential-linear distinguisher. Let's start by considering a cipher E on which we want to mount a differential-linear attack. Following Section 2.3, E is split into two parts $E = E_2 \circ E_1$ where

$$\mathbb{P}_{x \in \mathbb{F}_2^n} \left[E_1(x) \oplus E_1(x \oplus \Delta_{in}) = \Delta_m \right] = p$$

and

$$\mathrm{cor}_{y \in \mathbb{F}_2^n} \left[\langle \Gamma_m, y \rangle \oplus \langle \Gamma_{out}, E_2(y) \rangle \right] = q.$$

The core idea of this improvement is to change the sampling set to increase the probability of the differential part. For instance, we could consider the set:

$$\mathcal{X} = \{ x \in \mathbb{F}_2^n \ : \ E_1(x) \oplus E_1(x \oplus \Delta_{in}) = \Delta_m \}$$

By sampling elements in $\mathcal{X}$, we obtain $\mathbb{P}_{x \in \mathcal{X}} \left[E_1(x) \oplus E_1(x \oplus \Delta_{in}) = \Delta_m \right] = 1$, hence the associated differential-linear distinguisher has the following correlation:

$$cor_{x \in \mathcal{X}} \left[\langle \Gamma_{out}, E(x) \rangle \oplus \langle \Gamma_{out}, E(x \oplus \Delta_{in}) \rangle \right] = q^2$$

Although the probability of the differential part is 1, we can not use $\mathcal{X}$ in practice because its structure is rather complicated and also because it depends on the key. However, it might be possible to find a vectorial space $\mathcal{U}$: *neutral space* such that

$$x \in \mathcal{X} \Rightarrow x \oplus \mathcal{U} \subset \mathcal{X}.$$

Therefore, given one element of x and a neutral space $\mathcal{U}$, we can generate $2^{\dim(\mathcal{U})}$ good elements for free. Hence, once we have identified a neutral space $\mathcal{U}$, there are two scenarios:

- $|U| > q^{-4}$. In this case, we have enough data to mount a differential-linear attack on E, see Section 3.1.1.
- $|U| < q^{-4}$. In this case, we lack data to mount the attack with one neutral space. Hence, a new approach needs to be considered, see Section 3.1.2

3.1.1 Differential-Linear Cryptanalysis with Neutral Space

In this section, we follow the ideas from [BLT20] and discuss how to improve the correlation of a differential-linear distinguisher given a neutral space $\mathcal{U}$ big enough to mount the attack, i.e $|U| > q^{-4}$. In Algorithm 3.1, we summarize the improved distinguisher, it first consists in picking a random element $a \in \mathbb{F}_2^n$, then empirically computing $cor_{x \in a \oplus \mathcal{U}} \left[\langle \Gamma_{out}, E(x) \rangle \oplus \langle \Gamma_{out}, E(x \oplus \Delta_{in}) \rangle \right]$ and compare it to q^2.

Complexity. Since $cor_{x \in a \oplus \mathcal{U}} \left[\langle \Gamma_{out}, E(x) \rangle \oplus \langle \Gamma_{out}, E(x \oplus \Delta_{in}) \rangle \right] = q^2$ if $a \in \mathcal{X}$, our algorithm will succeed if the randomly chosen element in step 1 happens to be an element of $\mathcal{X}$. On the other hand, since $a \in \mathcal{X} \Leftrightarrow [E_1(a) \oplus E_1(a \oplus \Delta_{in}) = \Delta_m]$, then $\mathbb{P}_{a \in \mathbb{F}_2^n}(a \in \mathcal{X}) = p$, the algorithm should loop $\mathcal{O}(p^{-1})$ times. Hence the complexity of the attack is $\mathcal{O}(p^{-1} q^{-4})$ compared to $\mathcal{O}(p^{-2} q^{-4})$ for the traditionnal differential-linear cryptanalysis.

Algorithm 3.1 Improved differential-linear distinguisher with neutral spaces

Require: U
1: $a \leftarrow_R \mathbb{F}_2^n$
2: $c \leftarrow cor_{x \in a \oplus \mathcal{U}} \left[\langle \Gamma_{out}, E(x) \rangle \oplus \langle \Gamma_{out}, E(x \oplus \Delta_{in}) \rangle \right]$ ▷ Empirical computation
3: **if** $c = q^2$ **then**
4: **return** Success
5: **else**
6: Go to 1

3.1.2 Differential-Linear Cryptanalysis with Probabilistic Neutral Space

Following the ideas from [BLT20], we relax some hypotheses on the definition of a neutral space to increase its dimension, hence the data that we can generate. Indeed, we now want the probability $\mathbb{P}_{x \in \mathcal{X}, u \in \mathcal{U}} \left[x \oplus u \in \mathcal{X} \right] = p'$ to be high rather than equal to 1. In that case, the associated differential-linear distinguisher will have the following correlation:

$$cor_{x \in \mathcal{A}} \left[\langle \Gamma_{out}, E(x) \rangle \oplus \langle \Gamma_{out}, E(x \oplus \Delta_{in}) \rangle \right] = p' q^2$$

Therefore, the data threshold will also increase to $p'^{-2} q^{-4}$. Similarly to the previous section, Algorithm 3.2 describes an improved differential-linear distinguisher using probabilistic neutral spaces. Now, the complexity is $\mathcal{O}(p^{-1} p'^{-2} q^{-4})$.

Algorithm 3.2 Improved differential-linear distinguisher with probabilistic neutral spaces

Require: U
1: $a \leftarrow_R \mathbb{F}_2^n$
2: $c \leftarrow cor_{x \in a \oplus \mathcal{U}} \left[\langle \Gamma_{out}, E(x) \rangle \oplus \langle \Gamma_{out}, E(x \oplus \Delta_{in}) \rangle \right]$ ▷ Empirical computation
3: **if** $c \geq p' q^2$ **then**
4: **return** Success
5: **else**
6: Go to 1

3.1.3 Computing a Neutral Space

In this section, we will describe an algorithm that finds a neutral space defined in previous sections. It starts by considering some element $u \in \mathbb{F}_2^n$ and empirically compute $\mathbb{P}_{x \in \mathcal{X}} \left[x \oplus u \in \mathcal{X} \right]$. Then if this probability is higher than some threshold (hence u is a neutral vector), we store it and repeat this process for another $u' \in \mathbb{F}_2^n$. Then, with the gathered elements, we can derive the vectorial space that they spawned and its basis.

Since looking at all the vectors of $\mathbb{F}_2^n$ would be too costly, we will look into the vectors of low hamming weight to build our basis. Hence, the elements obtained may

spawn a neutral that is not the biggest. The Algorithm 3.3 describes the procedure that empirically computes the probability $\gamma_u = \mathbb{P}_{x\in\mathcal{X}}[x \oplus u \in \mathcal{X}]$ for u an element of hamming weight lower than some threshold w.

Algorithm 3.3

Require: Number of samples T, input difference Δ_{in}, output difference Δ_m, hamming weight w

Ensure: Probabilities $(\gamma_u)_{u\in\mathcal{B}(0,w)}$

Let $s = 0$ and $c_e = 0$ for $u \in \mathcal{B}(0, w)$.
for $i = 1$ to T **do**
    Pick a random X and compute $E_1(X)$ and $E_1(X \oplus \Delta_{in})$
    if $E_1(X) \oplus E_1(X \oplus \Delta_{in}) = \Delta_m$ **then**
        Increment s
        for $u \in \mathcal{B}(0, w)$ **do**
            $\hat{X} \leftarrow X \oplus u$.
            if $E_1(\hat{X}) \oplus E_1(\hat{X} \oplus \Delta_{in}) = \Delta_m$ **then**
                Increment c_u
for $u \in \mathcal{B}(0, w)$ **do**
    $\gamma_u = c_u/s$
return $(\gamma_u)_{u\in\mathcal{B}(0,w)}$

For some probability threshold p_T, we can consider the vectors that have an empirical probability γ_u larger than p_T and the vectorial space that they spawn. Let $e_1 \cdots e_{\dim(\mathcal{U})}$ denote a basis of $\mathcal{U}$, we can see that:

$$\begin{aligned}\forall u \in \mathcal{U},\ \mathbb{P}_{x\in\mathcal{X},u\in\mathcal{U}}[x \oplus u \in \mathcal{X}] &\geq \prod_{i=1}^{\dim(\mathcal{U})} \mathbb{P}_{x\in\mathcal{X}}[x \oplus e_i \in \mathcal{X}] \\ &\geq p_T^{\dim(\mathcal{U})}\end{aligned}$$

By considering the case of $p_T = 1$, $\mathcal{U}$ corresponds to a neutral space. In the case of $p_T < 1$, we obtain a probabilistic neutral space of probability $p_T^{\dim(\mathcal{U})}$. In practice it would be better to compute and consider $\prod_{i=1}^{\dim(\mathcal{U})} \mathbb{P}_{x\in\mathcal{X}}[x \oplus e_i \in \mathcal{X}]$ as bound for the probability since it is tighter. Nonetheless, it is important to note that the choice of basis influences the bound.

3.1.4 Using the Conditional Differential Framework for Finding Better Subspaces

The production of vectors in a neutral space can be achieved practically by following a process similar to the one used in conditional differential attacks. The basis elements of $\mathcal{U}$, as defined in [KMN10b], are referred to as *freebits*. These involve more complex relations derived from the differential paths, rather than just simple single-bit relations.

Three types of conditions are presented in Sect.3.2 of[KMN10b]: type-0, which only involves public bits; type-1, which involves both secret and public bits; and type-2, which are conditions directly on the key bits. The *freebits* are the basis elements that do not affect type-1 conditions. These definitions allow for the improvement of previous attacks in two ways: by increasing the number of key bits recovered by the differential-linear attack using type-2 conditions, and by increasing the size of $\mathcal{U}$ using the freebits.

This section provides some general ideas and hints for using this framework to improve the analysis of ARX constructions. In Section 3.3, we present a detailed example of how to determine additional key bits with type 2 conditions and how to increase the number of freebits with evolved relations for Chaskey. Note that some of the relations derived by this method could not have been found by Algorithm 3.2.

Conditional differential framework for differential-linear attacks The conditional differential framework for differential-linear attacks can be improved in three ways using the definitions from [KMN10b, Sect. 3.2]:

1. By exhaustively testing the input values that maintain the type-1 conditions fixed to a specific value, we can increase the size of $\mathcal{U}$. Indeed, if these conditions are assumed to be true, they will remain true for the entire sampling set. However, these tested bits are not entirely unconstrained since the type-1 conditions must remain fixed.

2. The consideration of a particular set of plaintexts to verify the differential path can involve presupposing information on the associated key bits or key bit relations. This information is given directly by conditions of type-2 and indirectly by conditions of type-1. This information can be utilized to recover more bits and reduce the complexity of the key-search part in the final rounds.

3. Combination of both: guessing some key bits, that might be useful for the linear part, and that simultaneously might allow to detect sampling bit relations that follow the path with probability 1.

Main ideas for exploiting the conditions on ARX In the following, we outline some general concepts for utilizing conditions on ARX constructions. Rather than relying solely on single non-active bit flipping to define freebits, we can investigate the impact of flipping the parity of differences as extra sampling bits, when feasible. Hence, we will establish guidelines for flipping the parity of differences. We can distinguish various significant scenarios, and we present here four cases of particular interest:

(i) If a pair of differences is going to be erased after a modular addition, changing the parity of one will need changing the parity of the other.

(ii) If a bit difference is staying at the same position (and not propagating further) after a modular transition, changing its parity will not affect the transition.

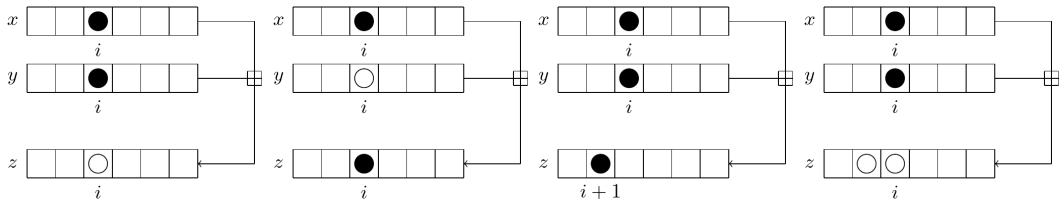


Figure 3.1: The four different scenarios

(iii) If two active bits at position i will produce a difference after the modular addition at position $i+1$ (move the difference), flipping both active bits at the same time will change the parity of the output at $i+1$.

(iv) If two words are added with a difference in position i and in positions i and $i+1$ respectively, and we want to absorb the differences after the modular additions, the carries of the previous positions will not affect the bits after position i. We can also change the parity of the three bits simultaneously, and the differences will still be absorbed, and the values will stay the same. Of course, all this might have an effect on further rounds, which will have, in turn, to be taken into account.

It is also useful to keep in mind that when we find multiple input bits that only affect the differential transitions through XOR operations, then swapping an even number of these bits won't affect the verification of the differential path. When considering carries, they can have a low probability of affecting transitions. One thing to note is that if there is a sum of two zeros at a specific position i, then the bits at lower positions will not affect the carries at higher positions. This means that if we guess a small number of key bits to fix two bits to zero, it may generate many more bits for the sampling part with probability one, only if they influence the differential path through these carries.

3.2 Application to Chaskey

The goal of this section is to present our improved attack on Chaskey. Indeed, we will show how the application of the new techniques presented in [Bei+22] resulted in increasing the dimension of $\mathcal{U}$. Nonetheless, the improvements presented in [Bei+22] are not solely restricted to the differential parts since key recovery optimizations on the linear part are also presented. However, this chapter focuses on the differential aspect of this work (refer to [Bei+22] for the other improvements). First, we will describe the Chaskey specification, then we will apply our new algorithmic technique to compute some elements of $\mathcal{U}$ and finally, we will use the conditional differential technique to recover the missing elements of $\mathcal{U}$.

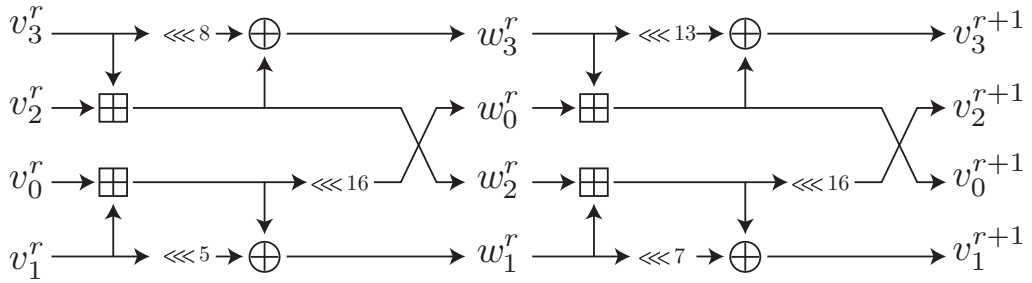


Figure 3.2: The round function of Chaskey.

Algorithm 3.4 Chaskey round function

Require: $(v_0^r, v_1^r, v_2^r, v_3^r)$
$w_0^r \leftarrow (v_0^r \boxplus v_1^r) \lll 16$
$w_1^r \leftarrow (v_1^r \lll 5) \oplus (v_0^r \boxplus v_1^r)$
$w_2^r \leftarrow v_2^r \boxplus v_3^r$
$w_3^r \leftarrow (v_3^r \lll 8) \oplus (v_2^r \boxplus v_3^r)$
$v_0^{r+1} \leftarrow w_0^r \boxplus w_3^r$
$v_1^{r+1} \leftarrow (w_1^r \lll 7) \oplus (w_1^r \boxplus w_2^r)$
$v_2^{r+1} \leftarrow (w_1^r \boxplus w_2^r) \lll 16$
$v_3^{r+1} \leftarrow (w_3^r \lll 13) \oplus (w_0^r \boxplus w_3^r)$
return $(v_0^{r+1}, v_1^{r+1}, v_2^{r+1}, v_3^{r+1})$

3.2.1 Chaskey Specfication

Chaskey [Mou+14] is a lightweight MAC algorithm whose underlying primitive is an ARX-based permutation in an Even-Mansour construction, i.e., Chaskey-EM. The permutation operates on four 32-bit words, i.e., the block size is 128 bits. In the version proposed in [Mou+14], the permutation employs 8 rounds. The round function is described in Algorithm 3.4 and depicted in Fig. 3.2. In [Mou15], the author proposed a version with an increased number of rounds (from 8 to 12), and this version is currently standardized in ISO/IEC 29192-6:2019. The designers claim security up to 2^{80} computations as long as the data is limited to 2^{48} blocks.

In Fig. 3.2 and Algorithm 3.4, $(v_0^r, v_1^r, v_2^r, v_3^r)$ denotes the input of the rth round function, and $(w_0^r, w_1^r, w_2^r, w_3^r)$ denotes the state after applying the half round for $(v_0^r, v_1^r, v_2^r, v_3^r)$.

3.2.2 Overview of the Attack

We now describe the attack on Chaskey, we will start by giving a brief overview, then we will get into the details of the neutral space $\mathcal{U}$.

Table 3.1: Probability that adding one basis element affects the output difference. The techniques we developed in [Bei+22] allowed to add the elements colored in red to the basis

Probability	Basis	Number of indices
$\gamma_j = 1$	v_2 : 16,17,18,19,20,22,23,24,25,30,31 v_3 : 16,17,18,19,20,22,23	18
$0.93 \leq \gamma_j < 1$	v_0 : 19,20,31 $\quad$ v_3 : 24,25,30 v_1 : 19,20	8
$\gamma_j = 1$	$v_0[8] \oplus v_1[8,13] \oplus v_2[29]$ $v_2[21,29] \oplus v_3[21]$ $v_0[18,21,30] \oplus v_1[21,26,30] \oplus v_2[3,26] \oplus v_3[26,27]$ $v_2[15] \oplus v_3[15]$	4

Similar to the previous differential-linear attack from [Leu16a], the attack we presented in [Bei+22] first consists in dividing the cipher into three sub ciphers. For the 7-round attack, we used E_1 covering 1.5 rounds, E_m covering 4 rounds, E_2 covering 0.5 rounds, and the key-recovery, i.e., the function F, is covering 1 round. We also presented a 7.5-round attack, where E_2 covers 1 round instead of 0.5 rounds. The differential characteristic and the linear trail are applied to E_1 and E_2, respectively, while the experimental differential-linear distinguisher is applied to the middle part E_m. Note that we focus the analysis on E_1 from the following subsection (details on the linear treatment of E_2 are given in [Bei+22]).

3.2.3 Finding More Neutral Vectors

Let's first recall the basis of $\mathcal{U}$ introduced in [BLT20] (see the first two-row blocks in Table 3.1). This initial basis was computed using Algorithm 3.3 with a weight $w = 1$. Here, the threshold of the probability is relaxed from 0.95 to 0.93, and $v_3[30]$ (in red) is newly added in the basis. If the Algorithm 3.3 was also able to find neutral elements with low weight, the conditional differential techniques provide us with elements with probability 1, which could not be found by the Algorithm since the weight of the vectors were bounded to restrict the computations (see the third-row block in Table 3.1). A linear subspace $\mathcal{U}$, formed by elements that don't affect the output with probability 1, and whose dimension is $18 + 4 = 22$ is finally used to attack 7-round Chaskey. In addition, all, i.e., $18 + 8 + 4 = 30$, basis elements are used to attack 7.5-round Chaskey.

In the following section, we provide the details on how to obtain these relations. We use the conditional differential framework and Fig. 3.3 to recover for free the value of some key bits and also to find additional bits of information for sampling and increase the

dimension of $\mathcal{U}$ from 18 as given in [BLT20] (and involving exclusively one-bit relations) to 22, or 23 if one-bit relation on the key is known. The new proposed set of freebits (or relations with probability 1) is optimal and no more such relations exist.

3.3 Exploiting the Conditions for Finding Chaskey Relations

In Fig. 3.3, we have depicted the relations and the influence of the input bits on the conditions of the differential path. The bits that stay white (and have no pink color beneath, coming from the carries of the furthest additions) are the bits that do not affect the differential transitions.

It is easy to see how the bits provided in [BLT20] as available for sampling with probability one are the only white ones, and therefore not needed for the differential conditions: [31,30,25,24,23,22,20,19,18,17,16] from v_2 and [23,22,20,19,18,17,16] from v_3. The differences are represented in grey. Dependencies in colors. A 'g' in the position of a difference means that this difference will go away (be absorbed) after the next addition. An 's' means that the difference stays where it is, while 'm' means that it moves one position to the left. The color of the bits with differences in each transition will be applied to all the bits that might affect this transition. Carries are not directly applied to the involved bits but to the upper row to report the difference this implies.

Please note that for instance bits 28 and 27 from v_2 cannot be included as the carry of the position 29 is needed by the orange bit relations, i.e., the differences after one round at position 29 of v_2 and v_3, but as said in Sect. 3.1.4, the bits of previous positions to 26 and 27 will not affect this orange carry anymore due to the particular configuration of 26 and 27. The bits provided in [BLT20] that are neutral with very high probability are 20 and 19 from v_1 and 31, 20 and 19 from v_0 and 25 and 24 of v_3.

Let us now see how can we use the conditional differential ideas and Fig. 3.3 to recover for free the value of some key bits and also to find additional bits of information for sampling and increasing the dimension of $\mathcal{U}$ from 18 as given in [BLT20] (and involving exclusively one-bit relations) to 22, or 23 if one-bit relation on the key is known.

Additional space for sampling Using Fig. 3.3 we can try to exploit the conditions to find more evolved relations for increasing the size of $\mathcal{U}$. Let us provide an example: we consider the case where we flip the bit from $v_0[8]$. The corresponding difference, marked with a 'g', will have a change of parity. For this difference to be absorbed, we need to also flip the other blue difference that will be used for absorbing this one: $v_1[8]$. However, if we flip this one, the value of the bit $v_1[13]$ after one round, which does not contain a difference, will be flipped also. Indeed this value is produced by the XOR of $v_1[8]$ shifted of 5 positions and the sum of v_0 and v_1, which has a difference in position 13, marked with an 's': these differences cancel out in both cases, but the value of the resulting bit will change with the parity of $v_1[8]$, and the value of this pink will affect the final light-pink transition in the third round, as can be seen in the picture. To avoid

this, we have to also flip $v_1[13]$: the state v_1 after 1 round will be known the same, but the orange bit $v_2[29]$ after one round that contains a difference and a 'g' will have the parity changed. To make the related transition satisfied, we need to also change the parity of the other orange bit with a 'g': we flip $v_2[29]$ from the first round, which does not have a difference, but that will change the parity of $v_3[29]$ after the XOR. This bit will not have any more influence in the remaining transitions, so we have found our close relation. In total, we found four new probability-one relations by hand using this same technique. We have verified these relations by implementing a similar algorithm to Algorithm 3.3 that tests whether a vector u is neutral, as well as exhaustively searched all the ones with weight at most 3, and found that no other such relations exist.

3.4 Conclusion

To summarize this chapter, we presented techniques and algorithms that were implemented to improve differential-linear attacks by artificially increasing the correlation of the underlying distinguisher by improving the search of neutral vectors. As an example of application, we presented the case of Chaskey which is an ARX design. Adapting these techniques to SPN ciphers [Bro+22a] resulted in an improvement on the attack on another cipher that was a candidate for the Advance Encryption Standard: Serpent [ABK00]. A nice formalization of this improvement in the SPN case uses S-box decomposition using binary decision tree [Bro+21]. This decomposition can be used to speed up key recovery since it allows to perform refined key guesses.

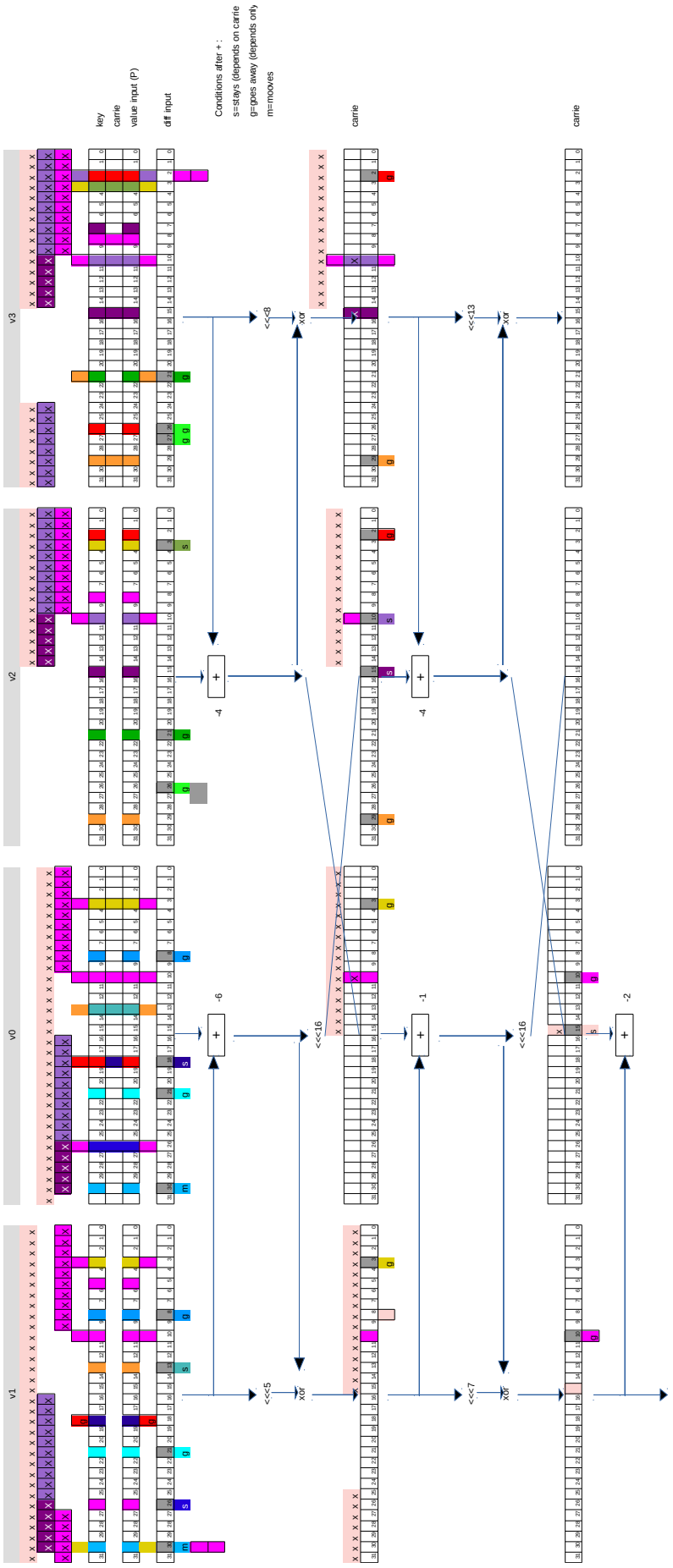


Figure 3.3: Conditions on the differential transitions for the 3 first rounds of Chaskey

Chapter 4
Better Steady than Speedy: Full break of SPEEDY-7-192

This chapter is inspired by [Bou+23a] which is a joint work with Christina Boura, Rachelle Heim Boissier, and María Naya-Plasencia. In this paper, we presented a full break on the main version of the block cipher Speedy [Lea+21] using differential cryptanalysis combined with evolved techniques.

Contents

We present in the next two sections an application of the techniques introduced in Section 2.1 against the SPEEDY family of block ciphers. Section 4.2 is dedicated to the distinguisher part, while Section 4.3 describes the key recovery part for SPEEDY-7-192.

We start by briefly presenting the specifications of the SPEEDY family of ciphers.

4.1 Specifications of the SPEEDY family of block ciphers

The SPEEDY family of ciphers is a family of lightweight block ciphers introduced by Leander, Moos, Moradi, and Rasoolzadeh at CHES 2021 [Lea+21]. The main design goal of these primitives was to achieve extremely low latency. This goal was notably reached thanks to the design of a dedicated 6-bit bijective S-box.

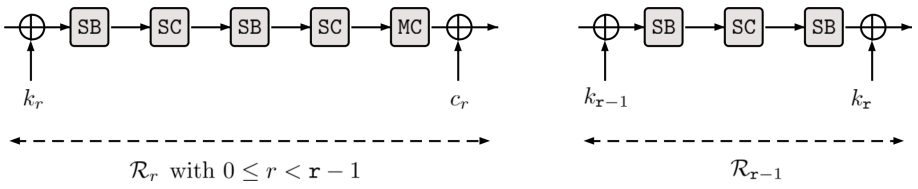


Figure 4.1: The round function of `SPEEDY-r-192` for the first $\mathtt{r} - 1$ rounds (left) and the last round (right).

There are different `SPEEDY` variants that differ in block size, key size, and number of rounds. More precisely, the block cipher `SPEEDY-r-`6ℓ has a block and key size of 6ℓ bits and is iterated over `r` rounds. The internal state is viewed as a $\ell \times 6$ rectangle array of bits. Following the notation of [Lea+21], we will denote by $x_{[i,j]}$, $0 \le i < \ell$, $0 \le j < 6$, the bit located at row i and column j of the state x. Note that all indices start from zero and the zeroth bit or word is always considered to be the most significant one. Furthermore, if there is an addition or a subtraction in the indices of the state, this is done modulo ℓ for the first (row) index and in modulo 6 for the second (column) index.

The default block and key size for `SPEEDY` is 192 bits and this instance is denoted by `SPEEDY-r-192`. It is suggested to iterate this instance over 5,6 or 7 rounds. Next, we provide the specifications of the round function for `SPEEDY-r-192`. Note that for this variant, the state is seen as $(\ell \times 6)$-bit rectangle, with $\ell = 32$.

4.1.1 Round function of `SPEEDY-r-192`

The internal state is first initialized with the 192-bit plaintext. Then, a round function $\mathcal{R}_r$ is applied to the state `r` times, where `r` is typically 5, 6 or 7. The round function is composed of four operations: First, `AddRoundKey` ($\mathtt{A}_{k_r}$) XORs the round subkey k_r to the state. Then, the `SubBox (SB)` operation applies a 6-bit S-box to each row of the state. Follows the `ShiftColumns (SC)` operation that rotates each column of the state by a different offset. These two operations (`SB` and `SC`) are repeated twice in an alternating manner. After this, the `MixColumns (MC)` operation multiplies each column of the state by a binary matrix. Finally, a constant c_r is XORed to the state by the `AddRoundConstant (AC)` operation. Note that, for the last round, the last `ShiftColumns` as well as the `MixColumns` and the `AddRoundConstant` operations are omitted, while a post-whitening key is XORed to the state. The round function $\mathcal{R}_r$ for the rounds $0 \le r < \mathtt{r} - 1$ while also for the round $\mathtt{r} - 1$ are depicted in Figure 4.1.

In the rest of our paper, we assume that the input (resp. output) to each of the described operations is a vector x (resp. y) $\in \mathbb{F}_2^{32\times 6}$.

- `AddRoundKey` ($\mathtt{A}_{k_r}$): The 192-bit round key k_r is XORed to the internal state. Hence,

$$y_{[i,j]} = x_{[i,j]} \oplus k_{r[i,j]}.$$

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S(x)	8	0	9	3	56	16	41	19	12	13	4	7	48	1	32	35
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
S(x)	26	18	24	50	62	22	44	54	28	29	20	55	52	5	36	39
x	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
S(x)	2	6	11	15	51	23	33	21	10	27	14	31	49	17	37	53
x	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
S(x)	34	38	42	46	58	30	40	60	43	59	47	63	57	25	45	61

Table 4.1: Table representation of the 6-bit S-box S

- `SubBox (SB)`: A 6-bit S-box S is applied to each row of the state. More precisely, for each row i, $0 \leq i < 32$, `SB` operates as follows:

$$(y_{[i,0]}, y_{[i,1]}, y_{[i,2]}, y_{[i,3]}, y_{[i,4]}, y_{[i,5]}) = S(x_{[i,0]}, x_{[i,1]}, x_{[i,2]}, x_{[i,3]}, x_{[i,4]}, x_{[i,5]}).$$

  The table representation of the S-box S is given in Table 4.1.

- `ShiftColumns (SC)`: This operation rotates the j-th column of the state, $0 \leq j < 6$, upside by j bits:

$$y_{[i,j]} = y_{[i+j,j]}.$$

- `MixColumns (MC)`: The `MC` operation of `SPEEDY` applies column-wise and is based on a cyclic binary matrix $\alpha = (\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6)$ whose values depend on the number of rows ℓ:

$$y_{[i,j]} = x_{[i,j]} \oplus x_{[i+\alpha_1,j]} \oplus x_{[i+\alpha_2,j]} \oplus x_{[i+\alpha_3,j]} \oplus x_{[i+\alpha_4,j]} \oplus x_{[i+\alpha_5,j]} \oplus x_{[i+\alpha_6,j]}.$$

  Recall that the additions $i + \alpha_*$ are considered mod ℓ.

  For $\ell = 32$, $\alpha = (1, 5, 9, 15, 21, 26)$.

- `AddRoundConstant` ($\mathtt{A}_{c_r}$): The 192-bit round constant c_r is XORed to the internal state. Hence,

$$y_{[i,j]} = x_{[i,j]} \oplus c_{r[i,j]}$$

  As this operation is not relevant to our analysis we omit the description of the constant values.

4.1.1.1 Key Schedule

The 192-bit master key of `SPEEDY-r-192` is loaded to the state of the first round key k_0. To obtain the next round key, the key schedule consists in simply applying a bit-permutation `PB`. Hence,

$$k_{r+1} = \mathtt{PB}(k_r), \text{ with } k_{r+1[i',j']} = k_{r[i,j]},$$

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P(i)	1	8	15	22	29	36	43	50	57	64	71	78	85	92	99	106
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
P(i)	113	120	127	134	141	148	155	162	169	176	183	190	5	12	19	26
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
P(i)	33	40	47	54	61	68	75	82	89	96	103	110	117	124	131	138
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
P(i)	145	152	159	166	173	180	187	2	9	16	23	30	37	44	51	58
i	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
P(i)	65	72	79	86	93	100	107	114	121	128	135	142	149	156	163	170
i	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
P(i)	177	184	191	6	13	20	27	34	41	48	55	62	69	76	83	90
i	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
P(i)	97	104	111	118	125	132	139	146	153	160	167	174	181	188	3	10
i	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
P(i)	17	24	31	38	45	52	59	66	73	80	87	94	101	108	115	122
i	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
P(i)	129	136	143	150	157	164	171	178	185	0	7	14	21	28	35	42
i	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
P(i)	49	56	63	70	77	84	91	98	105	112	119	126	133	140	147	154
i	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
P(i)	161	168	175	182	189	4	11	18	25	32	39	46	53	60	67	74
i	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
P(i)	81	88	95	102	109	116	123	130	137	144	151	158	165	172	179	186

Table 4.2: The bit-permutation P for `SPEEDY-r-192` with $\beta = 7$ and $\gamma = 1$.

such that

$$(i', j') := P(i, j) \text{ with } (6i' + j') \equiv (\beta \cdot (6i + j) + \gamma) \mod 6\ell,$$

where β and γ are parameters depending on the block length of the cipher and that satisfy the condition that $\gcd(\beta, 6\ell) = 1$. For `SPEEDY-r-192`, the parameters $\beta = 7$ and $\gamma = 1$ are suggested, leading to the permutation P described in Table 4.2.

Algorithm	# rounds attacked	Ref.	Data	Time (in C_E)	Memory	Security claim $(\mathcal{T}, \mathcal{D})$
SPEEDY-5-192	3	[RS22]	$2^{17.6}$	$2^{52.5}$	$2^{25.5}$	$(2^{128}, 2^{64})$
SPEEDY-5-192	5	[Bou+23a]	$2^{101.65}$	$2^{107.8}$	2^{42}	$(2^{128}, 2^{64})$
SPEEDY-6-192	5.5	[Bou+23a]	$2^{121.65}$	$2^{127.8}$	2^{42}	$(2^{128}, 2^{128})$
SPEEDY-6-192	6	[Bou+23a]	$2^{121.65}$	$2^{151.67}$	2^{42}	$(2^{128}, 2^{128})$
SPEEDY-7-192	7	[Bou+23a]	$2^{187.28}$	$2^{187.84}$	2^{42}	$(2^{192}, 2^{192})$

Table 4.3: Summary of SPEEDY cryptanalysis

4.1.2 Security Claims

The authors made security claims for the three main versions of SPEEDY-r-192. For the 5-round version the authors expect no attack with complexity better than 2^{128} in time when data complexity is limited to 2^{64}. On the other hand, SPEEDY-6-192 should achieve 128-bit security, while SPEEDY-7-192 is expected to provide full 192-bit security. In addition, the designers of SPEEDY gave a claim on the amount of key recovery rounds: "The attacker cannot add more than one round to extend a distinguisher". In Table 4.3, we give a summary of the cryptanalysis done on the SPEEDY family of ciphers. The differential attack presented in this chapter will contradict both the time and data security claims as well as the security claim on the number of key recovery rounds.

4.2 Finding Good Differentials on SPEEDY

4.2.1 Differential properties of SPEEDY

We describe in this section the differential properties of the non-linear and linear layers of SPEEDY.

4.2.1.1 Differential properties of the S-box

The SPEEDY family of cipher employs a 6-bit S-box S whose differential uniformity is $\delta_S = 8$. This means that the highest probability of a differential transition through S is 2^{-3}. One particularity of this S-box that we exploit in our attacks is that almost all 1-bit to 1-bit differential transitions are possible. Moreover, these minimal weight transitions often have a relatively high probability. Table 4.4 summarizes all these transitions, together with their corresponding probability. The full DDT is given in [Bou+23a].

Another particularity is that 1-bit to 1-bit differential transitions can be chained within one round through the SB $\circ$ SC $\circ$ SB operation. All of them are possible and three of them achieve the maximum probability of 2^{-6}.

α/β	1	2	4	8	16	32
1	2	-	4	2	4	2
2	1	2	4	4	2	2
4	-	3	2	-	3	1
8	1	1	3	3	1	1
16	-	-	4	4	3	4
32	1	1	2	3	1	-

Table 4.4: Summary of all the 1-bit input differences α to 1-bit output differences β. The corresponding probability can be obtained by multiplying the coefficients of the table by 2^{-5}. The symbol - means that the corresponding transition is impossible.

4.2.1.2 Differential properties of the `MixColumns` operation

The branch number of the `MC` operation is 8, which is the maximum possible value for the vector α chosen. As the maximum differential probability over 1 round is 2^{-6}, this means that an upper bound on the probability of any differential transition over two rounds is $(2^{-6})^8 = 2^{-48}$. The inverse `MixColumns` operation is defined with the vector

$$\alpha^{-1} = (4, 5, 6, 7, 10, 12, 14, 15, 16, 18, 19, 20, 21, 22, 23, 24, 25, 28).$$

This means in particular, that a column with a single active bit, will lead after the inverse of the `MixColumns` operation to 19 active bits, while a column with two active bits will be transformed after the inverse `MixColumns` to a column with at least 12 active bits.

4.2.2 Searching for good differential trails

We describe in this section the methodology we followed to find the trails used in our attacks. Our idea was to precompute at first all good one-round trails and then chain them to create longer trails that have a high probability.

4.2.2.1 Searching for good one-round trails.

Let M be the matrix used in the `MixColumns` operation. To find good one-round trails, we first computed and stored all ordered pairs of columns $(x, M(x)) \in \mathbb{F}_2^{32} \times \mathbb{F}_2^{32}$ such that both columns x and $M(x)$ have at most 7 active bits each. This led to a total of 5248 pairs $(x, M(x)) \in \mathbb{F}_2^{32} \times \mathbb{F}_2^{32}$. However, these 5248 pairs can be divided into 164 equivalence classes, each equivalence class corresponding to the 32 rotations of a different activity pattern inside a column. We then stored in a table `T` one representative per equivalence class and used these pairs to pre-compute and store all 1-round trails satisfying some particular criteria. To describe this phase we need to introduce the following notation. Let `st[0]` be the initial state for our computation. We denote by `st[1]` the resulting state after applying `MC` to `st[0]`, `st[2]` the state after applying `SB`

to st[1], st[3] the state after applying SC to st[2], st[4] the state after applying SB to st[3], st[5] the state after applying SC to st[4] and finally st[6] the state after applying MC to st[5]:

$$\texttt{st[0]} \xrightarrow{MC} \texttt{st[1]} \xrightarrow{SB} \texttt{st[2]} \xrightarrow{SC} \texttt{st[3]} \xrightarrow{SB} \texttt{st[4]} \xrightarrow{SC} \texttt{st[5]} \xrightarrow{MC} \texttt{st[6]}.$$

We computed all such propagations (st[0], st[6]) satisfying the following conditions:

- st[0] has a single active column c_0 such that $(c_0, M(c_0)) \in \texttt{T}$,
- st[5] has a single active column c_5 such that $(c_5, M(c_5)) \in \texttt{T}$,
- st[2] has at most two active bits per row,
- the probability of the trail (st[0], st[6]) is strictly higher than 2^{-49}.

For all trails satisfying the above conditions, we stored in a table the states (st[0], st[5]) together with the probability of the corresponding trail. We obtained a total number of 48923 one-round trails, which we stored. Note that each trail can be shifted column-wise to form 32 other valid one-round trails. Thus, in total, there are 1565536 one-round trails which satisfy our criteria.

We now justify the criteria used for computing these 1-round trails. Our main constraint was computing time, as considering all 1-round trails is computationally infeasible. Furthermore, as we wanted to store the trails and reuse them, the memory needed to be reasonable as well. Limiting the computation to states with a single active column before and after each MixColumns computation is a reasonable assumption, as states with more active columns would lead by the inverse ShiftColumns operation to many active rows. Furthermore, by doing initial experiments for computing long trails, we noticed that all good trails found never had more than 7 active bits in a column. This can be explained by the fact that more active rows naturally lead to lower probability transitions through the SubBox operation. We then limited the transition through the first SubBox operation to only transitions from rows with Hamming weight one to rows with Hamming weight at most two. While transitions activating in the output more bits per row can still lead to good trails respecting the other criteria, only a small proportion of these transitions does so, while the computational gain for not considering them is huge. Finally, we limited the probability of the trails to 2^{-49} in order not to have to store too many trails for the second phase. This particular bound came from our initial experiments, where we noticed that the probability of all 1-round trails that were part of the longer trails we found, had probability strictly higher than this bound.

We claim by no means that the chosen criteria lead to all the one-round trails that could be part of optimal longer trails, however, we believe that our strategy is a reasonable trade-off between optimality and efficiency.

4.2.2.2 Searching for longer trails.

In a second step, we used the precomputed 1-round trails to create longer ones. To do so, we started by chaining our pre-computed one-round trails to obtain r-round trails.

To begin, we exhaustively ran through all the precomputed one-round trails and searched for the ones that can be chained. Recall here that the starting state and ending state of each round trail are the states just before the `MixColumns` application. The chaining condition is very simple and consists in simply verifying that the final state of a one-round trail is the same as a column-wise rotation of the starting state of the following one by an integer $0 \leq \iota < 32$. Note that when $\iota \neq 0$, the full one-round trail concerned is also rotated column-wise. Also note that doing so, we only obtain an element of an equivalence class modulo the column-wise rotation. To make our search efficient, we first sorted the states by Hamming weight and active column coordinate of their initial and final state. Following this procedure, we found 1476978 2-round trails, each of them giving by rotation another 32 valid 2-round trails. We followed a similar procedure to obtain 46471749 3-round trails which can also be rotated column-wise to obtain 32 times more valid 3-round trails. To compute the 4-round trails we use in our attack on the 7-round version, we chained the 2-round trails with themselves rather than using the 3-round trails for the search to be more efficient. We stored the most interesting 4-round trails we found based on criteria of low probability and adaptability to the key recovery step as described in the next section.

From now on, for each r-round trail, we use the following notations. Let $st^{start}[k]$ (resp. $st^{end}[k]$) be the starting state (resp. the ending state) of each one-round trail composing the r-round trail, for $0 \leq k < r$. Denote also by c_{start} the active column of $st^{start}[0]$ and by c_{end} the active column of $st^{end}[r-1]$. Let w_0 be the Hamming weight of c_{start} (i.e. the number of active bits in c_{start}) and let w_1 be the Hamming weight of $M(c_{end})$, where M is the matrix used in `MixColumns`. Finally denote by P_k, the probability of the round k, for $0 \leq k < r$. The probability of the r-round trail, that we will call from now on core trail, is then given by $P_0 \times P_1 \times \cdots \times P_{r-1}$.

4.2.2.3 Extending the core trail.

To build our attack, we need to choose an r-round trail that will be extended one round backward and half a round forwards as shown in Figure 4.2. In this section, we describe the criteria that we used to select an r-round trail that is likely to result in a good $(r+1.5)$-round trail. The resulting $(r+1.5)$-round trail must have good probability, but also needs to be adapted to our key recovery step. In particular, as we will argue in detail in Section 4.3, it is important to have differentials that will allow for efficient sieving in plaintext. In particular, it is desirable that the $(r+1.5)$-round trail we construct has a sufficient number of inactive rows on the plaintext.

First, as described above, we need to make sure that the r-round trail selected leads to a $(r+1.5)$-round trail with good probability. For a r-round trail, the probability of the resulting $(r+1.5)$-round trail can be upper-bounded by $2^{-(w_0+1)\times 3} \times P_0 \times P_1 \times \cdots \times P_{r-1} \times 2^{-w_1 \times 3}$. Indeed, if w_0 is the Hamming weight of c_{start}, then by computing backward one

round there will be at least $w_0 + 1$ active S-boxes. As the highest probability transition through an S-box has probability 2^{-3}, the highest possible probability of this prepended round will be $2^{-(w_0+1)\times 3}$. In the same way, if w_1 is the Hamming weight of $M(c_{end})$, then there will be exactly w_1 active S-boxes through the first S-box layer of the next round. Thus, the probability of the appended half round will be at most $2^{-w_1\times 3}$. We generated all possible r-round core trails following the procedure described above and kept the ones providing high estimated probabilities.

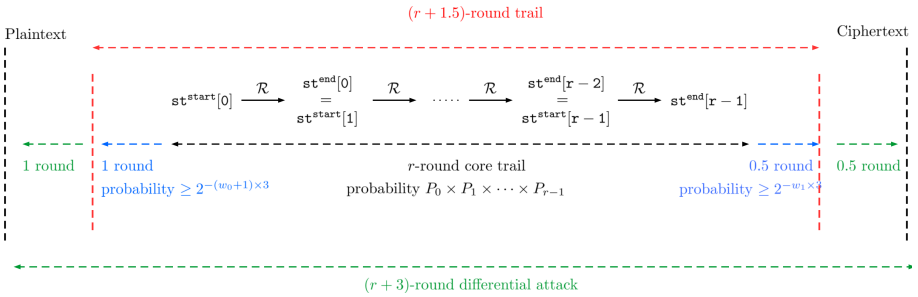
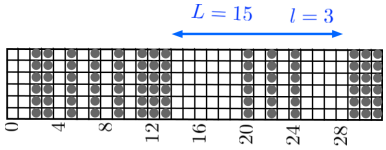


Figure 4.2: Generating $(r + 1.5)$-round trails from core r-round trails and extending them to mount $(r + 3)$-round attacks on SPEEDY.

Second, we want the r-round trail selected to lead to a $(r + 1.5)$-round trail that has a significant number of inactive rows on the plaintext for the sieving step to be efficient. First, consider the initial state of the r-round trail. The rows that are active in this state are exactly the rows that will be active in the state that follows the first SC operation in round 0 of the $(r + 1.5)$-round trail. To achieve better sieving, we want the transition from this state through $\mathtt{SC}^{-1} \circ \mathtt{SB}^{-1}$ to lead to an initial state of the $(r + 1.5)$-round trail that has low Hamming weight. To achieve this, not only the number of active rows but also the way those are distributed inside this state play a role in the efficiency of the sieving procedure. Let L be the size of a block of consecutive rows, where all rows are non-active except for l out of them. An example of such a state is shown below with $L = 15$ and $l = 3$.



Large values of L combined with small values of l naturally lead to better complexities. Indeed, we can carefully control the l active rows with some probability at a given cost. By doing so, we can generate several inactive rows in the plaintext as high as $L - 5 - l$, thus leading to a sieving of $2^{-[(L-5-l)]\times 6}$.

Using the above criteria, we selected an r-round trail, which we then extended in two ways, starting first by appending a round backward. This led to an $(r + 1)$-round trail. Then, to further improve the probability of our trail $(r + 1)$-round trail, we relied on the technique of multiple differentials.

4.2.3 Multiple differentials

The technique of multiple differentials consists in considering multiple $(r+1)$-round differential trails that all have the same input and output difference. To make the description of our technique simpler, we will describe how we built our multiple differentials in the case of our 7-round attack. In this case, $r = 4$. For our 7-round attack, the chosen 4-round core trail is the one displayed in red in Figure 4.3. This trail has probability $2^{-161.15}$. As shown in Figure 4.3, we extended it by one round backward and obtained a 5-round trail of probability $p_{main} = 2^{-170.56}$. We call this trail the *main trail*. Note that it is possible to extend the 4-round core trail backward with probability 2^{-6} for one round. However, this propagation, due to the diffusion properties of the inverse `MixColumns` transformation would lead to a column with 19 active bits (see Section 4.2.1.2). Such a scenario would have complicated the key-recovery phase and was not retained.

We limited our search to trails with probability smaller or equal to $p_{max} = p_{main} \times 2^{-25}$. Our new trails must thus verify that

- their input difference is such that the bits of coordinate $(i, j) \in \{2, 3, 4, 7, 8, 10, 12, 14, 16, 17, 18, 25, 27, 29\} \{1\}$ are active, whilst the other bits are inactive in the first state of Figure 4.3;
- their output difference is such the bits of coordinate $(i, j) \in \{1, 15, 16, 19, 21, 25, 31\} \times \{3\}$ are active, whilst the other bits are inactive in the second state surrounded by red in Figure 4.3.

To build our new trails, we rely on an algorithm that operates round by round.

Initial round. We start by building a list of potential initial one-round trails. We denote the initial state by `st[0]`, the state after the application of `MC` by `st[1]`, and so on and so forth as we did when constructing our one-round trails. We similarly construct our initial one-round trails to the way we constructed the one-round trails used to build our main trail. More precisely, we want our potential initial one-round trails to satisfy the following conditions:

- `st[0]` verifies the input condition;
- `st[5]` has a single active column c_5 such that $(c_5, M(c_5)) \in$ `T`;
- `st[2]` has at most two active bits per row.

To make the search more efficient, we added constraints on these initial round trails' probability and Hamming weight, using the fact that (`st[0]`, `st[6]`) must belong to a larger 5-round trail such that the probability of this larger trail is at most p_{max}. We will not describe these constraints in detail as they are very similar to previous techniques we used to build trails of reasonable probability. We obtained 6 potential initial round trails. Because of the second condition above, these new trails can be chained to our previously computed one-round trails. This property will be used to build our multiples.

Chaining the initial round. To find trails that satisfy our truncated differential constraints, we must now chain the potential initial round trails to the previously computed one-round trails. We do so in two steps for the chaining to be computationally feasible.

1. We chain the 2-round trails pre-computed to the potential initial one-round trails to form potential initial three-round trails. We get 8049 such 3-round trails.

2. We chain these potential initial 3-round trails to the previously computed 2-round trails to obtain 5-round trails.

We found 409 5-round trails that matched all our criteria. By adding their corresponding probabilities, we found a final probability of $2^{-169.95}$. As one can notice, using multiple differentials allows to improve the probability of the r-round differential, but this improvement is not as important as one would have expected by the number of found trails. This is because all of the additional trails found had unfortunately quite bad probabilities compared to the main one.

4.2.3.1 5.5-round differential trail

We describe now the 5.5-round differential trail we used to attack `SPEEDY-7-192` in the following section. This trail is depicted in Figure 4.3.

As stated before, the 5-round trail has probability $2^{-170.56}$, which is improved to $2^{-169.95}$ by using multiple trails. We then extended this differential 0.5 round forwards. For this step, we followed a particular approach. To go through the last S-box layer of the distinguisher part (see the before last state of Figure 4.3) an attacker has several choices. One extreme would be to fix to some concrete output value the transitions through all active S-boxes. This comes at a cost of a certain probability, but if we choose the transitions carefully we can guarantee very few active rows on the ciphertext. The other extreme is to consider truncated output differences for all the active S-boxes of this state. Thus the transition through the `SubBox` layer happens with probability 1, but almost all rows will be active in the output leading to very large structures of ciphertexts. What we decided to do is a trade-off between these two scenarios. More precisely, we decided to fix the transition `0x4` $\rightarrow$ `0x10` for the active S-boxes of rows 5, 11, and 19 and to allow more transitions for the S-boxes of rows 0 and 28. The choice of these two rows comes from the fact that after the `SC` operation, these two S-boxes activate some common rows. Our goal was to activate at most 7 rows after the `SC` operation (last state of Figure 4.3) and for this, we computed the highest probability to have at most 4 rows active between rows 23 and 31 and also row 0 after `SC`. We exhausted all possible configurations and we found the best one to be the one having the rows $24, 27, 28$ and 31 active after `SC`. One possibility for this was to force the output difference of the S-box of row 0 to be of the form `(0,*,0,0,*,0)` and the output difference of the S-box of row 28 to be of the form `(*,*,0,0,*,0)`, where `*` means that the corresponding bit is potentially active. The probability than to start from any difference of the above form in rows 0 and 28 and to activate at most the rows $24, 27, 28$ and 31 after the `SC` is $2^{-3.41}$.

This fact, together with the probability of $2^{-3.41}$ for the transition `0x4` $\rightarrow$ `0x10` for the other three active rows, gives a total probability of $2^{-13.64}$.

To summarize, as can be seen from Figure 4.3, our 5.5-round trail has then a total probability of

$$2^{-169.95} \times 2^{-13.64} = 2^{-183.59}.$$

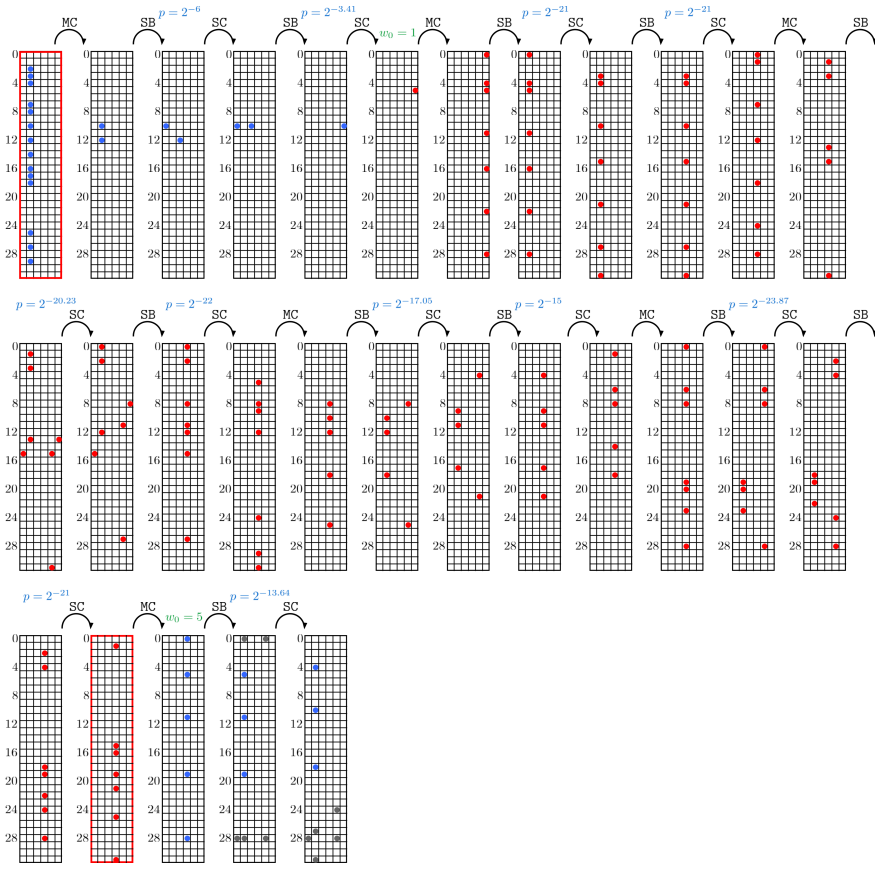


Figure 4.3: 5.5-round differential trail used to attack `SPEEDY-7-192`. The red part corresponds to the 4-round core trail, while the blue part corresponds to the 1.5-round extension. Grey bits are bits with unknown differences. The two states surrounded in red are the starting and final states of the multiple differentials considered.

4.3 Attack on `SPEEDY-7-192`

`SPEEDY-7-192` is the variant of the `SPEEDY` family suggested for applications where a security of 192 bits is needed. We show in this section, by using the techniques and ideas introduced earlier, how to recover the secret key of this version with less than 2^{192} encryptions. In addition, we will propose two ideas that will allow us to optimize the

complexity of the attack: one, already used for instance in [Bro+22b], is to not consider the rounds as blocks regarding their treatment concerning the differential distinguisher or the truncated part, but include some row transitions in the differential and let the rest go as truncated in the same round which we will apply in the input and output of the attack; the other is to consider the detailed equations over two rounds with merging techniques that will allow us to optimize the complexity of the key guessing part.

Our attack has a data complexity of $2^{187.28}$, a time complexity of $2^{187.84}$ and a memory complexity of 2^{42} and contradicts thus the designers' security claim for this variant, as has been acknowledged by them. More importantly, this cryptanalysis highlights that the security margin for this variant was overestimated. Our attack uses the differential found with the ideas from Section 2.1 and the implemented method described in Section 4.2.2. As described before, the main differential trail depicted in Figure 4.3 covers 5.5 rounds, and its probability, when taken together with its associated multiple trails described in Section 4.2.3, is $2^{183.59}$. The trail of Figure 4.3 can then be extended one round backward and half a round forwards as shown in Figure 4.4, to finally cover 7 rounds. This fact contradicts a particular statement of the designers that wrote that a one-round security margin for the key-recovery part should be sufficient.

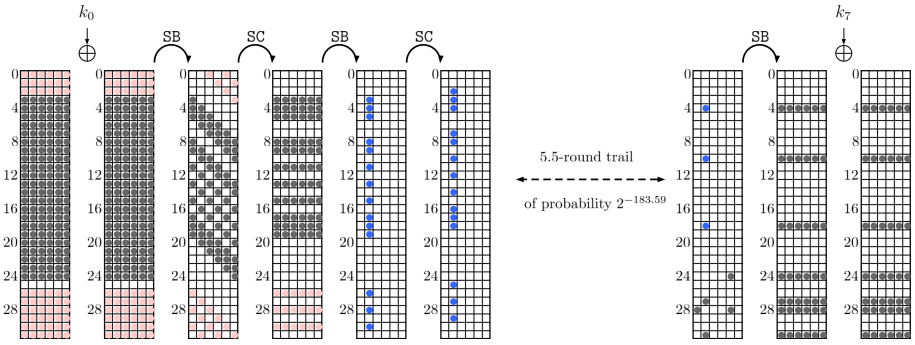


Figure 4.4: Key recovery part of the 7-round attack against SPEEDY-7-192

4.3.1 Trade-off between differential probability and efficient sieving

Our attack is performed in the decryption direction. The first step is to generate several relevant ciphertexts to implement the attack. If we impose no extra condition on the extension of the distinguisher to the plaintexts ($\Delta_{in} \to D_{in}$ as denoted in Figure 2.1.2) then D_{in} will have all but one rows active (see Figure 4.3). This would lead to very limited sieving and would thus leave us with too many potential pairs on which to perform the key recovery. For this reason, we propose a first improvement. This improvement consists in restricting the permitted transitions through the second S-box layer of Round 0. More precisely, the condition is that the three active bits in rows 26, 28 and 30 after the second S-box only generate a maximum of three active rows in the plaintext state

(among rows 26 to 31 and among rows 0 to 2). This condition allows to have 7 inactive rows (instead of 1 before) in the plaintext state at the cost of decreasing the overall differential probability. We denote by P_{in} the probability that it is verified. As we show next, since this probability is relatively high, the impact on the overall differential probability is limited.

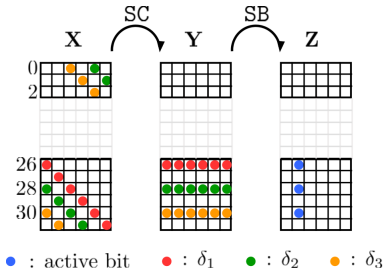


Figure 4.5: Transition of rows 26, 28 and 30 through the inverse of the second SB of round 0.

To compute P_{in}, we start with the state $\mathbf{Z}$, corresponding to the state after the second S-box application of Round 0, where the rows 26, 28 and 30 all have an active difference of 010000. Therefore, on the state $\mathbf{Y}$, we consider differences $\delta_1, \delta_2, \delta_3$ that propagate to 010000 through the S-box layer with probability $\mathbb{P}(\delta_1), \mathbb{P}(\delta_2), \mathbb{P}(\delta_3)$ respectively. Propagating backwards through SC, we obtain $\mathbf{X}_{\delta_1,\delta_2,\delta_3} = \mathtt{SC}^{-1}(\mathbf{Y}_{\delta_1,\delta_2,\delta_3})$. We are interested in states $\mathbf{X}_{\delta_1,\delta_2,\delta_3}$ that have at most three nonzero rows among rows 26 to 31 and rows 0 to 2. We define the function $\mathbb{1}_3$ as follows:

$$\mathbb{1}_3(\mathbf{X}_{\delta_1,\delta_2,\delta_3}) = \begin{cases} 0 \text{ if } \mathbf{X}_{\delta_1,\delta_2,\delta_3} \text{has more than 3 nonzero rows} \\ 1 \text{ else.} \end{cases}$$

The overall probability for the transition is given by the formula

$$P_{in} = \int_{\delta_1,\delta_2,\delta_3} \mathbb{1}_3(\mathbf{X}_{\delta_1,\delta_2,\delta_3}) \mathbb{P}(\delta_1)\mathbb{P}(\delta_2)\mathbb{P}(\delta_3)\,.$$

The obtained probability is $P_{in} = 2^{-2.69}$. We take this probability into account as part of the overall probability of the differential distinguisher, which now becomes $2^{p^*} = 2^{-(183.59+2.69)} = 2^{-186.28}$. Note that only 78 (instead of $\binom{9}{3} = 84$) difference patterns are possible in the plaintext.

4.3.2 Data generation

We build the data required for our attack in the decryption direction. Since there are 7 active rows on the ciphertexts, the size of each structure is $2^{7\times 6} = 2^{42}$. By following now the notations introduced in Section 2.1, we build 2^s structures of size 2^{42} each, such that 2^{s+42} equals $2^{186.28+1}$. This implies that there are $2^s = 2^{145.28}$ structures and $2^{145.28+2\cdot 42-1} = 2^{228.28}$ potential pairs. The cost of the data generation is $2^{187.28}C_E$, where

C_E is the cost of one encryption and can be estimated as $6*(1+6+6+6)+1+6+6 = 128$ bit-operations. Indeed, MC, SB are 6 bit-operations, the cost of AK is 1, and all these transformations can be applied in parallel.

4.3.3 Sieving of the pairs

Performing the key-recovery step on all $2^{228.28}$ pairs would exceed the complexity of the exhaustive search. Therefore, we will start with a sieving step to eliminate pairs that cannot have followed the differential. This sieving is done by looking at the differences in the plaintext. As can be seen from Figure 4.4, the 'good' plaintext pairs have a zero-difference in row 25 as well as 6 inactive rows among rows 26 to 31 and 0 to 2. The sieving will be performed on both the inactive and the active rows.

Inactive rows. Each inactive row represents a 6-bit filter. We consider each of the 78 possible difference patterns in the plaintext. For each pattern, since there are 7 inactive rows at the input, the sieving obtained from these rows is 2^{-42}.

Active rows $[3-24]$**.** We can proceed to sieve on each of these 22 active rows by taking into account the first S-box layer of Round 0. To make this step clear, we start by explaining the sieve on row 6. As can be seen from Figure 4.4, to follow the differential, a plaintext pair should generate after the application of the S-box a truncated difference of the form (0,*,*,*,0,0). By looking at the DDT of SPEEDY's S-box, we see that the input differences 0x16, 0x2d and 0x3c never propagate to an output difference of the form (0,*,*,*,0,0). Thus, any pair with one of those three plaintext differences at row 0 can be sieved out. This gives us a filter of $log_2(61/64) = 0.07$, as shown in Table 4.5. The filters for the other active rows are computed similarly and are reported in Table 4.5.

row	filter	row	filter	row	filter	row	filter
3	0.42	9	0.02	15	0.09	21	0.07
4	0.48	10	0.05	16	0.07	22	0.17
5	0.07	11	0.07	17	0.09	23	0.51
6	0.07	12	0.12	18	0	24	1.42
7	0.07	13	0.02	19	0.02		
8	0	14	0.07	20	0		
total filter 3.9							

Table 4.5: Sieving in the active rows $[3-24]$ of the plaintext.

Considering the 78 **different patterns in the rows [26-31] and [0-2].** Recall that there are in total 78 possible patterns *pat*, and each one corresponds to a subset of

exactly 3 active rows among rows 26 to 31 and 0 to 2 in the plaintext. We start from the difference (0,1,0,0,0,0) on the rows 26, 28 and 30 after the second S-box of Round 0. Then we propagate this difference backward through the two S-box layers of Round 0 and discard all the differences that do not follow the pattern considered. The number of possible differences on the plaintext allows us to filter $2^{-f_{pat}} = \frac{\#\text{Possible differences}}{2^{3\cdot 6}}$ (see [Bou+23a] for the 78 possible values of f_{pat}).

Summarizing the sieving step. For a pattern *pat*, the sieving corresponding to the inactive rows is 2^{-42} while the one on the active rows is $2^{-3.9} \cdot 2^{-f_{pat}}$. Thus, the total number of potential pairs for the key recovery step is

$$\sum_{pat} 2^{228.28-45.9-f_{pat}} = 2^{182.38} \sum_{pat} 2^{-f_{pat}} = 2^{186.42}\,.$$

This sieving step is the reason why we decided to perform the attack in the decryption direction. Indeed, using the 78 patterns in initial structures would have further increased the complexity.

4.3.4 Recovering the key

In this section, we describe our improved key recovery step. The key recovery algorithm is performed for each pair on the fly. As explained in the last section, the total number of pairs we will try in this step is $2^{186.42}$. For each pair, we check whether there exists a key that allows the pair to follow the differential. If not, the pair is discarded. Otherwise, as we will show, we obtain a partial key on which all bits are determined but a small number n_l which is equal to 8 on average. For each of the remaining pairs and associated partial key, we then try exhaustively all possible 2^{n_l} keys. For each pair, the key recovery is divided into three stages which can be summarized as follows. First, we determine bits of the last subkey k_7 using the fact that if the pair follows the trail, then it must belong to δ_{out} before the last SB application. Since the key schedule of SPEEDY consists simply of a permutation of the key bits, this, in turn, constrains the bits of k_0. Second, we determine more bits of k_0 using the fact that the pair must belong to δ_{in}. Lastly, we determine a few extra key bits using the penultimate S-box application (the first S-box application of the last round).

4.3.4.1 Stage 1 - Last subkey addition (k_7).

For each pair, we start by determining several bits of k_7. As can be seen from Figure 4.4, the ciphertext pairs are active on the rows [4,10,18,24,27,28,31]. For the rows [4,10,18,27,31] (respectively row 24), we want the partial key to be such that these rows satisfy the differential (0,1,0,0,0,0) (respectively (0,0,0,0,1,0)) before the last SB application. For each pair, this determines $6 \times 6 = 36$ key bits on average. The case of row 28 is only slightly different. If active, there are 2^6 possibilities for the six key bits, but 4 different patterns are possible before SB. A correct pattern is thus reached

row							row						
4	145	8	63	118	173	36	27	55	110	165	28	83	138
10	13	68	123	178	41	96	28	1	56	111	166	29	84
18	157	20	75	130	185	48	31	31	86	141	4	59	114
24	25	80	135	190	53	108							

Table 4.6: Guessed master key bits from the subkey k_7. Each row corresponds to one of the 7 active rows of the ciphertexts.

with probability 2^{-4}. The row 28 can thus determine 6 additional key bits at the cost of 2^2 guesses on average. This stage thus allows us to determine up to 42 key bits at the cost of 2^2 guesses. In Table 4.6, we detail which key bits of the master key are fixed after determining the value of k_7 on the rows corresponding to one of the 7 active rows in the ciphertext.

About the potentially active ciphertext rows. For the sake of simplicity, we consider in this analysis that the four ciphertext rows [24,27,28,31] are active, as it simplifies the key guessing procedure. We could just discard the pairs that do not verify this, leaving us with $2^{24+24-1} - 2^{47-6} \approx 2^{46.91}$ pairs for the partial structure on 4 lines instead of 2^{47}, but with a higher probability of reaching a good difference before the penultimate SB. In practice, there is no need to discard this data. It can also be treated with similar methods to the one presented here. Although these methods are slightly more expensive than the one presented here as a few more key bits might have to be partially guessed, they are used to handle a very small proportion of data. Thus, the difference in the cost will be negligible. We thus limit our explanations to the predominant case, with all the rows active. We allow rows [4,10,18] to be nonactive. For each of these rows, this gives on average a probability of 2^{-6} of having a difference that can match the required one (including the 0 difference). Thus, on average, only one 6-bit key word leads to the desired difference.

4.3.4.2 Stage 2 - First subkey addition (k_0)

We now focus on the addition of the first subkey k_0. This key recovery stage is performed row by row, and the order in which each row is treated is important to keep our time complexity as low as possible. For each row, we will use available information from both SubBox layers of Round 0 to determine more triplets of possible pairs and associated keys. Table 4.7 helps us understand how to exploit the first S-box of Round 0 for rows [3,...,24]. Recall that these rows are active in the plaintext and that they allowed us to perform a specific sieving given in Section 4.3.3. For each row, Table 4.7 provides the following information:

- Key determined gives the number of key bits already determined during Stage 1 (*i.e.* with subkey k_7).

- `Key left` gives the number of key bits that remain to be determined for this row (note that the sum of `Key determined` and `Key left` is always 6).
- `Differential Filter` gives the value of the filter that was applied during the sieving step to each pair.
- `Fixed bits` gives the amount of inactive bits after the first `SubBox` layer.
- `First S-box Cost` gives the overall cost in bits for a given row to check the propagation through the first `SubBox`. Since one can precompute the valid pairs of values and associated partial keys for each row, this cost is equal to (`Key left` + `Differential filter` - `Fixed bits`) rather than `Key left`. For each row and each key, the probability that they satisfy the differential is $2^{\texttt{Differential filter}-\texttt{Fixed bits}}$. In particular, for rows where the value of `First S-box Cost` is negative, then for each pair, there exists a key that satisfies the differential with probability < 1. Such rows allow us to discard more pairs.

Since row `25` is inactive, it does not provide any information about k_0 through the first `SB`. For the sake of simplicity, we do not analyze how to exploit the rows `0` to `2` and `26` to `31`. This could have been done by looking at the case of each specific pattern, but it wouldn't have significantly improved the attack whilst considerably lengthening the description of the key recovery step.

To perform the key recovery, we will also look into the propagation through the second S-box. More precisely, we will use the conditions set on the rows `[3,4,5,8,9,11,13,15,17,18,19]` after the application of the second S-box to sieve the pairs. At the output of the second S-box, these rows must have the exact difference `(0,1,0,0,0,0)`. This provides us with a 2^{-6} filter, but it is not straightforward how to exploit it. Indeed, because of the `SC` step, each row at the output of the second S-box layer depends on 6 rows at the output of the first S-box layer. It thus seems that to get a 2^{-6} filter, one first has to guess 6 rows of k_7, which is very costly. However, we use several improved techniques to get filters without having to guess too many rows before the first S-box step. We describe these techniques through an example which can be found in the paragraph dedicated to the rows `[18,19,22,21,23]` below. We now describe in detail the first three steps of Stage 2. Table 4.8 sums up the rest of Stage 2.

Rows `[4]`. We start by considering row `4`. This row allows us to perform a filter of -0.52 at the first S-box level of Round 0.

Rows `[18,19,20,21,23]`. We next consider the rows `18,19,20,21` and `23`. To understand why these rows are the next ones we consider, one must take into account the second S-box transition. Indeed, consider the rows `[17,18,19]` after the second S-box transition. These three rows are active, and must thus have the exact difference `(0,1,0,0,0,0)`. Since these rows are positioned next to each other, one does not need to guess $3 \times 6 = 18$ rows at the input of the first S-box, but only 8, namely the rows `[17,18,...,24]`. Further, we show that to get a filter, one does not need to guess all

row							Key determined	Key left	Differential Filter	Fixed bits	First S-box Cost
0	0	1	2	3	4	5	2	4	*	*	*
1	6	7	8	9	10	11	1	5	*	*	*
2	12	13	14	15	16	17	1	5	*	*	*
3	18	19	20	21	22	23	1	5	0.42	4	1.42
4	24	25	26	27	28	29	3	3	0.48	4	-0.52
5	30	31	32	33	34	35	1	5	0.07	3	2.07
6	36	37	38	39	40	41	2	4	0.07	3	1.07
7	42	43	44	45	46	47	0	6	0.07	3	3.07
8	48	49	50	51	52	53	2	4	0	2	2
9	54	55	56	57	58	59	3	3	0.02	2	1.02
10	60	61	62	63	64	65	1	5	0.05	3	2.05
11	66	67	68	69	70	71	1	5	0.07	3	2.07
12	72	73	74	75	76	77	1	5	0.12	3	2.12
13	78	79	80	81	82	83	2	4	0.02	2	2.02
14	84	85	86	87	88	89	2	4	0.07	3	1.07
15	90	91	92	93	94	95	0	6	0.09	3	3.09
16	96	97	98	99	100	101	1	5	0.07	3	2.07
17	102	103	104	105	106	107	0	6	0.09	3	3.09
18	108	109	110	111	112	113	3	3	0	2	1
19	114	115	116	117	118	119	2	4	0.02	2	2.02
20	120	121	122	123	124	125	1	5	0	2	3
21	126	127	128	129	130	131	1	5	0.07	3	2.07
22	132	133	134	135	136	137	1	5	0.17	3	2.17
23	138	139	140	141	142	143	2	4	0.51	4	0.51
24	144	145	146	147	148	149	1	5	1.42	5	1.42
25	150	151	152	153	154	155	0	6	*	*	*
26	156	157	158	159	160	161	1	5	*	*	*
27	162	163	164	165	166	167	2	4	*	*	*
28	168	169	170	171	172	173	1	5	*	*	*
29	174	175	176	177	178	179	1	5	*	*	*
30	180	181	182	183	184	185	1	5	*	*	*
31	186	187	188	189	190	191	1	5	*	*	*

Table 4.7: This table represents the information used for efficiently solving the key-recovery part of the attack. Each line in the table is associated with the same row in the state. The column `Key determined` indicates how many bits are already known from Stage 1 (those bits are depicted in red), and `Key left` is the number of bits that remains to be known. `Fixed bits` represents the number of inactive bits after the first `SB` of Round 0 and that therefore can be used to perform a sieving on the candidate keys. The `cost` is the difference between the previous values, and the `second filter` denotes the active rows in the second `SB`, as they will provide additional filtering to produce the fixed output difference.

of the 8 rows on which the rows `[17,18,19]` after the second S-box transition depends. We start by precomputing all the pairs of values that are in the codomain of the function

$$a, b, c, d, e, f \mapsto \left(S^{-1}(a, b, c, d, e, f), S^{-1}(a, b, c, d, e \oplus 1, f)\right)$$

and store them in a table of size 2^6. As depicted in Figure 4.3.4.2, we can perform a match using these tables on some of the bits before the second S-Box by computing the values of some rows (not necessarily all of them). We now guess the rows `[18,19,20,21,23]` at the entry of the first S-box. In total, 13 bits of the rows `[18,19,22,21,23]` later impact the rows `[17,18,19]` at the entry of the second S-box. There are thus 2^{26} possible pairs of values for these bits, whilst in total, the table contains 2^{18} possible pairs that verify the condition at the output of the S-box on the rows `[17,18,19]`. This thus results in a 2^{-8} filter. More precisely each pair matches a pre-computed valid pair in the table of size 2^{18} with probability 2^{-8}. In particular, whenever a pair is not discarded, the rows `[17,18,19]` before the second S-box are completely determined. This will allow us to filter more pairs as we guess more rows in the plaintext which impact the value of the rows `[17,18,19]` before the second S-box. Regarding the guess of rows `18,19,21` and `23`, since the transition relation for the rows `[17,18,19]` can be decomposed in three relations that each checks the transition for a row, we can use a more efficient technique for the guessing: *the parallel matching*. This technique developed in [CNV13a] allow to reduce the guessing cost if:

- The data is presented in two lists: $\mathcal{L}_A, \mathcal{L}_B$
- We want to recover $S\{(a, b) \in \mathcal{L}_A \times \mathcal{L}_B | R(a, b)\}$ for a relation R that can be decomposed as follows.

  $$R(a, b) \Leftrightarrow \forall i \; R_i(a_i, b_i)$$

  where (a_i) (resp. (b_i)) form a partition of the bits of a (resp. b).

Since this first guess fulfill the requirements, we obtain a reduction of the guessing cost from $2^{2.02+2.07+1+0.51+2.17} = 2^{7.77}$ to $2^{4.09} + 2^{3.68} + 2^{7.77-8} = 2^{4.94}$, or else can be more efficiently performed with small precomputations regarding these partial transitions with a cost for each step given by the number of remaining solutions, so $2^{7.77-8} = 2^{-0.23}$ in this example.

Row `24`. The next step consists in guessing row `24`. As we have described previously, for the pairs that have not been discarded yet, the three rows `[17,18,19]` before the second S-box are fixed. Two bits of row `24` later impact the value of these rows. Thus, we obtain an extra 2^{-2} filter. Therefore, as can be seen from Table 4.7 we obtain a partial guessing cost of $2^{1.42}$ and a partial data cost of $2^{-0.58}$.

The next steps of the key recovery are described in Table 4.8. This table must be read from top to bottom. Its columns provide the following information:

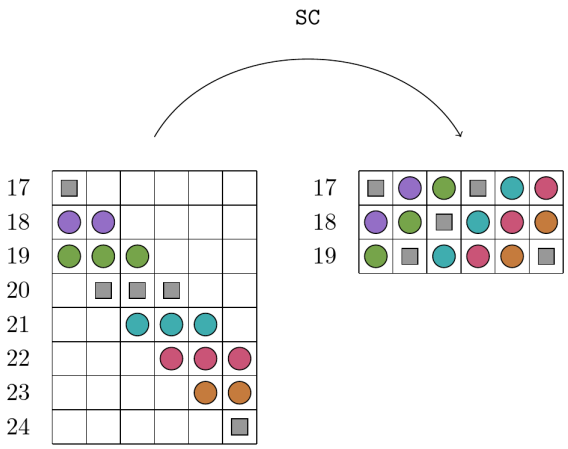


Figure 4.6: Parallel key guessing for Rows `[18,19,20,21,23]`.

- `Row guessed at the input`. This column displays the coordinate of the row guessed. The column considered first is at the top and the last one is at the bottom.
- `Partial guessing cost`. This column displays the cost of guessing each row and checking that the first S-box transition is valid (See Table 4.7).
- `Partial data filter`. For each row, this column displays the log of the probability that a valid partial key exists for each pair, taking into account the filter provided by the constraints after the second S-box. This column provides information on the evolution of the data after guessing each row (or group of rows). For a (group of) row(s), if the entry on this column is $-x$, then the number of pairs remaining after handling this (these) column(s) is multiplied by 2^{-x}.
- `Row determined at second S-box`. This column displays the second S-box rows that are fully determined after a given guess.

The complexity of the key recovery so far is given by the formula

$$\left[2^{186.42+2}2^{-0.52}(2^{4.94}+2^{-0.23}(2^{1.42}+2^{-0.58}(2^{3}+2^{-3}(\cdots(2^{1.42}+2^{0.58}))))\right]2^{-7}C_E$$
$$= 2^{186.15}C_E$$

and there are $2^{168.3}$ remaining pairs.

4.3.4.3 Stage 3 - Back to k_7 using the penultimate S-box.

For the remaining key bits, we will go back to k_7 and study the penultimate S-box. A similar approach to the second S-box is applied here: instead of using the second S-box transition to perform a guess and filter approach, the penultimate S-box is used. For each row of k_7, Table 4.9 shows which bits of the master key still need to be guessed

Row guessed at the input	Partial guessing cost	Partial data filter	Row determined at second S-box
4	-0.52	-0.52	
18,19,22,21,23	4.9	-0.23	17,18,19
24	1.42	-0.58	
20	3	-3	15
17	3.09	-0.91	
16	2.07	0.07	13
15	3.09	-0.91	
14	1.07	-0.93	11
13	2.02	-1.98	
11	2.07	0.07	9
12	2.12	-1.88	8
9	1	-3	
10	2.05	-1.95	
8	2	0	5
6	1.07	-0.93	3
5	2.07	-1.93	
7	3.07	-0.93	
3	1.42	-0.58	

Table 4.8: Description of Stage 2 of the key recovery.

(the bits in black). For each pair, we wish to find partial keys such that they lead to the difference 000100 before the first S-box of the last round on rows [0,5,11,19,31]. For each of these rows, Table 4.10 displays which rows of k_7 need to be guessed to check the transition to 000100 before the first S-box of the last round. More precisely, it provides the following information.

- Row considered. This column displays the coordinate of the row before the first S-box of the last round which will be used to filter the right key guesses.
- Involved rows of k_7. This column displays which rows of k_7 must be guessed to check the condition on the row considered before the first S-box of the last round.
- Number of missing bits. This column displays the number of bits in the involved rows of k_7 that have not yet been determined.

For each remaining pair and each of these five transitions, we recover the key bits that allow this transition and put them in tabs. By merging them, we then recover the pairs and associated partial keys that allow the whole state transition. After this step, only the key bits [15,16,152,153,159,180,187,188] are left to determine. This is done by guessing them. The complexity associated with this step is

$$2^{168.3}(2^8 + 2^8 + 2^9 + 2^{10} + 2^{10} + 2^{11}(1 + 2^8)) = 2^{180.31} C_E \,.$$

row							Key left	row							Key left
0	169	32	87	142	5	60	2	16	73	128	183	46	101	156	2
1	115	170	33	88	143	6	2	17	19	74	129	184	47	102	1
2	61	116	171	34	89	144	1	18	157	20	75	130	185	48	0
3	7	62	117	172	35	90	2	19	103	158	21	76	131	186	2
4	145	8	63	118	173	36	0	20	49	104	159	22	77	132	1
5	91	146	9	64	119	174	2	21	187	50	105	160	23	78	1
6	37	92	147	10	65	120	1	22	133	188	51	106	161	24	1
7	175	38	93	148	11	66	2	23	79	134	189	52	107	162	2
8	121	176	39	94	149	12	2	24	25	80	135	190	53	108	0
9	67	122	177	40	95	150	2	25	163	26	81	136	191	54	2
10	13	68	123	178	41	96	0	26	109	164	27	82	137	0	2
11	151	14	69	124	179	42	3	27	55	110	165	28	83	138	0
12	97	152	15	70	125	180	3	28	1	56	111	166	29	84	2
13	43	98	153	16	71	126	2	29	139	2	57	112	167	30	2
14	181	44	99	154	17	72	3	30	85	140	3	58	113	168	2
15	127	182	45	100	155	18	2	31	31	86	141	4	59	114	0

Table 4.9: In red, the master key bits that have already been determined. In black, the bits that still need to be determined.

Complexity summary. The final time complexity of our attack is

$$\mathcal{T} = 2^{187.28} C_E + 2^{179.42} C_E + 2^{186.15} C_E + 2^{180.31} C_E = 2^{187.84} C_E$$

4.4 Conclusion

In this chapter, we managed to perform a full break on SPEEDY-7-192 which was presented in TCHES 2021. Although differential cryptanalysis is a well-known technique that designers consider when choosing their parameters, designing tight security claims is a difficult task. Indeed, the attack presented in this chapter contradicted both the

Row	Involved rows of k_7	# missing bits	Row	Involved rows of k_7	# missing bits
0	27,28,29,30,31,0	8	19	14,15,16,17,18,19	10
5	0,1,2,3,4,5	9	31	23,24,25,26,27,28	8
11	6,7,8,9,10,11	10			

Table 4.10: Description of Stage 3 of the key recovery.

security claims derived by the security parameters as well as the security claim on the number of key recovery rounds possible. This is possible for two reasons:

- There is a lot of finesse in the design of the attack. Indeed, the trade-offs used in this attack allow a balance of the complexity terms, hence minimizing the overall complexity. Also, the key recovery is very efficient thanks to the use of guessing techniques such as *parallel key guessing*.
- The branch-and-bound algorithm used by the designers of `SPEEDY` does not capture well the spectrum of differential distinguishers. Indeed, they only considered one-to-one transitions through the S-Box hence neglecting the cases where two active bits on the same row recombine into one active bit.

Chapter 5
Differential Meet-In-The-Middle Cryptanalysis

This chapter is a joint work with Christina Boura, Patrick Derbez, Gregor Leander, and María Naya-Plasencia. These results published in [Bou+23b] present a new cryptanalysis technique: *differential meet-in-the-middle cryptanalysis* as well as an application to `SKINNY-128-384` resulting in new best attack in the single tweakey setting, and an attack on `AES-256` which implies the best-known attack in the related key scenario when only two related keys are used.

Contents

In Section 5.1, we will describe the new cryptanalysis technique and improvements as well as compare it with both differential cryptanalysis and meet-the-middle cryptanalysis. Then in Section 5.2, we will present an application of the differential-meet-in-the-middle technique to `SKINNY-128-384`. Hence, we will first give the specification of the `SKINNY` blockcipher family and then we will describe the attack on 25-rounds `SKINNY-128-384`.

5.1 The new attack: Differential MITM

In this chapter, we present a new cryptanalysis technique in symmetric cryptography: *differential meet-in-the-middle cryptanalysis*. It consists in combining elements of

differential cryptanalysis with elements of meet-in-the-middle attacks. From a differential point of view, this new technique can be interpreted as a new method to perform the key recovery. From the meet-in-the-middle point of view, differential MITM attacks consist of a method to extend MITM attacks by adding some rounds in the middle (similar to sieve-in-the-middle [CNV13b; CNV13c]). We will present in this section a high-level description of the new technique. More precisely, we will provide a general framework that describes how to mount a differential MITM attack in a generic and simple way, and we will show how to combine this generic method with two techniques: the parallel treatment of data partitions in order to add one round mostly for free (idea that inherits from MITM attacks, like bicliques [DH77b], and that cannot be applied to classical differential attacks), as well as a technique to reduce the data complexity.

5.1.1 General framework

Consider a n-bit cipher E decomposed into three sub-ciphers: $E_{out} \circ E_m \circ E_{in}$, as depicted in Figure 5.2. Let the number of rounds of E_{in}, E_m and E_{out} be r_{in}, r_m and r_{out} respectively. Finally, let Δ_{in} be the input difference to the middle part E_m, Δ_{out} the output difference of E_m and suppose that the differential $\Delta_{in} \to \Delta_{out}$, covering the r_m middle rounds, has probability 2^{-p}.

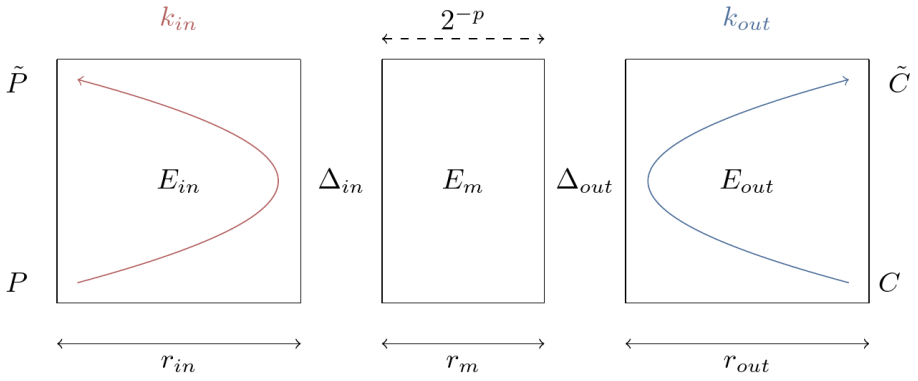


Figure 5.1: A high-level description of the Differential MITM technique.

We start our analysis with a first randomly chosen plaintext P of n bits and its associated ciphertext C, and we aim at generating a second plaintext-ciphertext pair $(\widetilde{P}, \widetilde{C})$ such that together they satisfy the differential on the middle rounds. Our new idea is to generate $(\widetilde{P}, \widetilde{C})$ with a meet-in-the-middle approach. For this, candidate plaintexts $\widetilde{P}$ are computed from both the plaintext P and the difference Δ_{in} for each possible value of the associated key K_{in}, while, in parallel, candidate ciphertexts $\widetilde{C}$ are computed from C and Δ_{out} for each possible value of the involved key, K_{out}. The match is then performed on the relation $E(\widetilde{P}) = \widetilde{C}$ (or $\widetilde{P} = E^{-1}(\widetilde{C})$) that happens with probability 2^{-n}. Note that the roles of the upper and lower part can be interchanged without loss of generality in order to optimize the data and memory complexity if we consider that access to both the encryption and the decryption oracles is granted.

Upper part. Given P, the aim is to guess the minimal amount of key information, that we will denote by K_{in}, such that we can compute the associated $\widetilde{P}$ that ensures $E_{in}(P) \oplus E_{in}(\widetilde{P}) = \Delta_{in}$. For each guess i for K_{in}, we obtain a different candidate for $\widetilde{P}$, that we denote by $\widetilde{P}^i$, leading to a total of $|K_{in}| = 2^{k_{in}}$ such values. From them, we can compute the $2^{k_{in}}$ associated ciphertexts $\widehat{C}^i = E(\widetilde{P}^i)$ with calls to the encryption oracle and store them in a hash table H.

Lower part. Similarly, given C, we can guess some key material K_{out}, of size $2^{k_{out}}$, and compute a new ciphertext $\widetilde{C}$ that satisfies the equation $E^{-1}_{out}(C) \oplus E^{-1}_{out}(\widetilde{C}) = \Delta_{out}$ if the key guess is correct. We obtain $2^{k_{out}}$ values for $\widetilde{C}^j$, each associated to a guess j for K_{out}.

Number of pairs and match. For the correct key guess, the transition $\Delta_{in} \to \Delta_{out}$ will happen with a probability 2^{-p}. Therefore, we will repeat the upper and lower procedures 2^p times with 2^p different messages P_ℓ so that we can expect one pair $(P_\ell, \widetilde{P}^i_\ell)$ to satisfy the differential together with the associated pair $(C_\ell, \widetilde{C}^j_\ell)$. When this is the case, we will find a collision for a certain ℓ between a $\widehat{C}^i_\ell$ computed in the upper part and stored in H and a $\widetilde{C}^j_\ell$ computed from the lower part. Each collision (i, j) has an associated key guess $K_{in} = i, K_{out} = j$, that we will consider as a potential candidate. The number of expected collisions for each fixed P_ℓ is $2^{k_{in}+k_{out}-|K_{in}\cap K_{out}|-n}$, hence resulting in a total of $2^{k_{in}+k_{out}-|K_{in}\cap K_{out}|-n+p}$.

Algorithm 5.1 Differential MITM attack

```
while right key not found do                          ▷ 2^p trials expected
    Randomly pick P
    C ← E(P)                                          ▷ Oracle call
    H ← ∅                                             ▷ hash table initialization
    for each guess i for K_in do                      ▷ Forward computation
        Compute P̃^i from i and P
        Ĉ^i ← E(P̃^i)                                  ▷ Oracle call
        H[Ĉ^i] ← H[Ĉ^i] ∪ {i}
    for each guess j for K_out do                     ▷ Backward computation
        Compute C̃^j from j and C
        for each i ∈ H[C̃^j] do
            Complete (i, j) to retrieve the master key
            Try candidates against extra data
```

Complexity. This attack can be decomposed in two steps, first, we look for key candidates then with the help of extra data, we check for each key candidate whether it stays valid or if it needs to be discarded. The time complexity of this attack can be estimated as

$$\mathcal{T} = 2^p \times (2^{k_{in}} + 2^{k_{out}}) + 2^{k_{in}+k_{out}-|K_{in}\cap K_{out}|-n+p},$$

where the first term corresponds to the first step, hence computations done in E_{in} and E_{out}, and the last one to the second step, hence number of expected key candidates. With this, we recover $K_{in} \cup K_{out}$, so if we expect fewer key candidates than the whole set $K_{in} \cup K_{out}$, (i.e $k_{in} + k_{out} - |K_{in} \cap K_{out}| - n + p < |K_{in} \cup K_{out}|$, which holds as long as $p < n$), we can guess the remaining bits of the master key and test the guess with additional pairs. Thus we recover the whole key with a complexity smaller than the cost of an exhaustive key search, and an additional cost of

$$2^{k-(|K_{in} \cup K_{out}|)} \times \max\{1, 2^{k_{in}+k_{out}-|K_{in} \cap K_{out}|-n+p}\}$$

to be added to the time complexity $\mathcal{T}$. In the expected case where $k_{in} + k_{out} - |K_{in} \cap K_{out}| - n + p \geq 0$, the total time complexity is thus

$$\mathcal{T} = 2^p \times (2^{k_{in}} + 2^{k_{out}}) + 2^{|K_{in} \cup K_{out}|-n+p} + 2^{k-n+p}.$$

The (naive) data complexity of this first version of the attack can be estimated as

$$\mathcal{D} = \min(2^n, 2^{p+\min(k_{in},k_{out})})$$

since the data complexity cannot exceed the size of the codebook. Finally, the naive memory complexity is given by $\mathcal{M} = 2^{\min(k_{in},k_{out})}$, though it can be improved to $2^{\min(k_{in}-|K_{in} \cap K_{out}|, K_{out}-|K_{in} \cap K_{out}|)}$ by first guessing the common key material before running the attack.

5.1.2 Improvement: Parallel partitions for layers with partial subkeys

In this section, we will consider the case where the round key addition does not affect the whole state but only a part of it (as is for instance the case in Feistel constructions [Fei73], or in the `SKINNY` [Bei+16] and `GIFT` [Ban+17] ciphers), we denote by m the number of bits of the states that are affected by the key addition. In this scenario, it possible to extend the attack by one round with minimal increase in complexity. Indeed, the time complexity will not be affected as long as $m < p$, the data complexity has to be checked case-by-case as it might depend on the configuration of the differences in the external states (further techniques for reducing this complexity is discussed in Section 5.1.3), the memory complexity might be increased.

The main idea here is to consider families of 2^m ciphertexts C together with families of 2^m penultimate states X_{R-1} such that for each element of both families there exists a final round key κ_{final} that turns X_{R-1} into C after one round. Using both of these families, we will perform a differential-MITM attack and retrieve the last round key a posteriori.

Since we expect to try on average 2^p plaintexts in order to find one that will satisfy the differential of probability 2^{-p}, we can divide the final state of size 2^n into two parts. The part without the key addition will take 2^{p-m} different values. Hence, the attack will start from a given $\mathcal{C}$ and be repeated for each one of the 2^{p-m} values.

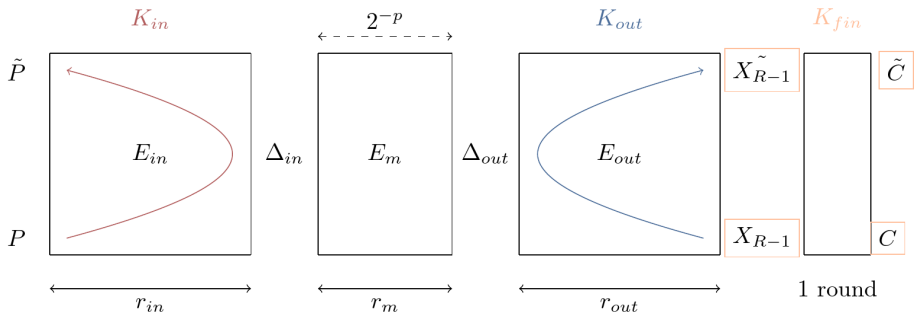


Figure 5.2: A high-level description of the improved Differential MITM technique.

(i) [bottom part] We start with the family of 2^m penultimate states X_{R-1} and build $\tilde{X_{R-1,j}}$ for $j \in K_{out}$. This results in a family of $2^{m+k_{out}}$ elements $\tilde{X_{R-1}}$.

(ii) [top part] We start with the family of 2^m ciphertexts $\mathcal{C}$, we obtain the P by decrypting. Then by guessing K_{in}, we obtain a family of $2^{m+k_{in}}$ possible values for $\tilde{P}_i$ which results in $\tilde{C}_i$ after encryption.

(iii) Match the families $\tilde{C}$ and $\tilde{\mathcal{C}}$.

The number of possible solutions might seem higher by a factor of 2^{2m}, but note that the following equation must be satisfied: $C \oplus \tilde{C} = X_{R-1} \oplus \tilde{X_{R-1}}$ around the final key addition. This adds m bit-conditions, or more if this final subkey was already determined by K_{in} and K_{out}, which is usually the case. This implies m additional conditions, and $2^{2m}2^{-m}2^{-m} = 1$, so the cost, given by the number of solutions, stays exactly the same as the attack with one round less. We will see how this technique can be applied in practice in Section 5.2. In the end the time complexity is:

$$\mathcal{T} = 2^{p-m} \times (2^{k_{in}+m} + 2^{k_{out}+m}) + 2^{k_{in}+k_{out}-|K_{in} \cap K_{out}|-n+p},$$

which is equal to the time complexity of the general framework of differential MITM attacks. The data complexity stays unchanged as well, regarding the memory complexity is increased by a factor 2^m. We can note that m can be smaller than the number of bits involved in the last round key addition k_{final} since some bits of K_{final} might be fixed at the beginning of the attack if $K_{in} \cap K_{out} \cap K_{final} \neq \emptyset$.

5.1.3 Improvement: Reducing data with imposed conditions

We explain here a way to obtain time-data-memory trade-offs for the original attack. If when choosing the plaintext P, we impose x of its bits, that might have been active otherwise, to a certain value, and if we expect the same from the associated plaintext $\widetilde{P}$, the overall probability of the attack will decrease to 2^{-p-x}, as we will have to repeat the procedure until a $\widetilde{P}$ that satisfies this constraint is found. More precisely, if $\widetilde{P}$ does not fit this condition, the corresponding tuple will not be stored in the hash table since

we do not have access to its ciphertext. However by doing so, the data complexity will be reduced by a factor of 2^x as well as the memory complexity. When combining this technique with the previous one, we can derive the following two inequalities for x:

$$p + x \leq n - x \quad \text{and} \quad 2^{p+x}(2^{k_{in}} + 2^{k_{out}}) < 2^k.$$

This type of trade-off applies in particular when all the code book of size 2^n is needed before fixing the x bits, and the data complexity becomes 2^{n-x}. Since we need to iterate the attack 2^{p+x} times instead of 2^p times, the time complexity is increased by a factor 2^x.

Data reduction without time increase. As the total number of candidates for the key of the input part (respectively output) will be $2^{k_{in}-x}$ (respectively $2^{k_{out}-x}$), if we are able to find these candidates with their associated $\widetilde{P}$ (respectively $\widetilde{C}$) in a complexity given by the number of solutions, the time complexity would become:

$$2^{p+x}(2^{k_{in}-x} + 2^{k_{out}-x}) = 2^p(2^{k_{in}} + 2^{k_{out}}),$$

which allows us to reduce the data complexity to 2^{n-x} while not increasing the time complexity. The optimal data complexity in this case will be $2^{\frac{n+p}{2}}$, obtained with x equal to $\frac{n-p}{2}$.

This can actually be done in many cases using rebound-like techniques [Men+09]. An example can be seen in Section 5.2.

5.1.4 Discussion and Comparison

As argued before, our new cryptanalysis technique is closely related to two families of techniques: MITM attacks and differential attacks. In this section we will discuss similarities and differences between these families and will try to identify cases where our new technique might be efficient or cases where it permits to reach better results compared to the best known attacks.

5.1.4.1 Relation to MITM attacks

In relation to MITM attacks [DH77a] and its variants, our attack has the potential of reaching more rounds because, similarly to the sieve-in-the-middle technique [CNV13b], some nonnegligible part of the cipher is considered in the middle. The starting point of our research was whether it was possible to add more rounds in the middle of a MITM-like attack and this is how we came up with the new attack, the idea was to increase the size of the middle part by considering a differential. The data complexity of the new attack could be higher than a classical MITM one as now we compute a new $\widetilde{P}$ from each guess of the key, and this plaintext can take many different values, besides the 2^p different plaintexts P_ℓ taken as starting points in order to find one that satisfies the differential. On the other hand, despite the fact that the sets of bits K_{in} or K_{out} involved in the parallel computations of the differential MITM attack are not

determined in the same way as the key bits involved in MITM attacks, we expect those quantities to be relatively close under similar settings, as this principally depends on the propagation properties of the round function. More precisely, it seems that the more aligned [Bor+21] the round function is, the closer the sets will be.

Therefore, we expect that ciphers where classical MITM attacks work well, can also be interesting targets for differential MITM attacks. This is actually how we found the application shown in Section 5.2 on SKINNY. Indeed, the best known attack against SKINNY-384-128 previous to ours was a MITM one [Don+21].

5.1.4.2 Relation to differential attacks

Curiously enough, our new attack can also be seen as a new way of performing the key-recovery part associated to a differential distinguisher.

Classical differential attacks. In Section 2.1.2, we gave a description of differential attacks. The time complexity of this type of attack consists in the sum of the three steps::

$$\mathcal{T} = 2^{p+1} + 2^{p+d_{in}+d_{out}-n} + 2^{p+d_{in}+d_{out}-n} C_{KR}$$

By comparing the complexities, the differential MITM approach tends to be more effective if d_{in} and d_{out} are large , which can happen when several rounds are appended, the complexity of a differential MITM attack for an equivalent number of rounds might be more interesting. Indeed, the influence of the input and output extensions to the complexity are added and not multiplied. In particular, our attack can become much more efficient when the key size of the cipher is bigger than the state size, otherwise 2^p might already be close to the limit.

5.2 Differential Meet-the-Middle attacks against SKINNY-128-384

We provide in this section our applications to SKINNY-128-384. We start by recalling the specifications of the SKINNY family of ciphers.

5.2.1 Specifications of SKINNY

The SKINNY family of tweakable block ciphers was designed by Beierle et al. [Bei+16]. It is a family of lightweight ciphers following a classical SPN structure but implementing a very compact S-box, a linear layer based on a sparse non-MDS binary matrix and a lightweight key schedule. The block size n can be 64 or 128 bits and for both versions the state is seen as a 4×4 matrix of 4-bit or 8-bit cells. Row 0 is considered the uppermost one and column 0 is taken to be the leftmost one. The numbering of the words inside the state matrix is as follows.

Table 5.1: Number of rounds for each of the main variants `SKINNY-n-t`.

Block size n	$t = n$	$t = 2n$	$t = 3n$
64	32	36	40
128	40	48	56

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

`SKINNY` follows the tweakey framework [JNP14] and XORs a tweakey to the two upmost rows of the state. There exist three main variants for the tweakey size: $t = n$, $t = 2n$ and $t = 3n$ and the corresponding variant is denoted by `SKINNY-n-t`. Furthermore, the tweakey to block size ratio is denoted by $z = t/n$. The tweakey state is viewed also as a set of z 4×4 arrays of cells. For $z = 3$, which is the variant of interest to us, the three tweakey arrays are denoted by **TK1**, **TK2** and **TK3**.

The round function of `SKINNY` is depicted on Figure 5.3, and the number of times this function is iterated depends on both n and t, as shown in Table 5.1.

One round of `SKINNY` is composed of five operations applied in the following order: `SubCells (SC)`, `AddConstants (AC)`, `AddRoundTweakey (ART)`, `ShiftRows (SR)` and `MixColumns (MC)`. We now briefly describe the operations that are of interest to us.

`SubCells (SC)` This operation applies an S-box to all cells of the state. The table representation of the 8-bit S-box used in the 128-bit variants is given in [Bei+16].

`AddRoundTweakey (ART)` We describe this step only for the variants with $z = 3$. Here, the first and second rows of the 3 tweakey arrays **TK1**, **TK2** and **TK3** are extracted and XORed to the internal state, respecting the bit positions inside the arrays. More formally, if $\texttt{IS}_{i,j}$ is the cell at the intersection of row i and j of the state, we have that for $(i, j) \in \{0, 1\} \times \{0, 1, 2, 3\}$: $\texttt{IS}_{i,j} = \mathbf{TK1}_{i,j} \oplus \mathbf{TK2}_{i,j} \oplus \mathbf{TK3}_{i,j}$.

`ShiftRows (SR)` This operation rotates the cells inside a row to the right by a certain offset that depends on the row. More precisely, cells in row i, where $0 \leq i < 4$, are rotated by i positions to the right.

`MixColumns (MC)` This operation updates the state by multiplying each column by a binary matrix $\mathbf{M}$. This matrix, as well as its inverse are as follows.

$$\mathbf{M} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}, \quad \mathbf{M^{-1}} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}.$$

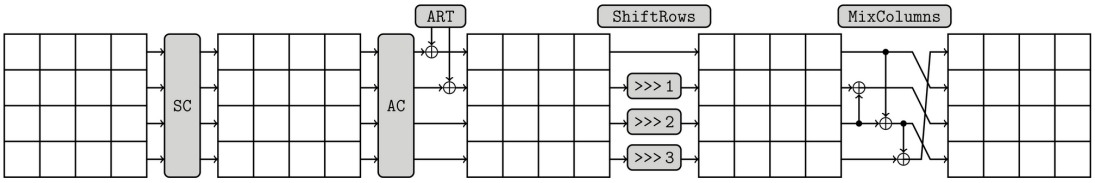


Figure 5.3: Round function of SKINNY [Bei+16]

Next we describe the tweakey schedule of SKINNY-128-384, the variant we analyse in this work.

Tweakey schedule of SKINNY-128-384 At each round, all tweakey arrays are updated as follows (see also Figure 5.4). First, the same permutation P_T is applied on the cell positions of the 3 tweakey arrays: for all $0 \leq i \leq 15$, we have that $\mathbf{TK1}_i \leftarrow \mathbf{TK1}_{PT[i]}$ with

$$P_T = [9, 15, 8, 13, 10, 14, 12, 11, 0, 1, 2, 3, 4, 5, 6, 7],$$

and the same exact permutation is applied to the cells of **TK2** and **TK3**. Finally, the cells of the first two rows of **TK2** and **TK3** are individually updated by the LFSR :

$$(\mathtt{x}_7||\mathtt{x}_6||\mathtt{x}_5||\mathtt{x}_4||\mathtt{x}_3||\mathtt{x}_2||\mathtt{x}_1||\mathtt{x}_0) \rightarrow (\mathtt{x}_0 \oplus \mathtt{x}_6||\mathtt{x}_7||\mathtt{x}_6||\mathtt{x}_5||\mathtt{x}_4||\mathtt{x}_3||\mathtt{x}_2||\mathtt{x}_1),$$

where x_0 is the LSB in a byte.

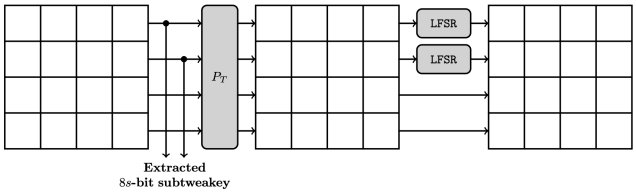


Figure 5.4: Tweakey schedule of SKINNY. All tweakey arrays **TK1**, **TK2** and **TK3** follow the same transformation with the only exception that no LFSR is applied to **TK1**. [Bei+16]

Property. In most attacks, including ours, it is more efficient to guess round-key bits than key bits. Since the key-schedule of SKINNY is fully linear, guessing enough (e.g. 384 in the **TK3** model) independent round-key bits allows to uniquely determine the master key and thus all remaining round-key bits. SKINNY has a unique property which makes easy the evaluation of the dimension of any set of round-key bytes: in the **TKx** model, a round-key byte $\mathtt{K_r[i]}$ always depends on exactly x master key bytes, one from each **TK** keys and they all have the same index. Furthermore, given x round-key bytes from the 30 first rounds and depending on the same x master key bytes, they are

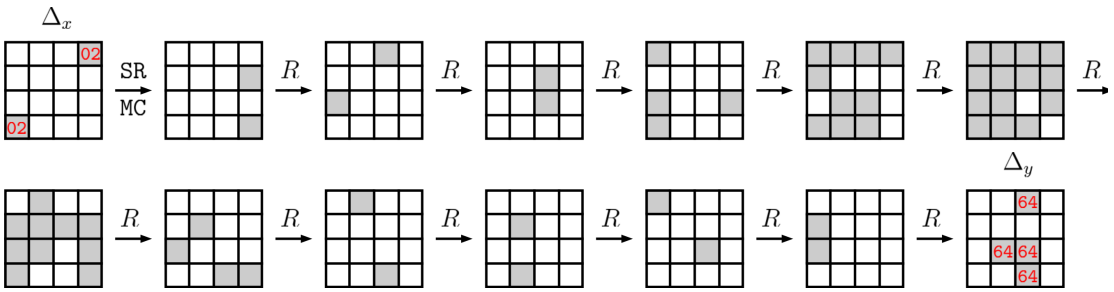


Figure 5.5: Truncated differential trail for the attack on 23 rounds.

always independent and allow to uniquely determine their x corresponding master key bytes.

We present now 3 attacks against round-reduced variants of SKINNY-128-384 by using our new differential meet-in-the-middle technique. We first describe a simple attack against 23 rounds as well as several improvements and trade-offs. We then explain how this attack can be extended to an attack on 24 rounds without increasing the attack's overall complexity. Finally, we describe a new attack against 25 rounds.

5.2.2 An attack against 23-round SKINNY-128-384

As explained in Section 5.1, differential meet-in-the-middle attacks rely on two classical cryptanalysis techniques: differential attacks and meet-in-the-middle attacks. The main idea is to use a meet-in-the-middle attack to generate a pair following a given differential in the middle rounds. Thanks to this procedure, we are able to extend a differential distinguisher by more rounds than with a classical early-abort procedure, as discussed in Section 5.1.4.

Differential. The truncated differential used in our attack against 23 rounds of SKINNY-128-384 is depicted in Figure 5.5. It has 56 active S-boxes, without counting those of the first and the last round. We verified that this truncated differential can be successfully instantiated by using the constrained programming Choco-solver [PFL16], and more precisely the model developed by Delaune *et al.* to search for the best differential characteristics for the SKINNY family of block ciphers [Del+21]. The best instantiation of this truncated differential has a probability of 2^{-119} and there are in total 2048 instantiations with this same probability. These instantiations can be divided into four groups $(\Delta_x^{(i)}, \Delta_y^{(i)})$, for $i = 1, 2, 3, 4$, each one having 512 trails inside, starting with the same difference $\Delta_x^{(i)}$ and terminating after 13 rounds with the same difference $\Delta_y^{(i)}$. We found as well many more differential trails with the same input/output differences but smaller probabilities: 2560 with probability 2^{-120}, 7168 with probability 2^{-121}, 18432 with probability 2^{-122} and 44800 with probability 2^{-123}. Thus the probability of the differential depicted on Figure 5.5 is higher than $2^{-105.9}$.

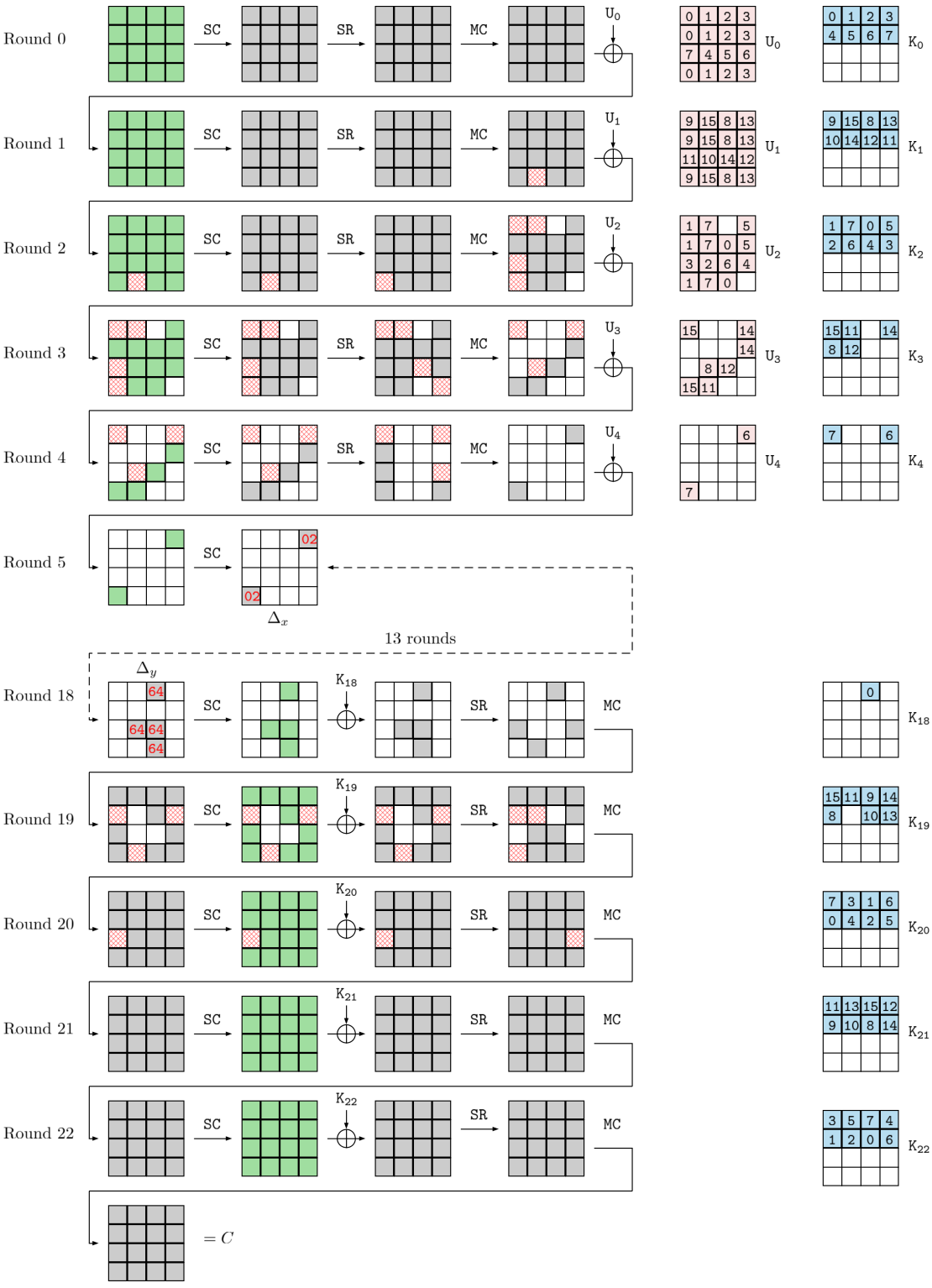


Figure 5.6: Core attack against 23 rounds of $SKINNY-128/384$. Knowledge of blue key bytes allows to compute values of green ones and thus to propagate differences. No difference in both white and red bytes, but the red ones are required to compute green bytes. Indexes in subkey bytes are the indexes of the corresponding master key bytes. The equivalent subkeys U_i are computed as $MC(SR(K_i))$, from the original subkeys K_i.

The attack. We describe now our core attack against 23-round `SKINNY-128-384`.

1. Ask for the encryption of the whole codebook.
2. Randomly pick one plaintext/ciphertext pair (P, C).
3. For each possible value i of K_{in} compute the tuple $(P, \widetilde{P}, i)$ so that the difference on the state after the 6th S-box layer is [0 0 0 2 0 0 0 0 0 0 0 0 2 0 0 0] (see Figure 5.5). Doing so requires to know the values of all the active S-boxes involved in the probability 1 transition $\Delta_x \rightarrow \Delta_P$, where Δ_P is the plaintext difference, and k_{in} is the set of subkey bytes needed to compute them from the plaintext.
4. Store all these tuples in a hash table. This step requires to guess 31 subkey bytes as depicted in Figure 5.6.
5. Similarly, for each possible value j of k_{out} compute the tuple $(C, \widetilde{C}, j)$ so that the difference on the state before the 19th S-box layer is `0x64` on all active bytes. The set k_{out} involves 32 bytes and thus there are 2^{256} such tuples.
6. For each of them check for possible matches on the hash table. The match is performed on both the new ciphertext (i.e. $(\widetilde{P}, \widetilde{C})$ must be a valid plaintext/ciphertext pair) as well as on the linear relations between the subkey bytes of the upper and lower guess.
7. Each match leads to a (full) key candidate that can be tried against very few additional plaintexts (3 in our case).
8. Repeat from Step 2 until the right key is retrieved.

The data complexity of this attack is 2^{128} since in Step 1 we ask for the encryption of the full codebook. The memory complexity is determined by Step 3 in which 2^{248} words of $128 + 248 = 376$ bits each are stored. Note that we do not need to store C_1 since it is common to all tuples. Thus the memory complexity is $2^{249.5}$ 128-bit words. The time complexity is 2^{248} for computing the hash table, 2^{256} for performing Step 4, and, as shown in Table 5.2, $k_{in} \cup k_{out}$ is the full key so that the complexity of Step 6 is $2^{384-128} = 2^{256}$. Finally, the attack has to be repeated $2^{105.9}$ times (the probability of the distinguisher) in order to construct one right differential pair. Hence, the overall complexity of our attack is $2^{105.9} \times 2^{256} = 2^{361.9}$.

Decreasing memory complexity. It is possible to decrease the memory complexity of the attack by avoiding the match on the linear relations between both k_{in} and k_{out}. Indeed, since the key-schedule of `SKINNY` is fully linear, we can first guess the intersection of k_{in} and k_{out}, and only then run the attack. The dimension of the intersection is $248 + 256 - 384 = 120$ and thus the memory complexity can be decreased to $2^{249.5-120} = 2^{129.5}$.

Table 5.2: Subkey bytes involved in the 23-round core attack.

Byte	k_{in}	k_{out}	# equations
0	$K_0[0]$, $K_2[2]$	$K_{18}[2]$, $K_{20}[4]$, $K_{22}[6]$	2
1	$K_0[1]$, $K_2[0]$	$K_{20}[2]$, $K_{22}[4]$	1
2	$K_0[2]$, $K_2[4]$	$K_{20}[6]$, $K_{22}[5]$	1
3	$K_0[3]$, $K_2[7]$	$K_{20}[1]$, $K_{22}[0]$	1
4	$K_0[4]$, $K_2[6]$	$K_{20}[5]$, $K_{22}[3]$	1
5	$K_0[5]$, $K_2[3]$	$K_{20}[7]$, $K_{22}[1]$	1
6	$K_0[6]$, $K_2[5]$, $K_4[3]$	$K_{20}[3]$, $K_{22}[7]$	2
7	$K_0[7]$, $K_2[1]$, $K_4[0]$	$K_{20}[0]$, $K_{22}[2]$	2
8	$K_1[2]$, $K_3[4]$	$K_{19}[4]$, $K_{21}[6]$	1
9	$K_1[0]$	$K_{19}[2]$, $K_{21}[4]$	0
10	$K_1[4]$	$K_{19}[6]$, $K_{21}[5]$	0
11	$K_1[7]$, $K_3[1]$	$K_{19}[1]$, $K_{21}[0]$	1
12	$K_1[6]$, $K_3[5]$	$K_{21}[3]$	0
13	$K_1[3]$	$K_{19}[7]$, $K_{21}[1]$	0
14	$K_1[5]$, $K_3[3]$	$K_{19}[3]$, $K_{21}[7]$	1
15	$K_1[1]$, $K_3[0]$	$K_{19}[0]$, $K_{21}[2]$	1

Data-time-memory trade-off. To decrease the data complexity of our attack, as described in Section 5.1.3, it is possible to only ask for the encryption of a portion of the whole codebook, let say 2^{128-x} plaintext/ciphertext pairs. In this case, the probability that we have access to the corresponding ciphertext of $\widetilde{P}$ is 2^{-x} and the attack has to be ran 2^x times to compensate. Overall, the complexity of our 23-round attack is then $\mathcal{D} = 2^{128-x}$ plaintext/ciphertext pairs, $\mathcal{M} = 2^{129.5-x}$ 128-bit words and $\mathcal{T} = 2^{361.9+x}$ encryptions. To expect at least one pair following the differential under the extra constraint on the plaintexts, x cannot be higher than $(128 - 105.9)/2 = 11.05$.

As explained in Section 5.1.3, in practice, given any pair of ciphertexts, we can enumerate the possible values for both k_{in} and k_{out} with a complexity roughly equivalent to the number of solutions. Such a procedure is described in Section 5.2.3. Thus, the trade-off does not increase the time complexity and the overall complexity of our attack is $D = 2^{117}$, $T = 2^{361.9}$ and $M = 2^{118.5}$.

5.2.3 Enumeration Procedure of k_{out} for the 23-round Attack

In this section we describe a procedure to retrieve the possible values of k_{out} in the case where the pair of ciphertexts is given. Its complexity is 2^{128} simple operations, showing that the data/time/memory trade-off described in Section 5.1.3 can be applied without increasing the time complexity. Since the key is not applied on the full state, the procedure is more complex than a classical rebound but still relies on the fact that, on average, knowing the differences at both the input and output of an Sbox leads to one pair of actual values.

The steps of our enumeration procedure are depicted on Figure 5.2.3. The main idea is to propagate differences from the ciphertexts to Round 18, get the actual values since differences in this internal state are fully known, and propagate them back to the ciphertexts to obtain the key material we want. Except from Step 1 in which we guess 8 values, all following steps perform one guess each. Steps 7 and 8 have both a probability of success of 2^{-8} and thus we expect only $2^{13\times 8} = 2^{104}$ partial solutions at the end of Step 8.

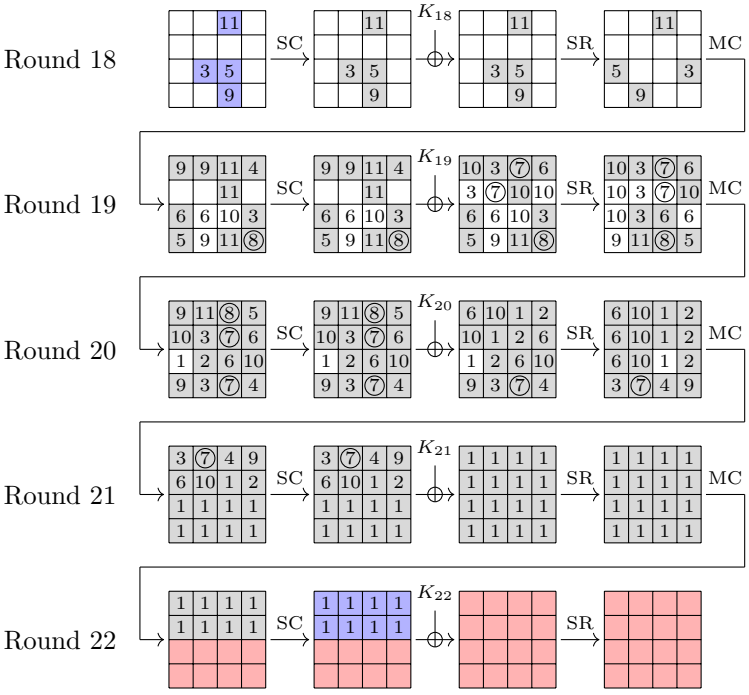


Figure 5.7: Enumeration procedure of k_{out} for the 23-round attack. Differences in blue cells and actual values in red cells are known. No difference in white cells.

5.2.4 Extension to 24 rounds

Our differential meet-in-the-middle attack against 23-round `SKINNY` can be extended to an attack against 24 rounds without increasing its overall complexity by using the generic improvement presented in Section 5.1.2. This can be achieved since on one hand the key-schedule is linear (and enough subkey bytes are involved in our attack so that the master key is fully retrieved) and on the other hand the subkey is only applied on half of the state in each round. The scenario of this new attack is quite similar to the original one:

1. Ask for the encryption of the whole codebook.

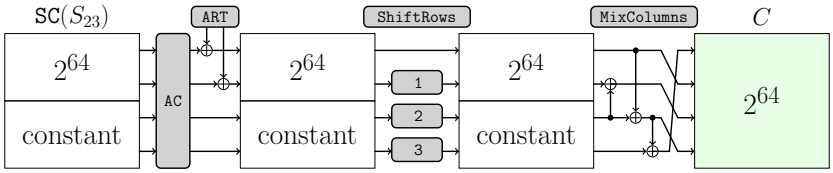


Figure 5.8: Last round of the attack against 24 rounds.

2. Pick 2^{64} plaintext/ciphertext pairs (P_ℓ, C_ℓ) such that $MC^{-1}(C_\ell)$ is constant on the last two rows as depicted in Figure 5.8. Here we exploit the fact that the round key is only applied on the first two rows of the internal state.

3. As for this original attack, compute all possible tuples $(P_\ell, \widetilde{P}_\ell^i, i)$ for each value i of k_{in} and each P_ℓ from the structure defined at the previous step such that the state difference after the 6th S-box layer is `0x02` on bot.h active bytes.

4. Store them in a hash table. Note that the tuples are computed for all the 2^{64} plaintexts selected at Step 2 so the memory complexity is $2^{248+64} = 2^{312}$ 504-bit words.

5. For each value j of k_{out} and each state $S_{23,\ell}$ coherent with Step 2 (i.e. 2^{64} states, one for each possible value of the subkey K_{23}), compute all possible tuples $(S_{23,\ell}, \widetilde{S}^j_{23,\ell}, j)$ so that the difference on the state before the 19th S-box layer is `0x64` on the four active bytes.

6. Check for possible matches on the hash table. The match is now performed on three quantities:

   - the difference between the last states: $C \oplus \widetilde{C} = \texttt{MC} \circ \texttt{SR}(\texttt{SC}(S_{23}) \oplus \texttt{SC}(\widetilde{S}_{23}))$. This is a 64-bit filter because the difference is zero on the two last rows since $MC^{-1}(C)$ is constant on these rows.
   - the filter on the keys (from key schedule equations): a 120-bit filter as for the original attack against 23 rounds (15 equations on 8 bits each, see Table 5.2).
   - the filter on the keys (from equations describing the last round). Indeed, since $k_{in} \cup k_{out}$ generates the master key, K_{23} can be rewritten as $f(k_{in}) \oplus g(k_{out})$ where f and g are both linear and, because of the linearity of all the operations, the equation $C = \texttt{MC}(\texttt{SR}(\texttt{SC}(S_{23}) \oplus K_{23}))$ can thus be rewritten as $C \oplus \texttt{MC}(\texttt{SR}(f(k_{in})) = \texttt{MC}(\texttt{SR}(\texttt{SC}(S_{23}) \oplus g(k_{out})))$. This represents a 64-bit filter.

7. Repeat from Step 2 until the right key is retrieved.

The attack has to be repeated enough times so that the structure contains at least one pair following the differential. Since during Step 2 we generate 2^{64} pairs and since the probability of the differential is $2^{-105.9}$, the procedure has to be repeated $2^{41.9}$ times.

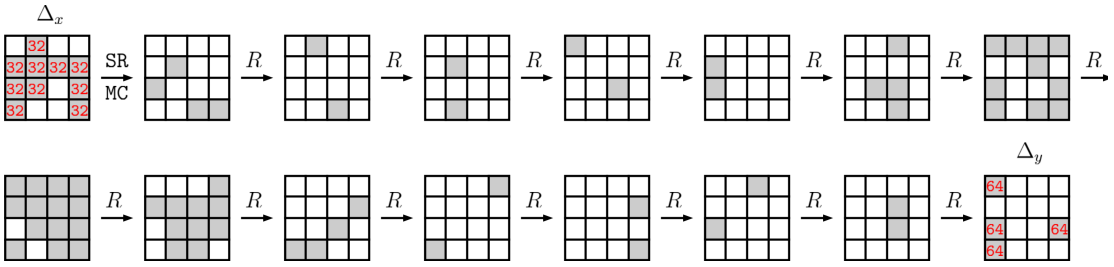


Figure 5.9: Truncated differential trail for the attack on 25 rounds.

Thus the data complexity is 2^{128} (i.e. the whole codebook is needed), the memory complexity is around 2^{314} 128-bit words and the time complexity $2^{256+64+41.9} = 2^{361.9}$ encryptions.

Note that previous improvements regarding both the memory and data complexities still apply and thus our attack has complexity: $\mathcal{D} = 2^{128-11} = 2^{117}$ plaintext/ciphertext pairs, $\mathcal{M} = 2^{194-11} = 2^{183}$ 128-bit words and $\mathcal{T} = 2^{361.9}$ encryptions.

5.2.5 An attack against 25 rounds of SKINNY-128-384

To mount a differential meet-in-the-middle attack against 25 rounds of SKINNY, we used the differential depicted in Figure 5.9. The best instantiation of this truncated differential has a probability of 2^{-131}. By fixing the difference of the active bytes to 0x32 at the input and to 0x64 at the output, we found several instantiations with a high enough probability: 2048 with probability 2^{-131}, 10240 with 2^{-132}, 28672 with 2^{-133} and finally 73728 trails with probability 2^{-134}. Thus, the probability of the depicted differential is higher than $2^{-116.5}$. This differential is then extended by 4 rounds to the plaintext and 5 rounds to the ciphertext to reach 24 rounds as depicted in Figure 5.10.

The key bytes involved in the attack are given in Table 5.3, leading to a complexity of $\mathcal{D} = 2^{128}$ data, $\mathcal{T} = 2^{256} \times 2^{116.5} = 2^{372.5}$ encryptions and $\mathcal{M} = 2^{249.5}$ 128-bit words. Furthermore, the dimension of the intersection between both k_{in} and k_{out} is once again $248 + 256 - 384 = 120$, which allows to reduce the memory complexity to $2^{129.5}$ 128-bit words.

As for the 23-round attack described above, this attack can be extended by one round without increasing its overall complexity. As a consequence, and after applying the data/time/memory trade-off presented in Section 5.1.3, the complexity of our attack against 25 rounds is $\mathcal{D} = 2^{128-x}$ plaintext/ciphertext pairs, $\mathcal{M} = 2^{194-x}$ 128-bit words and $\mathcal{T} = 2^{372.5+x}$ encryptions. In this case, x cannot be higher than $(128-116.5)/2 = 5.75$. Furthermore, we can again apply this trade-off without increasing the time complexity and thus the final complexity is $D = 2^{122.3}$, $T = 2^{372.5}$ and $M = 2^{188.3}$.

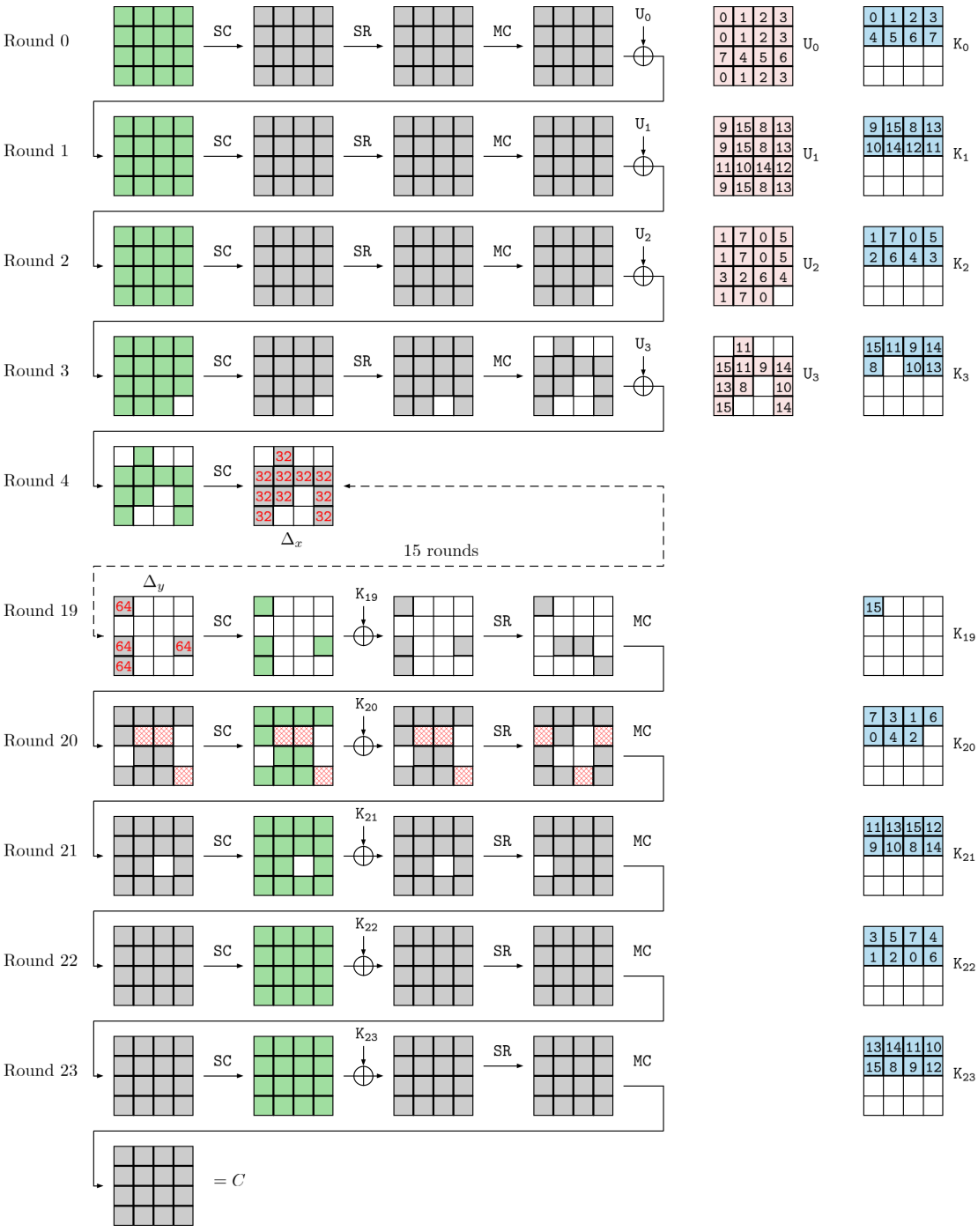


Figure 5.10: Core attack against 24 rounds of SKINNY-128-384. Knowledge of blue key bytes allows to compute values of green ones and thus to propagate differences. No difference in both white and red bytes, but the red ones are required to compute green bytes. Indexes in subkey bytes are the indexes of the corresponding master key bytes.

Table 5.3: Subkey bytes involved in the 24-round core attack.

Byte	k_{in}	k_{out}	# equations
0	$K_0[0]$, $K_2[2]$	$K_{20}[4]$, $K_{22}[6]$	1
1	$K_0[1]$, $K_2[0]$	$K_{20}[2]$, $K_{22}[4]$	1
2	$K_0[2]$, $K_2[4]$	$K_{20}[6]$, $K_{22}[5]$	1
3	$K_0[3]$, $K_2[7]$	$K_{20}[1]$, $K_{22}[0]$	1
4	$K_0[4]$, $K_2[6]$	$K_{20}[5]$, $K_{22}[3]$	1
5	$K_0[5]$, $K_2[3]$	$K_{22}[1]$, $K_{24}[0]$	1
6	$K_0[6]$, $K_2[5]$	$K_{20}[3]$, $K_{22}[7]$	1
7	$K_0[7]$, $K_2[1]$	$K_{20}[0]$, $K_{22}[2]$	1
8	$K_1[2]$, $K_3[4]$	$K_{21}[6]$, $K_{23}[5]$	1
9	$K_1[0]$, $K_3[2]$	$K_{21}[4]$, $K_{23}[6]$	1
10	$K_1[4]$, $K_3[6]$	$K_{21}[5]$, $K_{23}[3]$	1
11	$K_1[7]$, $K_3[1]$	$K_{21}[0]$, $K_{23}[2]$	1
12	$K_1[6]$	$K_{21}[3]$, $K_{23}[7]$	0
13	$K_1[3]$, $K_3[7]$	$K_{21}[1]$, $K_{23}[0]$	1
14	$K_1[5]$, $K_3[3]$	$K_{21}[7]$, $K_{23}[1]$	1
15	$K_1[1]$, $K_3[0]$	$K_{19}[0]$, $K_{21}[2]$, $K_{23}[4]$	2

5.2.6 Comparison of the attacks on SKINNY-128-384 with differential attacks

One may wonder whether the improved attacks on SKINNY-128-384 and in particular the one reaching 25 rounds, are exclusively due to the quality and the length of the differential distinguisher used, thus, that a differential attack could reach the same number of rounds if mounted on this distinguisher. In this section we show that this is not the case, and that the best classical differential attacks reached strictly fewer rounds than our attack. We show that even when considering optimal lower bounds for the complexities of classical differential attacks, it is not possible to reach more than 22 rounds using the early abort technique and 24 rounds using a parallel key recovery for SKINNY-128-384 this way. This shows that differential MITM attacks can reach more rounds than differential ones in certain scenarios.

As explained in Section 2.1.2, the time complexity of a differential attack is:

$$\mathcal{T} = 2^{p+1} + 2^{p+d_{in}+d_{out}-n} + 2^{p+d_{in}+d_{out}-n} C_{KR}$$

To mount a differential attack on SKINNY-128-384, there are two ways to perform the key recovery, hence to compute C_{KR}.

Sequential key recovery. This paragraph is not included in the publication [Bou+23b] and describes the tool I implemented to compute sequential key recovery for SKINNY-128-384. This case corresponds to performing an early abort as described in Section 2.1.2.3. To obtain the best complexity possible, I implemented a tool that given

a differential pattern and a total number of rounds will compute the complexity for a differential attack obtained through an early abort technique. For an attack on R rounds, the tool proceeds as depicted in Algorithm 5.2. It consists of repeating the following procedure for all the possible tuples (r_{in}, r_{out}) such that $r_{in} + r_{out} + r_m = R$. First, it propagates Δ_{in} (resp. Δ_{out}) to the input (resp. to the output). Then it enumerates all the conditions to propagate a pair from D_{in} to Δ_{in} and D_{out} to Δ_{out} together with the minimal amount of key required to verify this condition. Finally, it computes the complexity on an early abort using the conditions computed.

Algorithm 5.2 Optimise Differential Attack with Sequential Key recovery

$(r^*_{in}, r^*_{out}) \leftarrow (0, 0)$
$c^* \leftarrow$ Max Int
for (r_{in}, r_{out}) such that $r_{in} + r_{out} + r_m = R$ **do**
    $L_{cplx} \leftarrow$ Empty List ▷ Complexity list initialisation
    $\text{Char}_{top} \leftarrow \text{propag}_{top}(\Delta_{in}, r_{in})$ ▷ Top propagation
    $\text{Char}_{bot} \leftarrow \text{propag}_{bot}(\Delta_{out}, r_{out})$ ▷ Bot propagation
    $L_{cplx} \leftarrow \{\max\{2^{p+1}, 2^{\frac{p+1+d_{in}}{2}}\}\} \cup L_{cplx}$ ▷ First step
    $L_{cplx} \leftarrow \{2^{p+d_{in}+d_{out}-n} \cup L_{cplx}\}$ ▷ Second step
    $L_{top} \leftarrow \text{Cond}_{top}(\text{Char}_{top})$ ▷ List of the conditions on the top part
    $L_{bot} \leftarrow \text{Cond}_{bot}(\text{Char}_{bot})$ ▷ List of the conditions on the bot part
    $L_{cond} \leftarrow L_{top} \cup L_{bot}$
    for $c \in L_{cond}$ **do**
        $k_c \leftarrow \sharp\text{Guess}(c)$ ▷ number of key guesses required to check the condition c
        $f_c \leftarrow \sharp\text{Filter}(c)$ ▷ number of condition bits of c
        $\mathcal{C} \leftarrow L_{cplx}[0]$ ▷ previous step complexity
        $L_{cplx} \leftarrow \{\mathcal{C} \cdot 2^{n_c - f_c}\} \cup L_{cplx}$
    if $\sum_{\mathcal{C} \in L_{cplx}} \mathcal{C} < c^*$ **then**
        $(r^*_{in}, r^*_{out}) \leftarrow (r_{in}, r_{out})$
        $c^* \leftarrow \sum_{\mathcal{C} \in L_{cplx}} \mathcal{C}$
return c^*, r^*_{in}, r^*_{out}

Using the tool, we were able to successfully attack a maximum of 22 rounds of SKINNY-128-384 with a data complexity of $2^{122.75}$ and a time complexity of $2^{367.1}$. For the case of 23 rounds, the best attack has a time complexity of $2^{398.5}$, hence it is more expensive than an exhaustive search, though some new optimization techniques might make it work in the future. Algorithm 5.2 is the first step to a more general tool: Automatic key recovery. An ongoing work will compute the order on the condition that will minimize the complexity and will allow the use of more advanced techniques like parallel matching and parallel key recovery.

Parallel key recovery. In the case of SKINNY-128-384, we will show that performing a parallel key recovery cannot result in an attack exceeding 24 rounds. Indeed, we will consider lower bounds on the complexity: the number of solutions. Starting from the

differential depicted in Figure 5.9, the complexity of a differential attack on 24 rounds of SKINNY-128-384 cannot be lower than:

$$2^{116+128}(2^{120} + 2^{128} + 2^{120+128-120}) = 2^{373},$$

where the parameters are $p = 116$, $d_{in} = 128$, $d_{out} = 128$ (Figure 5.10), and $C_k = (2^{120} + 2^{128} + 2^{120+128-120})$. Indeed, considering both parts in parallel, we have $64 + 64 + 64 + 56$ key bits involved in the input for a condition on 128 bits of recovering the input difference (so 2^{120} key candidates), and $64 + 64 + 64 + 56 + 8$ key bits involved in the output part, with 128-bit conditions (so 2^{128} key candidates). The memory need is 2^{128} for the parallel computation. The linear relations between both partial keys are $64 + 56$, given by the even and odd subkeys respectively. So the previous result provides a lower bound on the complexity that such an attack would obtain. If we try to add one extra round and attack 25 rounds, a minimal additional factor of 2^{64} has to be added to one of the parallel computations, as the number of partial solutions will increase by this amount. This would be too expensive and we can therefore conclude that we cannot build classical differential attacks on 25 rounds of SKINNY-128-384 based on this distinguisher.

This extra round can be added in our case due to the different nature of the attack, and its relation to MITM cryptanalysis. Therefore, it is fair to conclude that in this case, differential meet-in-the-middle attacks are more powerful than classical differential attacks, reaching one extra round while having a similar complexity.

5.3 Conclusion

We introduced a new cryptanalysis technique: *differential meet-in-the-middle* and provided an application to SKINNY-128-384 resulting in a 2-round improvement of the best single key attack. The emergence of this new type of attack raises some interesting questions. One may wonder about the criteria required for a cipher to be more vulnerable to differential MITM attacks rather than classical differential attacks or classical meet-the-middle attacks. Also, the set of keys involved in differential MITM attacks does not always coincide with the set of keys required to mount a classical differential attack, it would be interesting to have a better understanding of the behavior of these key sets. Lastly, we can wonder whether it is possible to combine MITM with some other type of attack like linear attacks or differential-linear.

Chapter 6 Quantum impossible differential attacks

This chapter is inspired by [DNS22], which is a joint work with María Naya-Plasencia and André Schrottenloher. In this paper, we presented a quantum version of the *impossible differential cryptanalysis* and its application to `AES` and `SKINNY`. This work answers an open problem proposed in [BNS19] that wonders about the efficiency of the quantized version of impossible differential attacks.

Contents

This chapter organizes as follows: first, we will study the *multiple collision search* problem in the quantum setting to solve the *Data Generation* and the *Pair Sieving* step of the impossible differential framework. Then we will describe a quantum version of the *early-abort* algorithm. Finally, we will give an application of this technique to `SKINNY` and `AES`.

6.1 Quantum Collision Search and Pair Generation

The first step of impossible differential attacks is to solve a *limited birthday* problem. Following the definition given in Section 6.1, we want to generate data using structures that will later be used to create differential pairs. In the quantum setting, we will consider the two steps *Data Generation* and *Pair Sieving* at the same time. For this

step, we use existing quantum collision search algorithms [Kap+16; Bon+23]. Indeed, both quantum and classical algorithms for the limited birthday problem reduce it first to a *multiple collision search* problem.

Problem 6.1 (Multiple collision search)**.** *Let $f : \{0,1\}^n \to \{0,1\}^m$ be a random function,with $n \leq m \leq 2n$ and k such that $k \leq 2n - m$. Given query access to f, find* 2^k collision pairs *of f, i.e., 2^k pairs of entries (x, y) such that $x \neq y$ and $f(x) = f(y)$.*

The constraints on k, n, m above ensure that enough collisions actually exist. For the best-known classical and quantum algorithms, the query complexity matches the time complexity, although the memory consumption can be quite large depending on the respective values of k, n, m. In the classical setting, the problem is solved in time $\mathcal{O}\left(2^{(m+k)/2}\right)$. In the quantum setting, the best-known complexities are summarized as follows in [Bon+23]:

Theorem 6.1 (From [Bon+23])**.** *Let $f : \{0,1\}^n \to \{0,1\}^m$, $n \leq m \leq 2n$ be a random function. Let $k \leq 2n - m$. There exists an algorithm finding 2^k collisions in quantum time $\widetilde{\mathcal{O}}\left(2^{C(k,m,n)}\right)$ (incl. qRAM gates), and using $\widetilde{\mathcal{O}}\left(2^{C(k,m,n)}\right)$ quantum queries to f, where:*

$$C(k, m, n) = \max\left(\frac{2k}{3} + \frac{m}{3}, k + \min\left(m - n, \frac{m}{4}\right)\right) \quad .$$

The corresponding query lower bound, valid for any acceptable k and m, was given by Liu and Zhandry in [LZ19]: $\Omega(2^{2k/3+m/3})$. It is conjectured to be tight for all $n \leq m \leq 2m$ and $k \leq 2n - m$. Note that Theorem 6.1 includes as special cases the Brassard-Høyer-Tapp collision search algorithm [BHT98] ($k = 0, m = n$) and Ambainis' element distinctness algorithm [Amb07] ($k = 1, m = 2n$). For the latter, a precise analysis was done by Childs and Eisenberg [CE05]. They showed that the polynomial factor in the complexity was essentially a constant. By running $(\left\lfloor \frac{\pi}{2} 2^{m/3} \right\rfloor)^2$ evaluations of f, and additional memory operations, a solution is obtained with probability exponentially close to 1. The algorithm of [Bon+23], similarly to Ambainis', relies on a quantum walk, so we can also expect its polynomial factor to be small.

In this paper, we assume the evaluation of the function to be the most costly step, so we take these complexities (including the one of Theorem 6.1) without $\mathcal{O}$ and with a multiplicative factor $\frac{\pi^2}{4}$ only. We focus hereafter only on the exponent.

Quantum Data Generation. Like the classical ones, the quantum algorithms for Pair Generation consist in finding multiple collisions in *structures*. For any input (resp. output) structure we can define a function $H_x : \{0,1\}^{d_{in}} \to \{0,1\}^{n-d_{out}}$ (resp. $H'_x : \{0,1\}^{d_{out}} \to \{0,1\}^{n-d_{in}}$), which is the restriction of E to the structure, composed with a linear mapping that projects to a space orthogonal to D_{out} (resp. D_{in}): $H_x(y) = L \circ E(y)$ .Thus, any collision of H_x is a pair y, z such that: • $y, z \in T_x \implies y \oplus z \in D_{\text{in}}$, and • $H_x(y) = H_x(z) \implies L\left(E(y) \oplus E(z)\right) = 0 \implies E(y) \oplus E(z) \in D_{\text{out}}$ by definition of L and H_x. The same goes for H'_x by replacing E by its inverse.

Following [Bon+23], we have three choices for input structures, depending on the number of required pairs N:

- If $N < \frac{2^{2d_{in}}}{2^{n-d_{out}}} \iff \log_2 N < 2d_{in} - n + d_{out}$, then we need only one structure. To recover all the pairs, we need a time exponent :

$$\max\left(\frac{2\log_2 N}{3} + \frac{n-d_{out}}{3}, \log_2 N + \min\left(n - d_{out} - d_{in}, \frac{n-d_{out}}{4}\right)\right)$$

- If $\frac{2^{2d_{in}}}{2^{n-d_{out}}} < 1$, then we fall back on the approach of [Kap+16], which is to repeat N times a Grover search among structures, to find one that contains a pair. Checking if a structure of size $2^{d_{in}}$ contains a pair is an element distinctness problem, which is solved with Ambainis' algorithm in time $\mathcal{O}\left(2^{2d_{in}/3}\right)$. The time exponent for this case is $\log_2 N + \frac{n-d_{out}}{2} - \frac{d_{in}}{3}$.

- If $1 < \frac{2^{2d_{in}}}{2^{n-d_{out}}} < N$, we need to consider several structures and extract all of their collision pairs. This gives a time exponent:

$$\log_2 N + \max\left(\frac{2}{3}(n - d_{in} - d_{out}), \min\left(n - d_{out} - d_{in}, \frac{n-d_{out}}{4}\right)\right) \quad .$$

By considering E^{-1} instead of E, we can also swap the roles of d_{in} and d_{out}.

Conjecture. If we conjecture that the complexity $\mathcal{O}(2^{\frac{2k}{3}+\frac{m}{3}})$ for the multiple collision search problem can actually be reached for the whole parameter range of Theorem 6.1 (and therefore, that the query lower bound is tight everywhere), all the six cases above can be merged into a single formula reminiscent from the classical one.

Conjecture 6.1. *Let $E : \{0,1\}^n \to \{0,1\}^n$ be a block cipher. Given quantum oracles for E and its inverse (O_E and $O_{E^{-1}}$), the limited birthday problem can be solved in quantum time*

$$Q_N = \max\left(\min_{\Delta\in\{d_{in},d_{out}\}} N^{\frac{2}{3}} 2^{\frac{n-\Delta}{3}}, \right.$$
$$\left. \min\left(N2^{\frac{2}{3}(n-d_{out}-d_{in})}, \min_{\Delta\in\{d_{in},d_{out}\}} N2^{\frac{n}{2}-\frac{\Delta}{6}-\frac{d_{in}+d_{out}}{3}}\right)\right) \quad .$$

6.2 Quantum Key Recovery

In this section, we describe a quantum version of the early-abort algorithm in the case of impossible differential cryptanalysis (Algorithm 2.4) and we study its time complexity.

6.2.1 Assumptions on the Attack

Recall that we have divided $K_{\text{in}\cup\text{out}} = K_1 \times K_2 \times \cdots \times K_\ell$, and that there exists separate *test functions* $T_1(p, k_1)$, $T_2(p, k_1, k_2), \ldots, T_\ell(p, k_1, \ldots, k_\ell)$ that decide whether a pair p invalidates the key guess $k_1, \ldots, k_\ell$. Our goal is to solve the following problem.

Problem 6.2. *Given the table $\mathcal{T}_0$ that contains the precomputed pairs, find the keys $(k_1, \ldots, k_\ell)$ such that $\mathcal{T}_\ell(k_1, \ldots, k_\ell) = \emptyset$*

In this Section 2.1.2, we introduced the early abort algorithm (Algorithm 2.4), the following algorithm is a version of early abort for impossible differential.

Algorithm 6.1 Early Abort algorithm for impossible differential attacks.

Input: a table $\mathcal{T}_0$ for the pairs
Output: finds the only key k for which there is no invalidating pair

for all $k_1 \in K_1$ **do**
  Build $\mathcal{T}_1(k_1) = \{p \in \mathcal{T}_0, T_1(p, k_1) = 1\}$
  for all $k_2 \in K_2$ **do**
    Build $\mathcal{T}_1(k_1) = \{p \in \mathcal{T}_1, T_2(p, k_1, k_2) = 1\}$
...
    for all $k_\ell \in K_\ell$ **do**
      Sieve the table $\mathcal{T}_{\ell-1}(k_1, \ldots, k_{\ell-1})$ according to T_ℓ, obtain $\mathcal{T}_\ell$
      If $\mathcal{T}_\ell = \emptyset$, then return $(k_1, \ldots, k_\ell)$
...

Algorithm 6.1 enumerates all the key guesses for which there exists no invalidating pair, using ℓ nested loops. The quantum version of this algorithm relies on nested amplitude amplification subroutines, using amplitude amplification. In order to ensure its correctness, we first need to make some *classical* assumptions on the attack. If these assumptions are satisfied, then we will show that our algorithm solves Problem 6.2 *with probability 1.*

The first assumption consists in considering the number of key candidates. Indeed, we want a single key solution, it will happen as long as N is large enough.

Assumption 1. *Given the initial table $\mathcal{T}_0$, there exists a single key $(k_1, \ldots, k_\ell)$ such that $\mathcal{T}_\ell(k_1, \ldots, k_\ell) = \emptyset$.*

Second, we need global bounds on the size of intermediate tables to control the size of the quantum memory required. In the following, we will use the notation $N_i(k_1, \ldots, k_i) := |\mathcal{T}_i(k_1, \ldots, k_i)|$, and $N_0 := N$.

Assumption 2. *No intermediate table exceeds twice its expected size:*

$$\forall k_1, \ldots, k_i, N_i(k_1, \ldots, k_i) \leq 2\Big(\prod_{j=1}^{i} \sigma_j\Big) N \ .$$

Assumption 2 can be reduced to the assumption that, though it can variate between key guesses, there is an upper bound σ_i on the probability to be filtered. This is a heuristic assumption related to the independence of the filtering conditions, which is supposed to hold for the targeted cipher.

Assumption 3. *Let $\mathcal{T}_i(k_1, \ldots, k_i)$ be any intermediate table (incl. the case $\mathcal{T}_0$). Then:*

$$\forall k_{i+1}, \mathbb{P}_{p \in \mathcal{T}_i(k_1,\ldots,k_i)} \left(T_{i+1}(p, k_1, \ldots, k_{i+1})\right) \leq \sigma_i \ .$$

In other words, the probability to pass the condition T_i can variate, but it has some global upper bound σ_i.

Using the Chernoff inequality, it is possible to upper bound the maximal size of intermediate tables over all key guesses, hence recovering Assumption 2.

Lemma 6.1. *Under Assumption 3:*

$$\mathbb{E}\left(N_i(k_1, \ldots, k_i)\right) \leq \left(\prod_{j=1}^{i} \sigma_j\right) N_0 \ , \quad \textit{and:}$$

$$\mathbb{P}\left(\exists i, \exists k_1, \ldots, k_i, N_i(k_1, \ldots, k_i) \geq 2\left(\prod_{j=1}^{i} \sigma_j\right) N_0\right) \leq \sum_{i=1}^{\ell} \left(\prod_{j=1}^{i} |K_j|\right) e^{-2\left(\prod_{j=1}^{i} \sigma_j\right)^2 N_0} \ .$$

Proof. Let us fix i and a choice of $k_1, \ldots, k_i$. For all pairs p in $\mathcal{T}_{i-1}(k_1, \ldots, k_{i-1})$, the events that p falls in $\mathcal{T}_i(k_1, \ldots, k_i)$ are independent and of probability less than σ_i (by Assumption 3). This also allows us to deduce:

$$\forall k_1, \ldots, k_i, \mathbb{P}_{p \in \mathcal{T}_0}\left(p \in \mathcal{T}_i(k_1, \ldots, k_i)\right)$$
$$\leq \mathbb{P}_{p \in \mathcal{T}_0}\left(T_1(p, k_1) = 1\right) \times \ldots \times \mathbb{P}_{p \in \mathcal{T}_{i-1}(k_1,\ldots,k_{i-1})}\left(T_i(p, k_1, \ldots, k_i) = 1\right) \leq \prod_{j=1}^{i} \sigma_j \ .$$

By independence, we deduce

$$\mathbb{E}\left(N_i(k_1, \ldots, k_i)\right) \leq \left(\prod_{j=1}^{i} \sigma_j\right) N_0$$

Next, we apply a Chernoff bound for this given choice of $k_1, \ldots, k_i$:

$$\mathbb{P}\left(N_i(k_1, \ldots, k_i) \geq 2\left(\prod_{j=1}^{i} \sigma_j\right) N_0\right) \leq \mathbb{P}\left(N_i(k_1, \ldots, k_i) - \mathbb{E}\left(N_i(k_1, \ldots, k_i)\right) \geq \left(\prod_{j=1}^{i} \sigma_j\right) N_0\right)$$
$$\leq e^{-2\left(\prod_{j=1}^{i} \sigma_j\right)^2 N_0}$$

By summing this over all choices of key sequences, we obtain the bound of Lemma 6.1. □

Thanks to Lemma 6.1, we can check that Assumption 2 is satisfied with high probability. In practice, we can just stop the filtering process when the intermediate tables become expectedly too small, e.g. of size below 2^{16}.

6.2.2 Filtering of a Table

From now on, we assume that Assumption 1 and Assumption 2 are satisfied. Let $M_i = 2\left(\prod_{j\leq i}\sigma_j\right)N_0$ be the (maximal) size of intermediate tables $\mathcal{T}_i$ at level i. We manipulate quantum states that contain either:

- Partial key guesses k_i: in that case, we just use a register with as many qubits as there are bits in k_i
- Intermediate tables $\mathcal{T}_i$: in that case, we allocate $M_i = 2\,\mathbb{E}\,(N_i)$ qubit registers, where each register holds either a pair or a dummy element. By Assumption 2, this is enough space to represent all intermediate tables; thus, although this data can be in superposition (as it depends on the key guess), there is no case in which the table would exceed the space that we allocated to write it.

We introduce additional notations to make precise time and memory complexity estimates. Since we want to count the time complexity relatively to a cipher evaluation, we introduce: • t_i the time to evaluate the condition T_i; • t the time to perform an operation such as a copy or swap on a register that contains a pair. We count the memory complexity in the number of pairs (a pair can be stored on a $4n$-bit register).

Lemma 6.2. *For all i, there exists a quantum circuit F_i that maps:*

$$|\mathcal{T}_i(k_1,\ldots,k_i)\rangle\,|k_{i+1}\rangle\,|0\rangle \mapsto |\mathcal{T}_i(k_1,\ldots,k_i)\rangle\,|k_{i+1}\rangle\,|\mathcal{T}_{i+1}(k_1,\ldots,k_i,k_{i+1})\rangle\,|*\rangle \quad ,$$

where $*$ are computation qubits that depend only on $k_1,\ldots,k_i,k_{i+1}$. The time complexity, relative to a cipher evaluation, is bounded by:

$$\begin{cases} M_i(t_{i+1}+t) \text{ in the QRAQM model} \\ M_i(t_{i+1}+\frac{(\log_2 M_i)^2}{4}t) \text{ otherwise} \end{cases}$$

and the memory complexity by M_i or $M_i\frac{(\log_2 M_i)^2}{4}$ (mainly due to the $*$ state).

Proof. We start by computing the condition T_i for all pairs in the table $\mathcal{T}_i(k_1,\ldots,k_i)$. We store the result (0 or 1) in additional qubits. We initialize the output table $\mathcal{T}_{i+1}$ with M_{i+1} "dummy" pairs. Then, we take all the good pairs in $\mathcal{T}_i$ and move them to $\mathcal{T}_{i+1}$. There are two ways to do so.

- If we can use qRAM gates, this operation is efficient. We create a quantum register that contains a *counter* c, counting the number of good pairs which have been seen so far. The value of the counter at each time is a deterministic function of

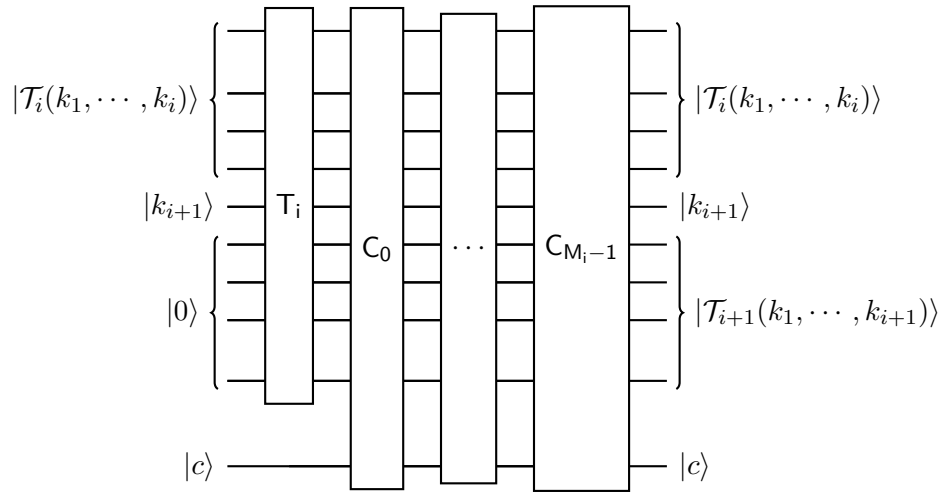


Figure 6.1: The quantum circuit of Lemma 6.2

$\mathcal{T}_i$. However, $\mathcal{T}_i$ is in superposition, and the counter is a superposition as well (entangled with the state of the table).

To move all the good pairs, we apply a sequence of M_i subcircuits $C_j (0 \leq j \leq M_i - 1)$. Each C_j does three operations: 1) read the pair at index j, check if it is good; 2) controlled on the pair being good, increment the counter c; 3) controlled on the pair being good, copy the pair at index c in $\mathcal{T}_{i+1}$.

Thus, the sequence of circuits requires $|\mathcal{T}_i| = M_i$ increment and copy operations. The use of qRAM gates comes from the copies. Indeed, the position at which we need to write in the next table depends on the current counter c. However, the value of c depends on the position of the good pairs among the table $\mathcal{T}_i$, so it depends on $\mathcal{T}_i$ itself. Since $\mathcal{T}_i$, throughout the algorithm, is in superposition, this functionality requires qRAM gates.

- The alternative to qRAM gates is to apply a reversible sorting network, which will move all the elements of $\mathcal{T}_i$ flagged "1" to, say, the M_i first indices in the memory. We then copy these M_i cells (conditioned on whether the copied pair is actually good).

In the second case, the amount of computations increases by a factor $\frac{(\log_2 M_i)^2}{4}$ due to the sorting network. However, the sorting network contains only comparators and swaps and does not require recomputing the filtering conditions. □

It should be noted that the $|*\rangle$ is here to ensure the reversibility of these operations. We could perform uncomputations to erase it immediately, but we prefer to wait until the table $\mathcal{T}_{i+1}(k_1, \ldots, k_i, k_{i+1})$ is not needed anymore. Then we will erase not only the table but also all of the computations that led to it.

Remark 6.1 (Reversible sorting networks). In order to sort a table of 2^ℓ values, we use the *odd-even mergesort* of Batcher [Bat68]. It uses $S(2^\ell) = 2^{\ell-1}\frac{\ell(\ell-1)}{2} + 2^\ell - 1$ comparators and swaps of registers, that we can approximate to $2^{\ell-2}(\ell^2)$ n-qubit register operations of time t each. In order to make it reversible, we need $S(2^\ell)$ qubits that contain the results of the comparators.

6.2.3 Exact Early Abort

We can build our early abort algorithm by interleaving calls to the filtering circuit of Lemma 6.2 with instances of exact amplitude amplification. For a current set of partial key guesses $(k_1, \ldots, k_i)$, with a corresponding intermediate table, we use amplitude amplification to check whether this set of key guesses leads to the single solution or not. We use $k_1^g, \ldots, k_\ell^g$ to denote this solution.

Lemma 6.3. *Let $1 \leq i \leq \ell$. Let t_i be the time (in quantum operations) to compute the condition T_i. There exists a quantum circuit (unitary) U_i that, on an input state of the form:*

$$|k_1, \ldots, k_i\rangle\, |\mathcal{T}_{i-1}(k_1, \ldots, k_{i-1})\rangle \quad ,$$

writes 1 in an output qubit iff there exists a completion $k_{i+1}, \ldots, k_\ell$ such that $k_1, \ldots, k_\ell$ is the good key and 0 otherwise. Assuming QRAQM, it runs in time:

$$2\sum_{j=i}^{\ell} M_{j-1}(t_j + t)\left(\frac{\pi}{2}\right)^{j-i}\sqrt{\prod_{m=i}^{j}|K_m|} \quad , \text{ using a memory of size } M_{i-1}.$$

Proof. We start by describing U_ℓ, the innermost step. It starts from the table $\mathcal{T}_{\ell-1}(k_1, \ldots, k_{\ell-1})$, and $k_1, \ldots, k_\ell$. It simply needs to compute T_ℓ for all the elements of the table and check if one of them passes the test (if it's the case, then this is not the good key guess). This is done in time $2M_{\ell-1}(t_\ell + t)$ (including an uncomputation).

Next, we assume the existence of U_{i+1} for some $i \geq 1$, and we show how to build U_i from U_{i+1}. Consider an input state of the form: $|k_1, \ldots, k_i\rangle\, |\mathcal{T}_{i-1}(k_1, \ldots, k_{i-1})\rangle$. We first use the circuit F_{i-1} of Lemma 6.2 to create the table $|\mathcal{T}_i(k_1, \ldots, k_i)\rangle$ (as well as the additional $|*\rangle$ state). Next, we try to find k_{i+1} such that $k_1, \ldots, k_i, k_{i+1}$ can be completed into the good key. We have:

- an auxiliary state $|\mathcal{T}\rangle$ containing $|k_1, \ldots, k_i\rangle$, the tables $\mathcal{T}_{i-1}$ and $\mathcal{T}_i$;
- a search space K_{i+1} of fixed size, containing exactly one or zero solution;
- a test function that, using the auxiliary state, can find whether k_{i+1} can be completed into the good key: this is the unitary U_{i+1} given by our recurrence hypothesis.

By using a single instance of amplitude amplification, we obtain a unitary that tests whether the current auxiliary state leads to a solution or not, which uses less than

$\frac{\pi}{2}\sqrt{|K_{i+1}|}+2$ calls to U_{i+1}. We apply this unitary. Finally, we uncompute the circuit F_{i-1} to delete the table $\mathcal{T}_i$ and the associated computation qubits.

We have by recursion, and by Lemma 6.2 (assuming the qRAM case):

$$\begin{cases}\mathsf{time}(U_i) = \left(\frac{\pi}{2}\sqrt{|K_{i+1}|}+2\right)(t+\mathsf{time}(U_{i+1})) + 2M_{i-1}(t_i+t)\\ \mathsf{time}(U_\ell) = 2M_{\ell-1}(t_\ell+t)\end{cases}$$

so we approximate $\mathsf{time}(U_i)$ to:

$$2\sum_{j=i}^{\ell} M_{j-1}(t_j+t)\left(\frac{\pi}{2}\right)^{j-i}\sqrt{\prod_{m=i+1}^{j}|K_m|}\ .$$

The memory complexity is equal to the sum of all $M_{j-1}(t_j+t)$ for the involved filtering circuits. In practice, the size of the first table dominates all the others significantly, and we can approximate it by $M_{i-1}(t_i+t)$. □

Our early abort algorithm is then obtained by using U_1 in a quantum search for k_1, where we start from the table $\mathcal{T}_0$ (this one is also an exact amplitude amplification since we assumed that there was exactly one solution). The memory complexity is dominated by $|\mathcal{T}_0|$.

Corollary 6.1. *Under our assumptions, there exists a quantum early abort algorithm that finds the single good $k_1,\ldots,k_\ell$ in time:*

$$2N\sum_{j=1}^{\ell}(t_j+t)\left(\prod_{m=1}^{j-1}\sigma_m\right)\left(\frac{\pi}{2}\right)^{j}\sqrt{\prod_{m=1}^{j}|K_m|}\ ,$$

using $N(\max t_i+t)$ qubits, in the QRAQM model.

We can also drop the QRAQM requirement by using the QRAQM-free version of Lemma 6.2. This version replaces the multiplicative factor t_i+t by $t_i+\frac{(\log_2 M)^2}{4}t$ when filtering a table of size M. Therefore, we can simply upper bound the total time complexity by multiplying the formula of Corollary 6.1 by the maximum of all these factors.

Corollary 6.2. *Under our assumptions, there exists a quantum early abort algorithm that finds the single good $k_1,\ldots,k_\ell$ in time:*

$$N\left(\max_i t_i+\frac{(\log_2 N)^2}{4}t\right)\sum_{j=1}^{\ell}(t_j+t)\left(\prod_{m=1}^{j-1}\sigma_m\right)\left(\frac{\pi}{2}\right)^{j}\sqrt{\prod_{m=1}^{j}|K_m|}\ ,$$

using $N\left(\max t_i+\frac{(\log_2 N)^2}{4}t\right)$ qubits, but no QRAQM.

Compared with the classical formula, we have put a square root on each $|K_i|$. Indeed, the algorithm that we obtain is no more than Algorithm 2.4 in which each exhaustive search loop on a partial key subspace is replaced by a quantum search.

6.2.4 Improved early abort in Practice

In some cases, and using QRAQM, we can improve the computation of intermediate tables given in Lemma 6.2. Since usually, the treatment of the first table ($\mathcal{T}_0$, of size N) dominates the other steps, we will focus on this one. Note that these improvements do not incur any change in the rest of the algorithm.

Precomputation of the Conditions. Let us assume that the first filtering condition $T_1(p, k_1)$ depends actually only on a small part of the pair p, say p_1. Instead of having to compute T_1 for all p in $\mathcal{T}_0$, we can recompute the entire lookup table of T_1. Then, given k_1, we can obtain the next pairs in two steps: (1) we read the values of the p_1 such that $T(p_1, k_1) = 1$; (2) for each p_1, we fetch the pairs that have this value.

Previously, we needed a time $N(t_1 + t)$ (with QRAQM) to compute $\mathcal{T}_1(k_1)$ for a given k_1. Now, we can reduce this time to $2\sigma_1 Nt$, assuming the precomputation of T_1. Indeed, we can sort the pairs in $\mathcal{T}_0$ by their p_1 value, in order to read them efficiently.

We can also use this precomputation at a later step. For example, if $T_2(p, k_1, k_2)$ depends actually only on p_2, then we can also:

- precompute the lookup table of $T_2(\cdot, k_1, \cdot)$ for a given k_1, before running the nested search on k_2;
- for a given k_2, compute $\mathcal{T}_2(k_1, k_2)$ efficiently in time $2\sigma 2\sigma_1 Nt$ instead of time $2\sigma_1 N(t_2 + t)$

However, this now requires sorting $\mathcal{T}_1(k_1)$ with respect to p_2 when we compute it, and this additional term must be taken into account.

Quantum Filtering. It is possible to speed up the filtering using quantum search. Indeed, when computing $\mathcal{T}_1(k_1)$ from $\mathcal{T}_0$, we expect to find $\sigma_1 N$ filtered pairs among the N initial ones. In the QRAQM model, we can find filtered pairs using a Grover search among all the pairs. We need about $\frac{1}{\sqrt{\sigma_1}}$ search iterates, thus all of them are found in time $\sqrt{\sigma_1}N$ instead of N. However, we must fix a number of quantum search iterates for a computation that is done in superposition over k_1. Thus, there will be some probability of error.

Assumption 2 tells us that the actual number of filtered pairs, at any level, does not exceed twice its expectation. By strengthening this assumption a little, we can suppose that the actual probability of success of a search for good pairs will fall in $[a/2; 3a/2]$ where a is the expected success probability. Then, by Lemma 5 in [BNS19], the search succeeds with a probability larger than $3/4$ (if we measured its result immediately). If the table being constructed is large enough, this is sufficient.

Lemma 6.4. *There exists a quantum circuit QF_i that performs the same operation as in Lemma 6.2, up to an error ε bounded by $\varepsilon \leq e^{\frac{-1}{4}M_{i+1}}$. In the QRAQM model, the time complexity, relative to a cipher evaluation, is bounded by: $2M_i\frac{\pi}{2}\sqrt{\sigma_{i+1}}(t_{i+1} + t)$, and the memory complexity by M_i.*

Proof. Since we are building the table $\mathcal{T}_{i+1}$ from $\mathcal{T}_i$ and a new key guess k_{i+1}, we know that there is a maximum of $2\,\mathbb{E}\,(N_{i+1})$ pairs to look for in all cases. We run in parallel $4\,\mathbb{E}\,(N_{i+1}) = 2M_{i+1}$ quantum searches with $\frac{\pi}{4}\frac{1}{\sqrt{\sigma_{i+1}}}$ search iterates each. The search space is the current table of all pairs. The condition of success is the test function T_i. By our assumption, such an individual search succeeds with a probability of at least $3/4$ for all values of k.

We move all the obtained (possibly) good pairs into the next table. At this point, errors occur if the individual searches fail to obtain all the N_i independent pairs. We perform the following test on the search results: we test if at least $2\,\mathbb{E}\,(N_{i+1})$ contains good pairs. If this is the case, then we can expect to have met all the N_{i+1} pairs independently (up to some random collisions). We can write the result of this test in a qubit, but we do not measure it: we only use it to bound away the cases of failure.

Let us consider a fixed key k. We already know that $N_{i+1} \leq 2\,\mathbb{E}\,(N_{i+1})$. We denote by X the total number of successful searches among $4\,\mathbb{E}\,(N_{i+1})$. Hence we can write:

$$X = \sum_{j=0}^{4\,\mathbb{E}(N_{i+1})} X_j$$

By our assumption, each X_j has expectation at least $3/4$, hence

$$\begin{aligned}\mathbb{E}\,(X) &= \sum_{j=1}^{4\,\mathbb{E}(N_{i+1})} \mathbb{E}\,(X_j) \\ &= 3\,\mathbb{E}\,(N_{i+1})\,.\end{aligned}$$

Therefore, by Chernoff bounds, we obtain:

$$\mathbb{P}_{\mathcal{T}_i}\left(X \leq 2\,\mathbb{E}\,(N_{i+1})\right) \leq \mathbb{P}_{\mathcal{T}_0}\left(X \leq \frac{2}{3}\,\mathbb{E}\,(X)\right) \leq e^{-\frac{2(\mathbb{E}(X)/3)^2}{4\,\mathbb{E}(N_{i+1})}} \leq e^{\frac{-\,\mathbb{E}(N_{i+1})}{2}} = e^{\frac{-1}{4}M_{i+1}}\ .$$

□

By using this erroneous QF_i in a quantum search with t iterates, we will obtain a total error amplitude of about $t\sqrt{\varepsilon}$ (i.e. $t^2\epsilon$ failure probability). This is the result of a standard hybrid argument [Ben+97]. Therefore, if M_{i+1} is sufficiently small (typically 2^{16}), it remains negligible.

6.3 Application to SKINNY

In this section, we describe an impossible differential attack on SKINNY-128-256 and its quantization, Table 6.1 allows the comparison with previous work. The specifications of the SKINNYblock cipher family are given in Section 5.2.

Table 6.1: Summary of best quantum attacks on SKINNY-128-256 (with lower complexities than Grover's search). C = Classical; Q = Quantum; $*$ = QRAQM model. Q1, Q2 and QRAQM are defined in Section 1.4. $**$ means we have extrapolated the complexity on 21 rounds from the original attack on 24 rounds for comparison. Our results clearly provide the best quantum attack and therefore the security margin (the quantum time is counted in block cipher calls).

Algorithm	Rds.	Ref.	Time		Memory		Data	Setting
			C	Q	C	Q		
SKINNY-128-256	21	Grover		$2^{129.65}$		negl.	negl.	Q1
	21	[SMB18]**	$2^{151.2}$		$2^{103.2}$		2^{128}	C
	20	[SMB18]**	$2^{126.5}$		$2^{54.6}$		$2^{126.5}$	C
	21	Section 6.3		$2^{119.2}$	$2^{103.2}$	$(2^{95.2})^*$	$2^{119.2}$	Q2
	21	Section 6.3	2^{128}	$2^{117.5}$	$2^{103.2}$	$2^{106.3}$	2^{128}	Q1

6.3.1 Classical Impossible Attack

In this section, we give an impossible differential attack on SKINNY that achieves 21 rounds based on work from [SMB18]. Since the goal is to recover the full 256-bit tweakey, the generic bound is 2^{256}.

Impossible Pattern. We give the pattern in Figure 6.2.

Pair Generation. Here we give some parameters of the attack described in [SMB18]. With our notation, we get $d_{in} = 32$, $d_{out} = 72$, $c_{\text{in}} = 24$, $c_{\text{out}} = 72, N = 2^{103.17}$, where N ensures that only one subkey will remain in the end.

early abort. In this section we will describe the tweakey recovery.

1. Round 21

   a) From the knowledge of the ciphertext, compute $\Delta X_{21}[8]$ and $\Delta X_{21}[12]$. Based on the properties of MC operation on col(1) of W_{20}, we have $\Delta X_{21}[8] = \Delta X_{21}[12]$. This results in an s-bit filter.

   b) For this step we guess the value of $TK_{21}[0]$. From the knowledge of $Z_{21}[12]$ and $\Delta Z_{21}[12]$, we can compute $\Delta X_{21}[12]$. Based on the properties of MC operation on col(1) of W_{20}, we have $\Delta X_{21}[0] = \Delta X_{21}[12]$. Since $Y_{21}[0] = Z_{21}[0] \oplus TK_{21}[0]$, we perform a single-cell (s-bit) filter on the pairs by verifying the following equations.

   $$S^{-1}(Y_{21}[0]) \oplus S^{-1}(Y_{21}[0] \oplus \Delta Y_{21}[0]) = \Delta X_{21}[0] \ .$$

   c) For this step we guess the value of $TK_{21}[5]$. From the knowledge of $Z_{21}[13]$ and $\Delta Z_{21}[13]$, we can compute $\Delta X_{21}[13]$. Based on the properties of MC

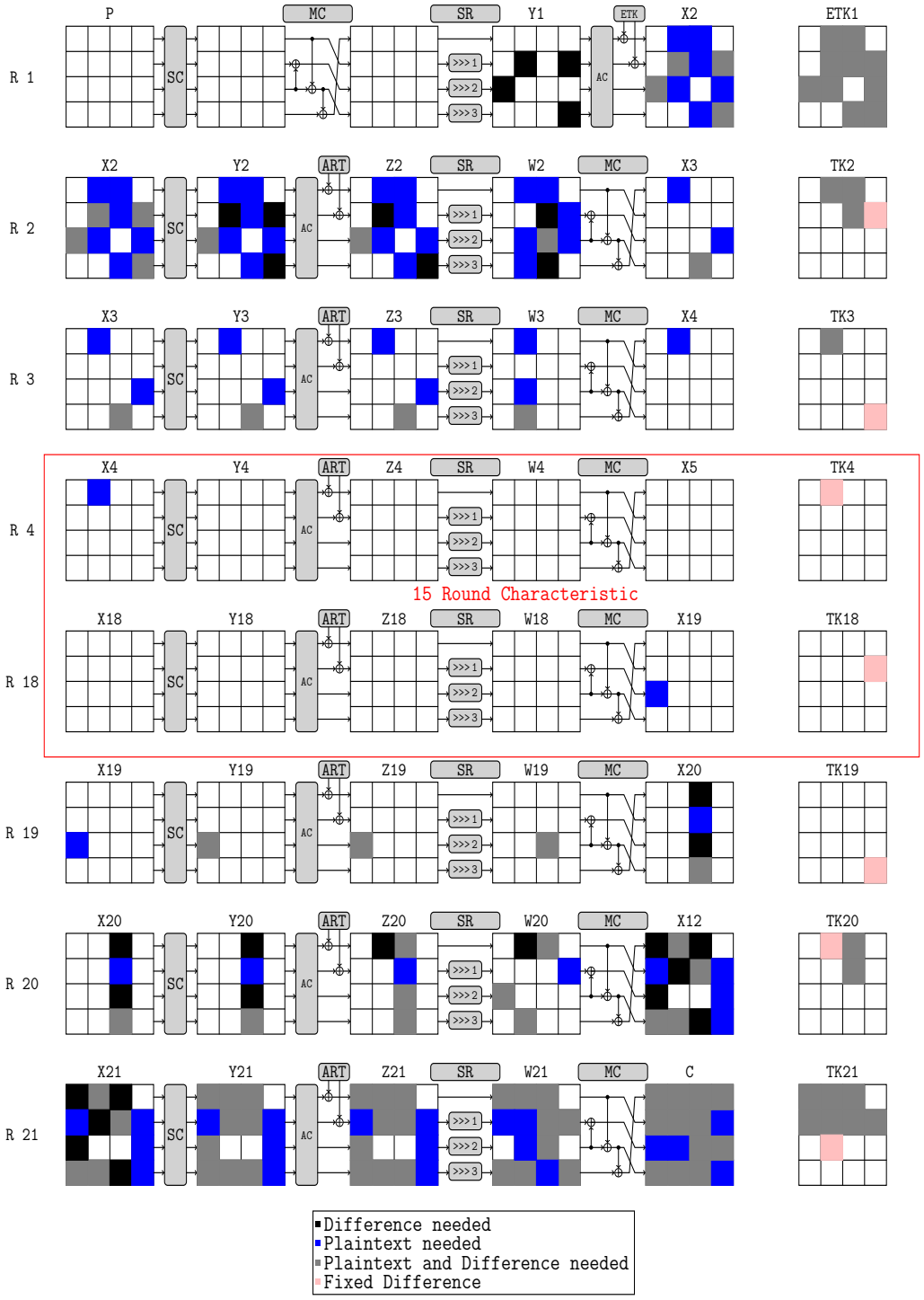


Figure 6.2: Impossible pattern (from [SMB18]).

operation on col(2) of W_{20}, we have $\Delta X_{21}[5] = \Delta X_{21}[13]$. Since $Y_{21}[5] = Z_{21}[5] \oplus TK_{21}[5]$, we perform a single-cell (s-bit) filter on the pairs by verifying the following equations.

$$S^{-1}(Y_{21}[5]) \oplus S^{-1}(Y_{21}[5] \oplus \Delta Y_{21}[5]) = \Delta X_{21}[5] \ .$$

d) For this step we guess the value of $TK_{21}[1]$. Therefore with the knowledge of the ciphertext, we can compute ΔZ_{20} and check $\Delta Z_{20}[1] = \Delta TK_{20}[1]$. This leads to an s-bit filter.

$$S^{-1}(Y_{21}[1]) \oplus S^{-1}(Y_{21}[1] \oplus \Delta Y_{21}[1]) = \Delta X_{21}[1] \ .$$

e) For this step we guess the value of $TK_{21}[2]$. Based on the properties of MC operation on col(3) of W_{20} , we have $\Delta X_{21}[2] = \Delta X_{21}[14]$. Since $Y_{21}[2] = Z_{21}[2] \oplus TK_{21}[2]$ we perform an s-bit filter on the pairs by verifying the following equation.

$$S^{-1}(Y_{21}[2]) \oplus S^{-1}(Y_{21}[2] \oplus \Delta Y_{21}[2]) = \Delta X_{21}[2] \ .$$

f) For this step we guess the value of $TK_{21}[6]$. Based on the properties of MC operation on col(3) of W_{20} , we have $\Delta X_{21}[6] = \Delta X_{21}[14]$. Since $Y_{21}[6] = Z_{21}[6] \oplus TK_{21}[6]$, we perform an s-bit filter on the pairs by verifying the following equation.

$$S^{-1}(Y_{21}[6]) \oplus S^{-1}(Y_{21}[6] \oplus \Delta Y_{21}[6]) = \Delta X_{21}[6] \ ,$$

To continue the key recovery, we will perform two additional key guesses $TK_{21}[4, 7]$ so that the value of $X_{21}[4, 7]$ is known.

2. Round 20. For this step, we guess the value of $TK_{20}[2]$. From the knowledge of $Z_{20}[14]$ and $\Delta Z_{20}[14]$, we can compute $\Delta X_{20}[14]$ and $\Delta X_{20}[10]$. Based on the properties of MC operation on col(3) of W_{19} , we have $\Delta X_{20}[14] = \Delta X_{20}[2] = \Delta X_{20}[10]$. Since $Y_{20}[2] = Z_{20}[2] \oplus TK_{20}[2]$, we perform a two-cell ($2s$-bit) filter on the pairs by verifying the following equations:

$$S^{-1}(Y_{20}[2]) \oplus S^{-1}(Y_{20}[2] \oplus \Delta Y_{20}[2]) = \Delta X_{20}[14].$$
$$\Delta X_{20}[14] = \Delta X_{20}[10]$$

3. Round 19. For this step, we guess the value of $TK_{20}[6]$. Hence we can compute the value of $\Delta X19[8]$ and check if it matches our impossible pattern.

4. Round 1.

a) Guess $ETK[7]$. Hence by using the knowledge of the plaintext, we can compute $\Delta Y_2[7]$ and check $\Delta Y_2[7] = \Delta TK_2[7]$.

Table 6.2: Successive steps of the attack on SKINNY. Each step reduces the table size but increases the key space. The parameter s corresponds to a byte (8 bits).

Index	σ_i	$\lvert K_i\rvert$	Quantum partial complexity: $\prod_{j=1}^{i-1}\sigma_j\sqrt{\prod_{j=1}^{i}\lvert K_j\rvert}$	Classical partial complexity: $\prod_{j=1}^{i-1}\sigma_j\prod_{j=1}^{i}\lvert K_j\rvert$
1.a	2^{-s}	2^0	2^0	2^0
1.b	2^{-s}	2^s	$2^{-s/2}$	2^0
1.c	2^{-s}	2^s	2^{-s}	2^0
1.d	2^{-s}	2^s	$2^{-3s/2}$	2^0
1.e	2^{-s}	2^s	2^{-2s}	2^0
1.f	2^{-s}	2^s	$2^{-5s/2}$	2^0
	2^0	2^{2s}	$2^{-5s/2}$	2^s
2	2^{-2s}	2^s	2^{-2s}	2^{2s}
3	2^{-s}	2^s	$2^{-7s/2}$	2^s
4.a	2^{-s}	2^s	2^{-4s}	2^s
4.b	2^{-s}	2^s	$2^{-9s/2}$	2^s
4.c	2^{-s}	2^s	2^{-5s}	2^s
4.d	2^0	2^{3s}	$2^{-9s/2}$	2^{3s}
5	2^{-s}	2^{3s}	2^{-3s}	2^{6s}

b) For this step we guess the value of $ETK[8]$. From the knowledge of the plaintext, we can compute $\Delta Y_2[8]$ and $\Delta Y_2[15]$. Based on the properties of MC operation on col(3) of W_2 , we have $\Delta Y_2[8] = \Delta Y_2[15]$. We thus perform a single-cell (s-bit) filter on the pairs.

c) For this step we guess the value of $ETK[1]$. From the knowledge of the plaintext, we can compute $\Delta Y_2[5]$ and $\Delta Y_2[8]$. Based on the properties of MC operation on col(3) of W_2 , we have $\Delta Y_2[5] = \Delta Y_2[8]$. We thus perform a single-cell (s-bit) filter on the pairs.

d) To continue the key recovery, we will perform two additional key guesses $ETK[2, 9, 11]$ so that the value of $Y_2[2, 9, 11]$ is known.

5. Round 2 and 3. At this step, we have performed enough guesses on the tweakey material to know the value of $TK_3[1]$. We guess $TK_2[1, 2, 6]$ and compute Y_3 and ΔY_3. Therefore, from the knowledge of $TK_3[1]$, $\Delta Y_4[1]$ can be simply determined. Checking if $\Delta Y_4[1] = TK_4[2]$, will lead to an s-bit filter.

In Table 6.2 we give the cost of each step. In the end, the time complexity for the classical early abort is $2^{151.17}$.

6.3.2 Quantum impossible attack

Since 2 plaintext-ciphertext pairs are required to discriminate the right tweakey, the exhaustive search with Grover's algorithm has a complexity of $2^{129.65}$ encryptions. Our quantum attack will be valid if we manage to outperform this. From Section 6.1 we compute the complexity of the quantum pair generation as $Q_N = 2^{119.17}$ encryptions (in the classical setting $C_N = 2^{128}$ encryptions).

Filtering. Following the idea of Section 6.2, we build the quantum key recovery based on the classical key recovery described in Section 6.3.1. We use the formula of Corollary 6.1 to compute the quantum complexity of this step as:

$$2N \sum_{j=1}^{\ell} (t_j + t) \left(\prod_{m=1}^{j-1} \sigma_m \right) \left(\frac{\pi}{2} \right)^j \sqrt{\prod_{m=1}^{j} |K_m|} \ .$$

This formula features a $(t + t_i)$ term that carries the complexity of $4n$ bit operations and the complexity of performing the test T_i. We assume that $(t + t_i)$ is smaller than the complexity of the encryption, so by our formula with the values in Table 6.2, we get a quantum time of about $2^{104.32}$ encryptions, with QRAQM. Using the formula of Corollary 6.2, we can instead have a quantum time of $2^{117.46}$ encryptions, without QRAQM. The (quantum) memory complexity in the first case corresponds to the table of $2^{95.17}$ pairs remaining after the first condition 1.a has been enforced, which is independent of the key guesses. In the second case, we multiply the memory by a factor $2^{11.14}$ due to the sorting network.

Putting together the complexities of pair generation and key recovery, we give the total complexity for 21 rounds of `SKINNY`-128-256 in Table 6.3. We can propose two quantum attacks: 1) using QRAQM, we use a quantum pair generation and filtering algorithms to reach a complexity of about $2^{119.17}$ encryptions (Q2 setting). 2) Without QRAQM, we use the classical pair generation in classical time 2^{128}, which we combine with a quantum early abort step in quantum time $2^{117.46}$. The attack is now a Q1 attack since only the pair generation requires queries. The attack is faster than a Grover search of the full 256-bit secret key, which would be slightly above 2^{128} quantum operations (and the difference is more significant if classical operations are cheaper than quantum ones).

6.4 Application to 7-round AES

In this section, we use our framework to improve the key-recovery attacks against 7-round AES-192 and AES-256 in the quantum setting (the best previous attacks were Square attacks in [BNS19], see Table 6.4 for a comparison with previous work). The path of the attack that we consider is the one from [Mal+10] (Figure 6.3).

Table 6.3: Time and memory complexities of the different steps for 21-round SKINNY-128-256 (in encryptions), whether classical or quantum.

Complexity	Time	Mem.
Classical Pair Generation	2^{128}	negl.
Quantum Pair Generation	$2^{119.17}$	2^{24} (QRAQM)
Classical early abort	$2^{151.17}$	$2^{103.17}$
Quantum early abort (QRAQM)	$2^{104.32}$	$2^{95.17}$ (QRAQM) + $2^{103.17}$ (classical)
Quantum early abort (no QRAQM)	$2^{117.46}$	$2^{106.31}$ (qubits) + $2^{103.17}$ (classical)
Quantum Generic	$2^{129.65}$	negl.
Classical Generic	2^{256}	negl.

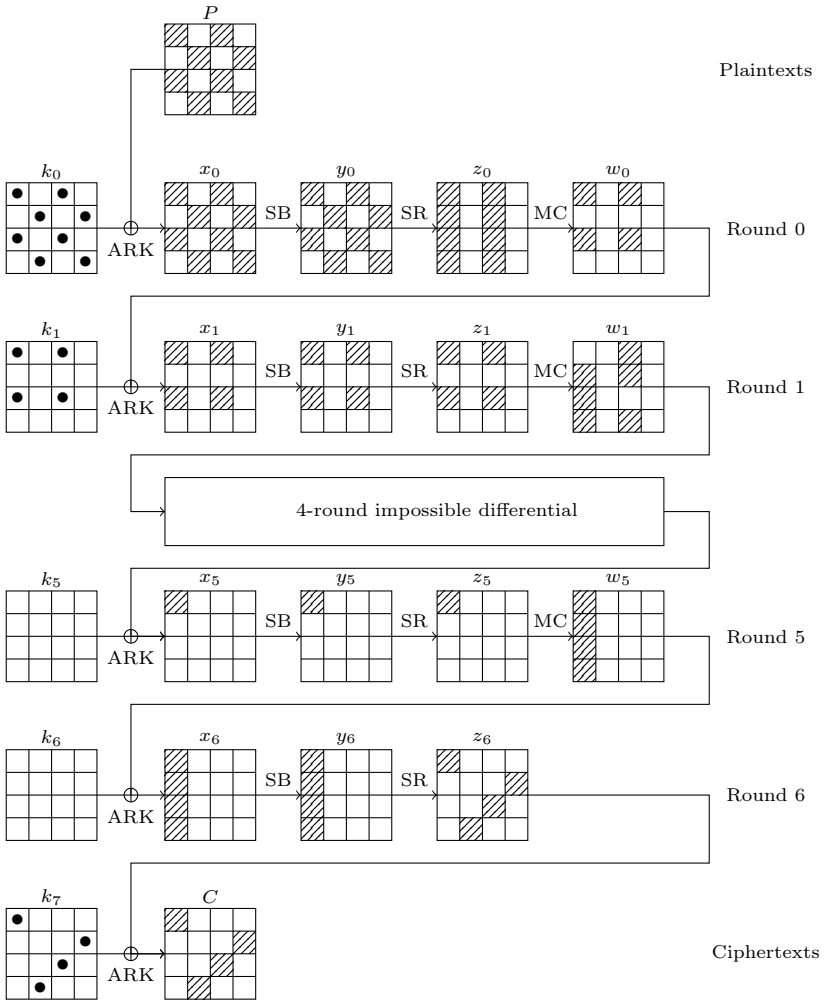


Figure 6.3: Impossible differential attack on 7-round AES

Table 6.4: Summary of best quantum attacks on AES-192/256 (with lower complexities than Grover's search). C = Classical; Q = Quantum; $*$ = QRAQM model. Q1, Q2 and QRAQM are defined in Section 1.4. $**$ means we have extrapolated the complexity on 21 rounds from the original attack on 24 rounds for comparison. We obtain a better memory than [DFJ13], and a time complexity comparable to [BNS19] (the quantum time is counted in S-Box evaluations).

Algorithm	Rds	Ref.	Time		Memory		Data	Setting
			C	Q	C	Q		
AES-192	7	[BNS19]-Grover		$2^{105.6}$		negl.	negl.	Q1
AES-256	7	[BNS19]-Grover		$2^{137.3}$		negl.	negl.	Q1
AES-256	7	[BNS19]-Square		2^{121}	2^{38}	$(2^{27})^*$	2^{37}	Q1
AES-192	7	[BNS19]-Square		$2^{103.4}$	2^{38}	$(2^{27})^*$	2^{37}	Q1
AES-256	7	[BNS19]-Square		2^{107}	2^{38}	$(2^{27})^*$	2^{37}	Q1
AES-256	7	[DFJ13]	$2^{99.6}$		2^{96}		2^{99}	C
AES-192/256	7	Section 6.4		$2^{101.5}$		$(2^{78.5})^*$	$2^{99.8}$	Q2
AES-192/256	7	Section 6.4		$2^{99.9}$		$(2^{78.5})^*$	$2^{99.8}$	Q2
AES-256	8	[BNS19]-Grover		2^{138}		negl.	negl.	Q1
AES-256	8	[BNS19]-DS-MITM		2^{136}	2^{88}	negl.	2^{88}	Q1

6.4.1 Description of `AES-256`

The Advanced Encryption Standard [FIP01] is a Substitution-Permutation Network (SPN) that can be instantiated using three different key sizes: 128, 192, and 256 bits. The 128-bit plaintext initializes the internal state viewed as a 4×4 matrix of bytes, i.e. values in the finite field $\mathbb{F}_{256}$, which is defined using the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$ over $\mathbb{F}_2$. Depending on the version of the `AES`, N_r rounds are applied to that state and $N_r = 14$ for the 256-bit version. Each of the N_r rounds (see Figure 6.4) applies four operations to the state matrix (except in the last round where the `MixColumns` operation is omitted):

- `AddRoundKey` (`AK`) adds a 128-bit subkey to the state.
- `SubBytes` (`SB`) applies the same 8-bit to 8-bit invertible S-Box $\mathbf{S}$ 16 times in parallel on each byte of the state.
- `ShiftRows` (`SR`) shifts the i-th row left by i positions.
- `MixColumns` (`MC`) replaces each of the four columns C of the state by $M \times C$ where M is a constant 4×4 maximum distance separable matrix over $\mathbb{F}_{256}$.

After the N_r-th round has been applied, a final subkey is added to the internal state to produce the ciphertext. The key expansion algorithm to produce the $N_r + 1$ subkeys is described in Figure 6.5. We refer to the original publication [FIP01] for further details.

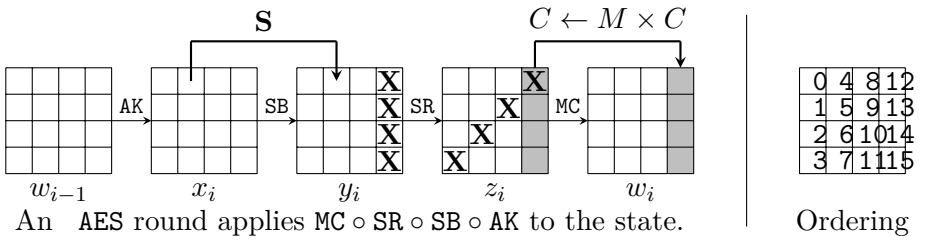


Figure 6.4: Description of one AES round and the ordering of bytes in an internal state

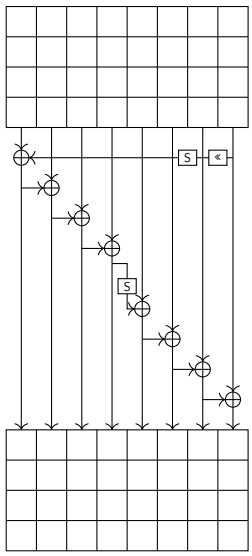


Figure 6.5: Key schedule of AES-256

Attack. Let $(P, P'), (C, C')$ be a pair satisfying the limited birthday conditions on Figure 6.3. It will invalidate any guess of these 16 bytes of key such that:

1. $w_0[1] = w'_0[1]$ and $w_0[3] = w'_0[3]$: this is a 2-byte condition on $k_0[0, 5, 10, 15]$, $p_0[0, 5, 10, 15]$ and $p'_0[0, 5, 10, 15]$. For a given pair, there exists on average 2^{16} matching guesses of $k_0[0, 5, 10, 15]$.

2. $w_0[9] = w'_0[9]$ and $w_0[11] = w'_0[11]$: this is an independent 2-byte condition on $k_0[2, 7, 8, 13], p_0[2, 7, 8, 13]$ and $p'_0[2, 7, 8, 13]$. For a given pair, there are on average 2^{16} matching guesses of $k_0[2, 7, 8, 13]$.

3. $MC^{-1}(w_5[0, 1, 2, 3])$ is a single byte: this is an independent condition on $k_7[0, 7, 10, 14]$ and the corresponding ciphertext bytes. For a given pair, there are on average 2^8 guesses of k_7 that satisfy this condition.

Table 6.5: Successive steps of the attack on AES (if we used the standard early abort). The parameter s corresponds to a byte (8 bits).

Index	σ_i	$\lvert K_i\rvert$	Quantum partial complexity: $\prod_{j=1}^{i-1}\sigma_j\sqrt{\prod_{j=1}^{i}\lvert K_j\rvert}$	Classical partial complexity: $\prod_{j=1}^{i-1}\sigma_j\prod_{j=1}^{i}\lvert K_j\rvert$
1.	2^{0}	2^{4s}	2^{2s}	2^{4s}
2.	2^{-2s}	2^{4s}	2^{2s}	2^{6s}
3.	2^{-2s}	2^{4s}	2^{2s}	2^{8s}
4.	2^{-3s}	2^{4s}	2^{s}	2^{9s}

4.a $w_1[0] = w'_1[0]$: this is a single-byte condition on $k_1[0, 10]$, and all the previous plaintext bytes. For a given pair, there are on average 2^8 matching guesses of $k_1[0, 10]$.

4.b $w_1[10] = w'_1[10]$: this is another single-byte condition with independent key bytes $k_1[2, 8]$, but the same plaintext bytes as before.

Thus each pair removes 2^{56} key guesses from a space 2^{128}. In order to have a single remaining key guess, we take $N = 2^{78.5}$. These pairs can be obtained in about $2^{99.8}$ Q2 queries. There are 4 subkey spaces: $K_1 = k_0[0, 5, 10, 15]$, $K_2 = k_0[2, 7, 8, 13]$, $K_3 = k_7[0, 7, 10, 14]$ and $K_4 = k_1[0, 10, 2, 8]$, each of size 2^{32}. The filtering probabilities are $\sigma_1 = 2^{-16}$, $\sigma_2 = 2^{-16}$, $\sigma_3 = 2^{-24}$. Similarly to Table 6.2, we summarize the successive steps in Table 6.5.

Using the simple early-abort strategy, we obtain a classical time complexity:

$$2^{32}\Big(\underbrace{2^{78.5}}_{\text{Build } \mathcal{T}_1} + 2^{32}\Big(\underbrace{2^{62.5}}_{\text{Build } \mathcal{T}_2} + 2^{32}\Big(\underbrace{2^{46.5}}_{\text{Build } \mathcal{T}_3} + 2^{32}\Big(\underbrace{2^{22.5}}_{\text{Sieve } \mathcal{T}_3}\Big)\Big)\Big)\Big) = 2^{150.5} \ .$$

And using Corollary 6.1, we obtain a quantum time complexity:

$$2N\Big(\left(\frac{\pi}{2}\right)\sqrt{|K_1|} + \left(\frac{\pi}{2}\right)^2\sigma_1\sqrt{|K_1||K_2|} + \left(\frac{\pi}{2}\right)^3\sigma_1\sigma_2\sqrt{|K_1||K_2||K_3|} + \left(\frac{\pi}{2}\right)^4\sigma_1\sigma_2\sigma_3\sqrt{|K_1||K_2||K_3||K_4|}\Big)\max_i(t+t_i) \ ,$$

where $t + t_i$ is the time to evaluate the test functions relatively to an encryption. In the case of AES, to facilitate comparison with [BNS19], we count the number of S-Boxes, the S-Box being the most costly component in quantum circuits for the AES. We remark that computing each test function requires 8 S-Boxes, so we approximate the factor $t + t_i$ by 8. This gives:

$$2^{79.5}\cdot 8\cdot\left(\left(\frac{\pi}{2}\right)2^{16} + \left(\frac{\pi}{2}\right)^2 2^{16} + \left(\frac{\pi}{2}\right)^3 2^{16} + \left(\frac{\pi}{2}\right)^4 2^{8}\right) \leq 2^{79.5}\cdot 2^3\cdot 2^{19} \leq 2^{101.5} \ .$$

We use $2^{78.5}$ qubits with QRAQM access to store the pairs.

The previous best quantum attacks on 7-round AES-192 and AES-256 are the quantum Square attacks of [BNS19], with respectively $2^{102.73}$ and $2^{106.73}$ reversible S-Boxes. So far our attack is comparable.

Quantum early abort. Following the discussion in Section 6.2.4, we can use quantum searches to speed up the construction of successive tables. The table $\mathcal{T}_1$, which is of expected size $2^{62.5}$, is constructed in time $4\left(\frac{\pi}{2}\right)\sqrt{\sigma_1}|\mathcal{T}_1|$ instead of $|\mathcal{T}_1|$. We do likewise for $\mathcal{T}_3$. Finally, when sieving $\mathcal{T}_3$ with respect to a guess of $k_1[0,10,2,8]$, we can use a quantum search instead of going through the table. The time complexity becomes:

$$\left(\frac{\pi}{2}\right)\sqrt{|K_1|}\Bigg(4N\left(\frac{\pi}{2}\right)\sqrt{\sigma_1}+\left(\frac{\pi}{2}\right)\sqrt{|K_2|}\Bigg(4N\sigma_1\left(\frac{\pi}{2}\right)\sqrt{\sigma_2}+$$
$$\left(\frac{\pi}{2}\right)\sqrt{|K_3|}\Bigg(4N\sigma_1\sigma_2\left(\frac{\pi}{2}\right)\sqrt{\sigma_3}+\left(\frac{\pi}{2}\right)\sqrt{|K_4|}\left(\left(\frac{\pi}{2}\right)\sqrt{2N\sigma_1\sigma_2\sigma_3}\right)\Bigg)\Bigg)\Bigg)\max_i(t+t_i)$$
$$=\max_i(t+t_i)\Bigg(\left(\frac{\pi}{2}\right)^5\sqrt{|K_1||K_2||K_3||K_4|}\sqrt{2N\sigma_1\sigma_2\sigma_3}+$$
$$4N\Bigg(\left(\frac{\pi}{2}\right)^2\sqrt{|K_1|\sigma_1}+\left(\frac{\pi}{2}\right)^3\sqrt{|K_1||K_2|}\sigma_1\sqrt{\sigma_2}+\left(\frac{\pi}{2}\right)^4\sqrt{|K_1||K_2||K_3|}\sigma_1\sigma_2\sqrt{\sigma_3}\Bigg)\Bigg)$$

We can now replace with our numerical values, obtaining:

$$8\cdot\left(\left(\frac{\pi}{2}\right)^5 2^{128/2}2^{23.5/2}+2^{80.5}\left(\left(\frac{\pi}{2}\right)^2 2^8+\left(\frac{\pi}{2}\right)^3 2^8+\left(\frac{\pi}{2}\right)^4 2^8\right)\right)\leq 2^{95.2}\ ,$$

where the dominant term is the computation of the table $\mathcal{T}_3$. The memory complexity stays the same as before.

6.5 Conclusion

In this chapter, we provided a quantized version of impossible differential attacks answering the open problem from [BNS19]. Our framework, which includes a generic complexity analysis, could be a good tool for analyzing the post-quantum security margin of symmetric primitives, as we have shown with `SKINNY`-128-256. Regarding AES, we have only been able to provide limited improvements of the 7-round attacks of [BNS19]. Currently, our best attack is limited by the generation of pairs. While the key recovery step benefits significantly from quantum search, potentially up to a complete quadratic speedup on its complexity, the first step seems a relatively more difficult problem in the quantum setting, as the limited birthday problem reaches a less than quadratic speedup. Consequently, as observed in [BNS19] with other attack families, quantum impossible differential attacks need to be optimized differently from the classical ones to balance the complexity terms of the different steps in the quantum setting. Indeed, one could for example try to reduce the data used as much as possible, and in particular, to make the pair generation step completely classical (thereby falling in the Q1 setting).

Conclusion

In this thesis, I have presented a collection of scientific contributions that come from collaborations with other cryptographers. All of these unprecedented human experiences were very rich in science and allowed me to develop expertise in differential cryptanalysis. As shown through this thesis, differential cryptanalysis is, under many forms, a very powerful cryptanalytic technique in the context of block ciphers. Indeed, in Chapter 4, we have shown that even the block ciphers published in the most renowned conferences can show critical weaknesses to differential attacks. However, building differential attacks can be a complex and time-consuming task since it requires looking into the details of the round function operations to exhibit weaknesses to exploit. In this regard, the development of an automatic tool would have the potential to greatly facilitate the analysis process of the designers. Indeed, the choice of security claims could be more refined hence optimizing efficiency while preserving security. However, designing an automatic tool is a difficult task since new techniques and refinements are continually proposed and would need to be taken into account for the tool to be complete.

The design of an automatic differential tool is part of an ongoing work. The tool will take as input a description of the cipher (round function and key schedule), a differential trail (input and output difference with associated probability and length) as well as the total number of rounds R that we want to attack. It will output a description of the best differential attack on R rounds together with its complexities, hence a description of the key recovery rounds as well as a description of the key recovery procedure. The first step is to obtain a tool that would automatically find the attack described in Chapter 4 (or a better attack) when applied on `SPEEDY-7-192`. As a consequence, the tool needs to encapsulate all the known optimizations starting with the one we used in the attack such as *probability-sieving trade-off*, *parallel merging*, and *additional input-output siveing*. The most challenging task is to design the optimization for the early abort technique since it requires our algorithm to recognize the ordering on the propagation conditions that would minimize the complexity. Indeed, in some cases, trying all the orders require too many operations. Therefore, the first step is to find a good model in which we can describe the key recovery optimization problem.

Another open problem concerns differential-MITM cryptanalysis depicted in Chapter 5. Indeed, we could wonder about the application of this technique to constructions other than blockciphers, for instance, hash functions. In addition, the techniques described in Chapter 5 can be combined with the ones from [EFK15] that

allow to exploit differential in the key part. Therefore we can expect this combination of techniques to result in a powerful preimage attack.

Bibliography

[ABK00] Ross J. Anderson, Eli Biham, and Lars R. Knudsen. "The Case for Serpent". In: *The Third Advanced Encryption Standard Candidate Conference, April 13-14, 2000, New York, New York, USA*. National Institute of Standards and Technology, 2000, pp. 349–354 (cit. on p. 40).

[Amb07] Andris Ambainis. "Quantum Walk Algorithm for Element Distinctness". In: *SIAM J. Comput.* 37.1 (2007), pp. 210–239 (cit. on pp. 13, 88).

[AS08] Kazumaro Aoki and Yu Sasaki. "Preimage Attacks on One-Block MD4, 63-Step MD5 and More". In: *SAC 2008*. Ed. by Roberto Maria Avanzi, Liam Keliher, and Francesco Sica. Vol. 5381. LNCS. Springer, 2008, pp. 103–119 (cit. on p. 10).

[AS09] Kazumaro Aoki and Yu Sasaki. "Meet-in-the-Middle Preimage Attacks Against Reduced SHA-0 and SHA-1". In: *CRYPTO 2009*. Ed. by Shai Halevi. Vol. 5677. LNCS. Springer, 2009, pp. 70–89 (cit. on p. 10).

[Ban+17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. "GIFT: A Small Present - Towards Reaching the Limit of Lightweight Encryption". In: *CHES 2017*. Vol. 10529. LNCS. Springer, 2017, pp. 321–345 (cit. on p. 70).

[Bat68] Kenneth E. Batcher. "Sorting Networks and Their Applications". In: *AFIPS Spring Joint Computing Conference*. Vol. 32. AFIPS Conference Proceedings. Thomson Book Company, Washington D.C., 1968, pp. 307–314 (cit. on p. 94).

[BBS05] Eli Biham, Alex Biryukov, and Adi Shamir. "Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials". In: *J. Cryptol.* 18.4 (2005), pp. 291–311 (cit. on p. 24).

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. "Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials". In: *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*. Ed. by Jacques Stern. Vol. 1592. Lecture Notes in Computer Science. Springer, 1999, pp. 12–23 (cit. on p. 18).

[BDK02] Eli Biham, Orr Dunkelman, and Nathan Keller. "Enhancing Differential-Linear Cryptanalysis". In: *ASIACRYPT 2002, Proceedings*. Ed. by Yuliang Zheng. Vol. 2501. LNCS. Berlin, Heidelberg: Springer, 2002, pp. 254–266 (cit. on p. 28).

[BDL20] Augustin Bariant, Nicolas David, and Gaëtan Leurent. "Cryptanalysis of Forkciphers". In: *IACR Trans. Symmetric Cryptol.* 2020.1 (2020), pp. 233–265.

[BDQ04] Alex Biryukov, Christophe De Canniere, and Michaël Quisquater. "On multiple linear approximations". In: *Advances in Cryptology–CRYPTO 2004: 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004. Proceedings 24*. Springer. 2004, pp. 1–22 (cit. on p. 9).

[Bea+15] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. "The SIMON and SPECK lightweight block ciphers". In: *Proceedings of the 52nd annual design automation conference*. 2015, pp. 1–6 (cit. on p. 4).

[Bei+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. "The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS". In: *CRYPTO 2016, Part II*. Vol. 9815. LNCS. Springer, 2016, pp. 123–153 (cit. on pp. 70, 73–75).

[Bei+22] Christof Beierle, Marek Broll, Federico Canale, Nicolas David, Antonio Flórez-Gutiérrez, Gregor Leander, María Naya-Plasencia, and Yosuke Todo. "Improved Differential-Linear Attacks with Applications to ARX Ciphers". In: *J. Cryptol.* 35.4 (2022), p. 29 (cit. on pp. xiii, 31, 36, 38).

[Ben+97] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh V. Vazirani. "Strengths and Weaknesses of Quantum Computing". In: *SIAM J. Comput.* 26.5 (1997), pp. 1510–1523 (cit. on p. 97).

[BG11] Céline Blondeau and Benoît Gérard. "Multiple Differential Cryptanalysis: Theory and Practice". In: *FSE 2011*. Vol. 6733. LNCS. Springer, 2011, pp. 35–54 (cit. on p. 17).

[BHT98] Gilles Brassard, Peter Høyer, and Alain Tapp. "Quantum Cryptanalysis of Hash and Claw-Free Functions". In: *LATIN*. Vol. 1380. Lecture Notes in Computer Science. Springer, 1998, pp. 163–169 (cit. on p. 88).

[Bih+01] Eli Biham, Vladimir Furman, Michal Misztal, and Vincent Rijmen. "Differential Cryptanalysis of Q". In: *FSE 2001*. Vol. 2355. LNCS. Springer, 2001, pp. 174–186 (cit. on p. 17).

[BLT20] Christof Beierle, Gregor Leander, and Yosuke Todo. "Improved Differential-Linear Attacks with Applications to ARX Ciphers". In: *CRYPTO 2020, Proceedings, Part III*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12172. LNCS. Cham: Springer, 2020, pp. 329–358 (cit. on pp. xiii, 31–33, 38, 39).

[BNS14] Christina Boura, María Naya-Plasencia, and Valentin Suder. "Scrutinizing and Improving Impossible Differential Attacks: Applications to CLEFIA, Camellia, LBlock and Simon". In: *ASIACRYPT (1)*. Vol. 8873. Lecture Notes in Computer Science. Springer, 2014, pp. 179–199 (cit. on pp. 25, 26).

[BNS19] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. "Quantum Security Analysis of AES". In: *IACR Trans. Symmetric Cryptol.* 2019.2 (2019), pp. 55–93 (cit. on pp. xvi, 87, 96, 102, 104, 106, 107).

[Bon+23] Xavier Bonnetain, André Chailloux, André Schrottenloher, and Yixin Shen. "Finding many Collisions via Reusable Quantum Walks". In: *EUROCRYPT 2023, to appear*. Lecture Notes in Computer Science. Available at: https://eprint.iacr.org/2022/676. 2023 (cit. on pp. 13, 88, 89).

[Bor+21] Nicolas Bordes, Joan Daemen, Daniël Kuijsters, and Gilles Van Assche. "Thinking Outside the Superbox". In: *CRYPTO 2021, Part III*. Ed. by Tal Malkin and Chris Peikert. Vol. 12827. LNCS. Springer, 2021, pp. 337–367 (cit. on p. 73).

[Bou+23a] Christina Boura, Nicolas David, Rachelle Heim Boissier, and María Naya-Plasencia. "Better Steady than Speedy: Full Break of SPEEDY-7-192". In: *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part IV*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14007. Lecture Notes in Computer Science. Springer, 2023, pp. 36–66 (cit. on pp. xiv, 43, 47, 58).

[Bou+23b] Christina Boura, Nicolas David, Patrick Derbez, Gregor Leander, and María Naya-Plasencia. "Differential Meet-In-The-Middle Cryptanalysis". In: *Crypto 2023* (2023), p. 1640 (cit. on pp. 67, 84).

[BR10] Andrey Bogdanov and Christian Rechberger. "A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN". In: *SAC 2010*. Ed. by Alex Biryukov, Guang Gong, and Douglas R. Stinson. Vol. 6544. LNCS. Springer, 2010, pp. 229–240 (cit. on p. 10).

[Bra+02] Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. "Quantum amplitude amplification and estimation". In: *Contemporary Mathematics* 305 (2002), pp. 53–74 (cit. on p. 13).

[Bro+21] Marek Broll, Federico Canale, Antonio Flórez-Gutiérrez, Gregor Leander, and María Naya-Plasencia. "Generic Framework for Key-Guessing Improvements". In: *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part I*. Ed. by Mehdi Tibouchi and Huaxiong Wang. Vol. 13090. Lecture Notes in Computer Science. Springer, 2021, pp. 453–483 (cit. on p. 40).

[Bro+22a] Marek Broll, Federico Canale, Nicolas David, Antonio Flórez-Gutiérrez, Gregor Leander, María Naya-Plasencia, and Yosuke Todo. "New Attacks from Old Distinguishers Improved Attacks on Serpent". In: *Topics in Cryptology - CT-RSA 2022 - Cryptographers' Track at the RSA Conference 2022, Virtual Event, March 1-2, 2022, Proceedings*. Ed. by Steven D. Galbraith. Vol. 13161. Lecture Notes in Computer Science. Springer, 2022, pp. 484–510 (cit. on p. 40).

[Bro+22b] Marek Broll, Federico Canale, Nicolas David, Antonio Flórez-Gutiérrez, Gregor Leander, María Naya-Plasencia, and Yosuke Todo. "New Attacks from Old Distinguishers Improved Attacks on Serpent". In: *CT-RSA 2022*. Vol. 13161. LNCS. Springer, 2022, pp. 484–510 (cit. on p. 55).

[BS91a] Eli Biham and Adi Shamir. "Differential Cryptanalysis of Feal and N-Hash". In: *EUROCRYPT '91*. Vol. 547. LNCS. Springer, 1991, pp. 1–16 (cit. on pp. 17, 21).

[BS91b] Eli Biham and Adi Shamir. "Differential Cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer". In: *CRYPTO '91*. Vol. 576. LNCS. Springer, 1991, pp. 156–171 (cit. on p. 17).

[BS92] Eli Biham and Adi Shamir. "Differential Cryptanalysis of the Full 16-Round DES". In: *CRYPTO '92*. Vol. 740. LNCS. Springer, 1992, pp. 487–496 (cit. on p. 17).

[Can+14] Anne Canteaut, Thomas Fuhr, Henri Gilbert, María Naya-Plasencia, and Jean-René Reinhard. "Multiple Differential Cryptanalysis of Round-Reduced PRINCE". In: *FSE 2014*. Vol. 8540. LNCS. Springer, 2014, pp. 591–610 (cit. on p. 17).

[CE05] Andrew M. Childs and Jason M. Eisenberg. "Quantum algorithms for subset finding". In: *Quantum Inf. Comput.* 5.7 (2005), pp. 593–604 (cit. on p. 88).

[CM16] Arka Rai Choudhuri and Subhamoy Maitra. *Differential Cryptanalysis of Salsa and ChaCha – An Evaluation with a Hybrid Model*. Cryptology ePrint Archive, Paper 2016/377. https://eprint.iacr.org/2016/377. 2016 (cit. on p. 17).

[CNV13a] Anne Canteaut, María Naya-Plasencia, and Bastien Vayssière. "Sieve-in-the-Middle: Improved MITM Attacks". In: *CRYPTO 2013, Part I*. Vol. 8042. LNCS. Springer, 2013, pp. 222–240 (cit. on pp. xv, 62).

[CNV13b] Anne Canteaut, María Naya-Plasencia, and Bastien Vayssière. "Sieve-in-the-Middle: Improved MITM Attacks". In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, 2013, pp. 222–240 (cit. on pp. xv, 10, 68, 72).

[CNV13c] Anne Canteaut, María Naya-Plasencia, and Bastien Vayssière. "Sieve-in-the-Middle: Improved MITM Attacks (Full Version)". In: *IACR Cryptol. ePrint Arch.* (2013), p. 324 (cit. on pp. xv, 68).

[De 19] Ronald De Wolf. "Quantum computing: Lecture notes". In: *arXiv preprint arXiv:1907.09415* (2019) (cit. on p. 13).

[DEH22] Nicolas David, Thomas Espitau, and Akinori Hosoyamada. "Quantum binary quadratic form reduction". In: *IACR Cryptol. ePrint Arch.* (2022), p. 466.

[Del+21] Stéphanie Delaune, Patrick Derbez, Paul Huynh, Marine Minier, Victor Mollimard, and Charles Prud'homme. "Efficient Methods to Search for Best Differential Characteristics on SKINNY". In: *ACNS 2021, Part II*. Ed. by Kazue Sako and Nils Ole Tippenhauer. Vol. 12727. LNCS. Springer, 2021, pp. 184–207 (cit. on p. 76).

[DFJ13] Patrick Derbez, Pierre-Alain Fouque, and Jérémy Jean. "Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting". In: *EUROCRYPT*. Vol. 7881. Lecture Notes in Computer Science. Springer, 2013, pp. 371–387 (cit. on p. 104).

[DH77a] Whitfield Diffie and Martin Hellman. "Special Feature Exhaustive Cryptanalysis of the NBS Data Encryption Standard". In: *Computer* 10.6 (1977), pp. 74–84 (cit. on pp. 4, 9, 72).

[DH77b] Whitfield Diffie and Martin E. Hellman. "Special Feature Exhaustive Cryptanalysis of the NBS Data Encryption Standard". In: *Computer* 10.6 (1977), pp. 74–84 (cit. on p. 68).

[DNS22] Nicolas David, María Naya-Plasencia, and André Schrottenloher. "Quantum impossible differential attacks: Applications to AES and SKINNY". In: *Des. Codes Cryptogr.* (2022), p. 754 (cit. on pp. xvi, 87).

[Don+21] Xiaoyang Dong, Jialiang Hua, Siwei Sun, Zheng Li, Xiaoyun Wang, and Lei Hu. "Meet-in-the-Middle Attacks Revisited: Key-Recovery, Collision, and Preimage Attacks". In: *CRYPTO 2021, Part III*. Ed. by Tal Malkin and Chris Peikert. Vol. 12827. LNCS. Springer, 2021, pp. 278–308 (cit. on p. 73).

[DR02] Joan Daemen and Vincent Rijmen. *The design of Rijndael*. Vol. 2. Springer, 2002 (cit. on p. 3).

[DR99] Joan Daemen and Vincent Rijmen. *AES proposal: Rijndael*. Submission to NIST AES competition. 1999 (cit. on p. 3).

[DSP07] Orr Dunkelman, Gautham Sekar, and Bart Preneel. "Improved Meet-in-the-Middle Attacks on Reduced-Round DES". In: *INDOCRYPT 2007*. Ed. by K. Srinathan, C. Pandu Rangan, and Moti Yung. Vol. 4859. LNCS. Springer, 2007, pp. 86–100 (cit. on p. 10).

[EFK15] Thomas Espitau, Pierre-Alain Fouque, and Pierre Karpman. "Higher-Order Differential Meet-in-the-middle Preimage Attacks on SHA-1 and BLAKE". In: *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*. Ed. by Rosario Gennaro and Matthew Robshaw. Vol. 9215. Lecture Notes in Computer Science. Springer, 2015, pp. 683–701 (cit. on p. 109).

[Fei73] Horst Feistel. "Cryptography and computer privacy". In: *Scientific american* 228.5 (1973), pp. 15–23 (cit. on p. 70).

[FIP01] FIPS 197. *Announcing the Advanced Encryption Standard (AES)*. National Institute for Standards and Technology, Gaithersburg, MD, USA. Nov. 2001 (cit. on pp. 4, 104).

[Fló22] Antonio Flórez-Gutiérrez. "Optimising linear key recovery attacks with affine Walsh transform pruning". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2022, pp. 447–476 (cit. on p. 9).

[GP10] Henri Gilbert and Thomas Peyrin. "Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations". In: *FSE*. Vol. 6147. Lecture Notes in Computer Science. Springer, 2010, pp. 365–383 (cit. on p. 26).

[Gro96] Lov K. Grover. "A Fast Quantum Mechanical Algorithm for Database Search". In: *STOC*. ACM, 1996, pp. 212–219 (cit. on p. 13).

[Guo+10] 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*. Ed. by Masayuki Abe. Vol. 6477. LNCS. Springer, 2010, pp. 56–75 (cit. on p. 10).

[HNS20] Akinori Hosoyamada, María Naya-Plasencia, and Yu Sasaki. "Improved Attacks on sLiSCP Permutation and Tight Bound of Limited Birthday Distinguishers". In: *IACR Trans. Symmetric Cryptol.* 2020.4 (2020), pp. 147–172 (cit. on p. 26).

[Iso11] Takanori Isobe. "A Single-Key Attack on the Full GOST Block Cipher". In: *FSE 2011*. Ed. by Antoine Joux. Vol. 6733. LNCS. Springer, 2011, pp. 290–305 (cit. on p. 10).

[JNP14] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. "Tweaks and Keys for Block Ciphers: The TWEAKEY Framework". In: *ASIACRYPT 2014, Part II*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8874. LNCS. Springer, 2014, pp. 274–288 (cit. on p. 74).

[Kap+16] Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. "Quantum Differential and Linear Cryptanalysis". In: *IACR Trans. Symmetric Cryptol.* 2016.1 (2016), pp. 71–94 (cit. on pp. 13, 88, 89).

[Ker83] Auguste Kerckhoffs. "La cryptographie militaire". In: *J. Sci. Militaires* 9.4 (1883), pp. 5–38 (cit. on p. 2).

[KMN10a] Simon Knellwolf, Willi Meier, and María Naya-Plasencia. "Conditional Differential Cryptanalysis of NLFSR-Based Cryptosystems". In: *ASIACRYPT 2010*. Vol. 6477. LNCS. Springer, 2010, pp. 130–145 (cit. on p. 17).

[KMN10b] Simon Knellwolf, Willi Meier, and María Naya-Plasencia. "Conditional Differential Cryptanalysis of NLFSR-Based Cryptosystems". In: *ASIACRYPT 2010, Proceedings*. Ed. by Masayuki Abe. Vol. 6477. LNCS. Berlin, Heidelberg: Springer, 2010, pp. 130–145 (cit. on pp. 34, 35).

[Knu94] Lars R. Knudsen. "Truncated and Higher Order Differentials". In: *FSE 1994*. Vol. 1008. LNCS. Springer, 1994, pp. 196–211 (cit. on p. 17).

[Knu98] Lars Knudsen. "DEAL-a 128-bit block cipher". In: *complexity* 258.2 (1998), p. 216 (cit. on pp. 18, 24).

[KR94] Burton S Kaliski and Matthew JB Robshaw. "Linear cryptanalysis using multiple approximations". In: *Advances in Cryptology—CRYPTO'94: 14th Annual International Cryptology Conference Santa Barbara, California, USA August 21–25, 1994 Proceedings 14*. Springer. 1994, pp. 26–39 (cit. on p. 9).

[KRS12] Dmitry Khovratovich, Christian Rechberger, and Alexandra Savelieva. "Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family". In: *FSE 2012*. Ed. by Anne Canteaut. Vol. 7549. LNCS. Springer, 2012, pp. 244–263 (cit. on p. 10).

[Kup13] Greg Kuperberg. "Another Subexponential-time Quantum Algorithm for the Dihedral Hidden Subgroup Problem". In: *TQC*. Vol. 22. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013, pp. 20–34 (cit. on p. 13).

[Lea+21] Gregor Leander, Thorben Moos, Amir Moradi, and Shahram Rasoolzadeh. "The SPEEDY Family of Block Ciphers Engineering an Ultra Low-Latency Cipher from Gate Level for Secure Processor Architectures". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.4 (2021), pp. 510–545 (cit. on pp. xiv, 43, 44).

[Leu16a] Gaëtan Leurent. "Improved Differential-Linear Cryptanalysis of 7-Round Chaskey with Partitioning". In: *EUROCRYPT 2016, Proceedings, Part I*. 2016, pp. 344–371 (cit. on p. 38).

[Leu16b] Gaëtan Leurent. "Improved Differential-Linear Cryptanalysis of 7-Round Chaskey with Partitioning". In: *EUROCRYPT 2016, Part I*. Vol. 9665. LNCS. Springer, 2016, pp. 344–371 (cit. on p. 17).

[LH94] Susan K. Langford and Martin E. Hellman. "Differential-Linear Cryptanalysis". In: *CRYPTO '94, Proceedings*. Ed. by Yvo Desmedt. Vol. 839. LNCS. Berlin, Heidelberg: Springer, 1994, pp. 17–25 (cit. on pp. 18, 28).

[LM91] Xuejia Lai and James L Massey. "A proposal for a new block encryption standard". In: *Advances in Cryptology—EUROCRYPT'90: Workshop on the Theory and Application of Cryptographic Techniques Aarhus, Denmark, May 21–24, 1990 Proceedings 9*. Springer. 1991, pp. 389–404 (cit. on p. 4).

[LZ19] Qipeng Liu and Mark Zhandry. "On Finding Quantum Multi-collisions". In: *EUROCRYPT (3)*. Vol. 11478. Lecture Notes in Computer Science. Springer, 2019, pp. 189–218 (cit. on p. 88).

[Mal+10] Hamid Mala, Mohammad Dakhilalian, Vincent Rijmen, and Mahmoud Modarres-Hashemi. "Improved Impossible Differential Cryptanalysis of 7-Round AES-128". In: *INDOCRYPT*. Vol. 6498. Lecture Notes in Computer Science. Springer, 2010, pp. 282–291 (cit. on p. 102).

[Mat93] Mitsuru Matsui. "Linear Cryptanalysis Method for DES Cipher". In: *EUROCRYPT '93*. Ed. by Tor Helleseth. Vol. 765. LNCS. Springer, 1993, pp. 386–397 (cit. on pp. 7–9).

[Mat94] Mitsuru Matsui. "The first experimental cryptanalysis of the Data Encryption Standard". In: *Annual International Cryptology Conference*. Springer. 1994, pp. 1–11 (cit. on p. 8).

[Men+09] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. "The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl". In: *FSE 2009*. Ed. by Orr Dunkelman. Vol. 5665. LNCS. Springer, 2009, pp. 260–276 (cit. on p. 72).

[Mer79] Ralph Charles Merkle. *Secrecy, authentication, and public key systems*. Stanford university, 1979 (cit. on p. 7).

[Mou+14] Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel, and Ingrid Verbauwhede. "Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers". In: *SAC 2014, Revised Selected Papers*. Ed. by Antoine Joux and Amr M. Youssef. Vol. 8781. LNCS. Cham: Springer, 2014, pp. 306–323 (cit. on p. 37).

[Mou15] Nicky Mouha. "Chaskey: a MAC Algorithm for Microcontrollers - Status Update and Proposal of Chaskey-12 -". In: *IACR Cryptol. ePrint Arch.* 2015 (2015). https://eprint.iacr.org/2015/1182, p. 1182 (cit. on p. 37).

[Nyb94] Kaisa Nyberg. "Linear approximation of block ciphers". In: *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer. 1994, pp. 439–444 (cit. on p. 9).

[PFL16] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Solver Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. 2016 (cit. on p. 76).

[RS22] Raghvendra Rohit and Santanu Sarkar. "Cryptanalysis of Reduced Round SPEEDY". In: *AFRICACRYPT 2022*. LNCS. Springer Nature Switzerland, 2022, pp. 133–149 (cit. on p. 47).

[SMB18] Sadegh Sadeghi, Tahereh Mohammadi, and Nasour Bagheri. "Cryptanalysis of Reduced round SKINNY Block Cipher". In: *IACR Trans. Symmetric Cryptol.* 2018.3 (2018), pp. 124–162 (cit. on pp. 98, 99).

[Sta+99] Data Encryption Standard et al. "Data encryption standard". In: *Federal Information Processing Standards Publication* 112 (1999) (cit. on p. 3).

[Wag99] David A. Wagner. "The Boomerang Attack". In: *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings*. Ed. by Lars R. Knudsen. Vol. 1636. Lecture Notes in Computer Science. Springer, 1999, pp. 156–170 (cit. on p. 18).

[Wan08] Meiqin Wang. "Differential Cryptanalysis of Reduced-Round PRESENT". In: *AFRICACRYPT 2008*. Vol. 5023. LNCS. Springer, 2008, pp. 40–49 (cit. on p. 17).