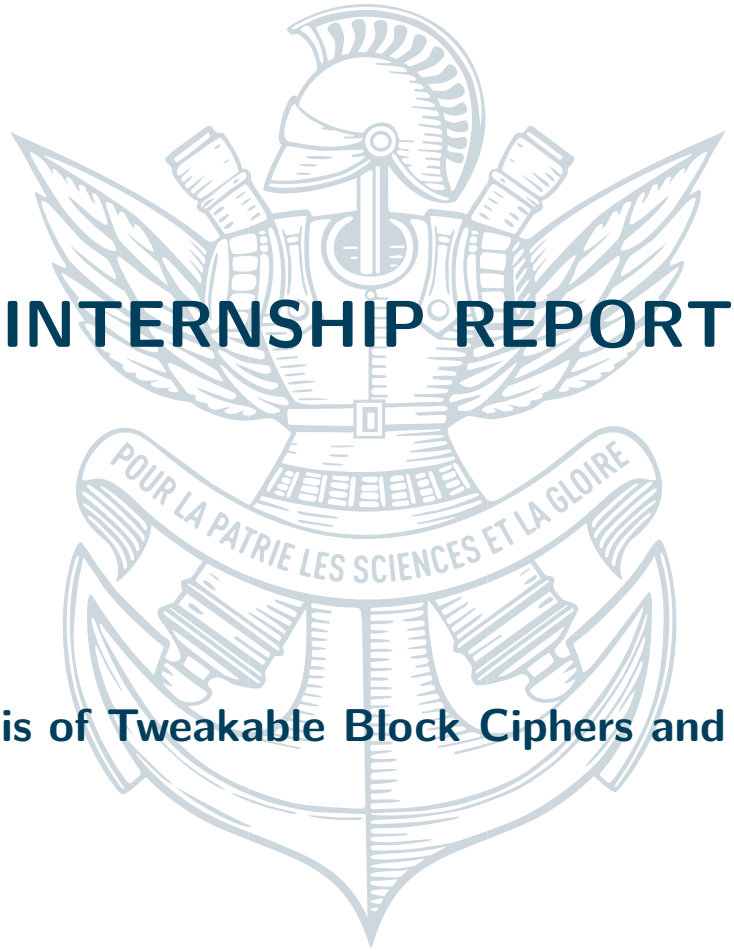
Cryptanalysis of Tweakable Block Ciphers and Forkciphers

Augustin Bariant



INTERNSHIP REPORT

Cryptanalysis of Tweakable Block Ciphers and Forkciphers

July 2, 2019

Augustin Bariant

CONTENTS

Acknowledgments I would like to thank Gaëtan LEURENT for all the pertinent explanations he provided me with and for the time we spent together working on very interesting problems. I would also like to thank Anne CANTEAUT who helped me find this internship in Inria, and all other members of the team SECRET for their caring, as they gently introduced me to the research sphere.

1 INTRODUCTION

1.1 Presentation of Inria and the Team SECRET

Inria is the main french public research institute in computer science. It is composed with more than 2500 researchers regrouped in 200 project-teams that work in specific advanced areas of digital science. In total, it implies more than 3000 scientists around the globe to meet the challenges of a constant evolving society and economy. Inria has also been supporting more than 160 start-ups, and is closely connected to numerous companies on state of the art problematics. Inria is divided into 8 etablissments implanted in France, among which Inria de Paris, previously located in Rocquencourt.

Team SECRET is a project-team of Inria de Paris, directed by Anne Canteaut and dedicated to symmetric cryptography, code-base cryptography, quantum information theory and post-quantum cryptanalysis. As modern cryptography is in jeopardy because of advanced cryptanalysis and the threat of quantum computers, new cryptographic primitives are being analysed and conceived by researchers of the team. I have mostly worked with my tutor Gaëtan Leurent on symmetric cryptography, but I was able to discuss with all the members of the team about crypto-related issues.

1.2 My Work in Inria

I started my internship studying KIASU-BC, a tweakable block cipher designed in 2014 by J. Jean et al. in the TWEAKEY framework [JNP14b] based on AES-128. I had to go through the entire paper to deeply understand the stakes behind every detail of the primitive. Then I studied existing attacks on the primitive, for example the square attack [DEM16], impossible differential and boomerang attacks [DL17]. I also took a close look to existing attacks on AES-128, because most of the attacks on KIASU-BC are variants of AES attacks. After more than two weeks, I could try to think by myself and started searching for attacks with a pencil and a notebook. Gaëtan Leurent advised me to study forkAES, as it is really close to KIASU-BC, but with a lack of diffusion which could lead to a vulnerability. I read the specifications of ForkAES by E. Andreeva et al. [ARVV18]. They created ForkAES as a primitive belonging to their forkcipher framework, using a block cipher to produce an authentificated encryption for very short messages. In response to this proposal, careful cryptanalysis was performed in "Cryptanalysis of ForkAES" by S. Banik et al. [BBJ+19]. The authors found really low complexity differential attacks on reduced-round ForkAES, which I found really interesting. I came up, with the crucial help of Gaëtan Leurent, with an attack on full ForkAES. Even though this attack is theoretical because the complexities are too high to be applicable, recovering the key requires less operations than the exhaustive search. ForkAES has not been standardized, but their designers produced another candidate based on the forkcipher framework for the NIST Lightweight Competition, ForkAE, using the ForkSkinny primitive.

I have a technical description of our attack on full ForkAES, which you will find in this report. We plan to submit this paper to the FSE conference in march 2020, the main conference in symmetric cryptography.

During the remaining month of my insternship, I will study different candidates of the NIST Lightweight Competition, which started in May. More than 50 candidate cryptography algorithms are being cryptanalyzed, and the winner of the competition will be standardized, like AES in the early 2000s.

1.3 Introduction to Tweakable Block Ciphers and Forkciphers

There are two types of cryptosystems : private key systems (a.k.a. symmetric cryptology) and public key systems (a.k.a. asymmetric cryptology). In symmetric cryptology, the paramount hypothesis is that the sender (Alice)

and the receiver (Bob) of a message share a common piece of information: a secret key. This key is only known by them, and the length of the key defines the maximum level of security they can achieve. In order to know this shared secret key, Alice and Bob have to use an algorithm based on asymmetric cryptology. In asymmetric cryptology, each individual knows a private secret key, linked with a mathematical relation to a public key known by everyone. Recovering the private key from the public key is considered as impossible, as the number of computations needed is too high. If Alice wants to safely send data to Bob via an insecure channel, she can encrypt the data with Bob's public key, and Bob will be able to decrypt the data with his private key. Because the private key is needed during the decryption, Bob is the only one that can recover Alice's message. Asymmetric cryptography algorithms can serve different purposes, among them the safe share of a common secret key. As I did not study asymmetric cryptography during my internship, we will now suppose that Alice and Bob share a common secret key K.

Now let's suppose that Alice wants to send a message (M) to Bob. Alice transforms M with the help of a encryption algorithm ($\mathcal{E}$) and the key (K) into a ciphertext: $C = \mathcal{E}(K, M)$. Bob then decrypts the ciphertext with the corresponding decryption algorithm $\mathcal{D}$ and the key and obtains the message: $M = \mathcal{D}(K, C)$. For Bob to be able to decrypt the message, $x \to \mathcal{E}(K, x)$ needs to be injective for a fixed K. The encryption and decryption algorithms have the same signature, which is

$$\mathcal{E} : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^*.$$

There are two main families of encryption algorithms : stream ciphers and block ciphers. I have mainly worked on block ciphers during my internship. They are low-level algorithms, called primitives, whose careful analysis ensure the security of the global algorithm. The most famous one is the AES, standardized in 2000 by the NIST and still widely used today in TLS 1.3 for example. There exist no formal proof of the security of primitives, but they are chosen so that known attacks do not threaten them. In addition, cryptanalysis is operated on these primitives to evaluate their security. Block ciphers (E) take a key and a n-bit plaintext as input and return a n-bit ciphertext. Their signature is

$$E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n.$$

Then, to encrypt a full message M, we pad the message so that its length reaches a multiple of n, and divide it into n-bit blocks. The block cipher is inherently deterministic: encrypting the same message under the same key will give the same result. Encrypting all blocks independently from one another and concatenating the result directly leads to collisions if the plaintext blocks are repeated, in which case the attacker knows that input blocks were equal.

To avoid that problem, a mode of operation uses the primitive to safely encrypt data. It usually takes a randomly chosen n-bit input vector (IV) as input. For instance, in the Cipher Block Chaining mode of operation (CBC), the first block is XORed with the IV before being encrypted. The resulting ciphertext block is XOR-ed with the second plaintext block etc...

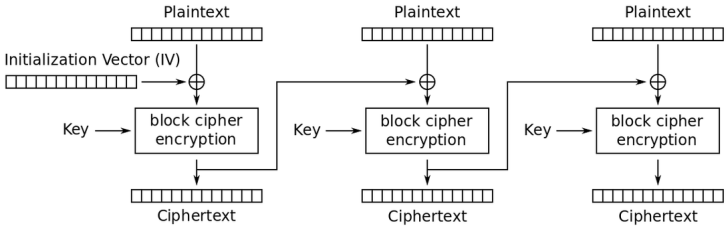


Figure 1: Cipher Block Chaining (CBC) mode of operation.

A collision is still possible theoretically, but it only happens after $2^{n/2}$ blocks, even if the plaintext is controlled by the attacker. The common modes of operations (CBC, CTR, GCM,...) have a proof of security up to $2^{n/2}$ blocks of data under the hypothesis that the primitive is secure. However these modes of operation are often difficult to design and to prove secure.

Liskov et al. introduced in 2002 the notion of Tweakable block ciphers [LRW11]. Like block ciphers, these primitives take a key and a n-bit plaintext input, but they take an additional t-bit tweak as input. The combination of the tweak and the key material is called tweakey. Their signature is

$$\widetilde{E} : \{0,1\}^k \times \{0,1\}^t \times \{0,1\}^n \to \{0,1\}^n.$$

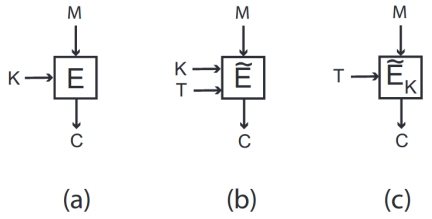


Figure 2: (a) *Standard block cipher* encrypts a message M under control of a key K to yield a ciphertext C. (b) *Tweakable block cipher* encrypts a message M under control of not only a key K but also a tweak T to yield a ciphertext C. The tweak can be changed quickly, and often is public. (c) Another way of representing a tweakable block cipher, here the key K shown inside the box.

This aims at simplifying the proofs and the designs of modes of operation, without loosing known results about primitives, as the extra-cost of making a block cipher "tweakable" is small. New modes of operation emerge from tweakable block ciphers : changing the tweak between different blocks of the same message ensures that collisions give no information to the attacker. Indeed $\widetilde{E}(K,T,M) = \widetilde{E}(K,T',M')$ if $T \neq T'$ gives no relation between M and M' for a secure tweakable primitive $\widetilde{E}$. This gives a security of up to 2^n blocks of data.

Designing a secure tweakable block cipher primitive is harder than a normal block cipher one, because it gives the attacker an additional degree of freedom, which is the choice of the tweak. Indeed we want the result of two encryptions under different tweaks to be completely independant.

The forkcipher framework was designed in 2018 by Andreeva et al. [ARVV18] for very short messages. It aims at producing authentificated encryption (AE) primitives. In addition to the privacy offered by an encryption algorithm, an AE ensures the integrity of the data. Usually, this is done by combining an encryption algorithm and a message authentification code (MAC). However, for very short messages, the size of the MAC is approximately the size of the encrypted message. Therefore having seperate algorithms to compute the MAC and the ciphertext is expensive. The clever idea behind the forkcipher framework is to be more efficient than a standard AE algorithm because of common computations of both ciphertexts and parallelization. A forkcipher outputs two ciphertexts, for redundancy, both leading to the plaintext. Once both ciphertexts were obtained, the reciever of the data can check if both ciphertexts lead to the same plaintext, and when appropriate, he knows that the data comes from a sender knowing the key. Forkciphers are built upon secure block ciphers, and their specifications depend on the block cipher used. We will only work on forkciphers based on tweakable block ciphers.

A forkcipher takes a n-bit plaintext, a k-bit key and a t-bit tweak input and returns two n-bit ciphertexts C_0 and C_1 derived from the same plaintext. The signature of this primitive is the following:

$$\widetilde{F} : \{0,1\}^k \times \{0,1\}^t \times \{0,1\}^n \to \{0,1\}^n \times \{0,1\}^n.$$

Block ciphers consist generally of a round function applied a specific number of times r_{tot} to the plaintext. A forkcipher applies this round function r_i times to the plaintext, then forks the state, and compute independently both ciphertexts, applying r_0 rounds with the tweakey material TK_0 (derived from the initial tweakey) in the first branch and r_1 rounds with the tweakey TK_1 (also derived from the initial tweakey) in the second branch. In most cases, $r_0 = r_1$ and $r_i + r_0 = r_{tot}$. Figure 3 shows how a forkcipher operates.

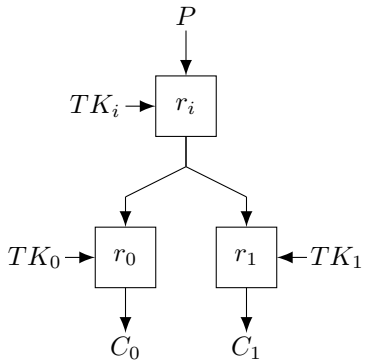


Figure 3: Illustration of an encryption by a forkcipher. Each box corresponds to several rounds of a block cipher round function.

1.4 Introduction to Cryptanalysis

The word cryptanalysis refers to the study of a primitive from the attacker's perspective. As seen in Section 1.3, the security of the mode of operation is based on the security of the primitive, therefore careful analysis of the primitive is required. We assume that the attacker can choose part of the plaintext or ciphertext, and model this as oracles. For instance, the encryption oracle allows the attacker to query the ciphertext corresponding to a specific chosen plaintext, encrypted under the key K. Different types of oracles exist when cryptanalysing a standard block cipher, but the usual ones are the encryption and the decryption oracles. For tweakable block ciphers, an encryption oracle can let the attacker choose the tweak in addition to the plaintext.

In symmetric cryptology, block ciphers frequently consist of a round function, applied a certain number of times to the plaintext. A natural way to analyse block ciphers is to find attacks on the round reduced primitive, with the highest number of rounds as possible. If the number of rounds attacked is close to the total number of rounds of the full primitive, the block cipher is considered as unsecure. As we can add an undeterminate number of rounds to a block cipher, there is naturally a compromise between security and performance. Consequently, good primitives are not only secure but also very efficient.

We intuitively describe the **level of security** offered by a cryptographic primitive as the logarithm of the number of encryptions needed to recover the key K. The level of security is at most k for a k-bit key, as the attacker can ask the oracle for one couple Plaintext/Ciphertext (M,C), then iterate over all 2^k keys and compute the ciphertext for the corresponding message M under each key. After 2^{k-1} encryptions in average, the attacker has a matching ciphertext, then tests the candidate key with another couple Plaintext/Ciphertext to confirm his guess (it is possible that two different keys have the same matching $M \rightarrow C$ for one couple, but very unlikely for two chosen plaintexts). The attacker can be sure to recover the key after 2^k encryptions, which induces a maximum level of security of k. This is called the brute-force attack. Typical values of k are 80, 128,196 or 256. However some attacks target weak keys, and therefore have a marginal probability of success. If ϵ is the probability of success of an attack, and T is its complexity, we can extend the definition of **level of security** (l) of an algorithm as:

$$l = log_2(T/\epsilon) \tag{1}$$

The complexity of an attack is divided into three different complexities: time, memory and data. The first two correspond to standard complexities used in computer science algorithms, that is respectively the number of computations and the maximum memory storage needed for the attack. The data complexity is the number of call to the encryption oracle of the attack.

We will show in our report that despite the designers claim of security, adding some features to existing encryption algorithms often lowers the security in a unexpected manner.

2
PRELIMINARIES

During my internship, I focused on KIASU-BC, a tweakable block cipher based on AES, and ForkAES, the forkcipher based on KIASU-BC. We'll therefore start with a presentation of AES block cipher, along with known attacks against it, then we will introduce KIASU-BC and ForkAES. We will then show the notation used for our paper.

2.1 Presentation of AES

AES is a block cipher designed by J. Daemen and V. Rijmen in 1998 [DR01], selected in 2000 after the NIST competition. It was standardized in 2000, and is widely believed to be a secure block cipher. Three versions of the algorithm were standardized although the designers proposed more versions in their original paper. The three versions offer different key size : 128, 196 and 256, with 128-bit blocks. We will only work on AES-128 (128-bit keys). AES is based on a design principle known as a substitution-permutation network, and is efficient in both software and hardware. AES-128 takes a 128-bit plaintext and a 128-bit key input and returns a 128-bit ciphertext. First, the 128-bit message is placed into a 4x4-byte array, called the state. For example, if $M = b_0 b_1 ... b_{15}$, the initial state will be

b_0	b_4	b_8	b_{12}
b_1	b_5	b_9	b_{13}
b_2	b_6	b_{10}	b_{14}
b_3	b_7	b_{11}	b_{15}

.

The encryption consists of 10 rounds, each of which is a combination of 4 operations: SubBytes (SB), ShiftRows (SR), MixColumns (MC) and AddKey (AK). The final round omits the MixColumns operation, and an initial round key is XORed to the state before the first round.

- The SubByte operation performs a permutation on each byte of the state.
- The ShiftRow operation shifts the second line of the state by 1 cell on the left, the third line by 2 cells on the left, and the last line by 3 cells on the left.
- The MixColumn operation multiply each column of the state by a matrix $\mathcal{M}$ in the field $\mathbb{F}_{2^8} = \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1)$, defined by :

$$\mathcal{M} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}.$$

- The AddKey operation XORs the state with the round key. Round keys are different from round to round, computed from the initial key with a key schedule.

After applying 10 rounds to the plaintext, the state is returned. Because every step is invertible, decrypting the message consists of applying 10 inverse rounds to the matrix and returning the state, which is the plaintext. AES resists differential attacks because of its very strong diffusion. This means that a single byte difference in the plaintext propagates very quickly to the entire state, and therefore leads to a completely different and unpredictable ciphertext. Indeed, the MixColumn operation spreads a single byte difference to an entire column difference, and the ShiftRow operation ensure that every column is affected. See the following scheme, where the grey cells correspond to non-zero differences and the white cells to zero difference:

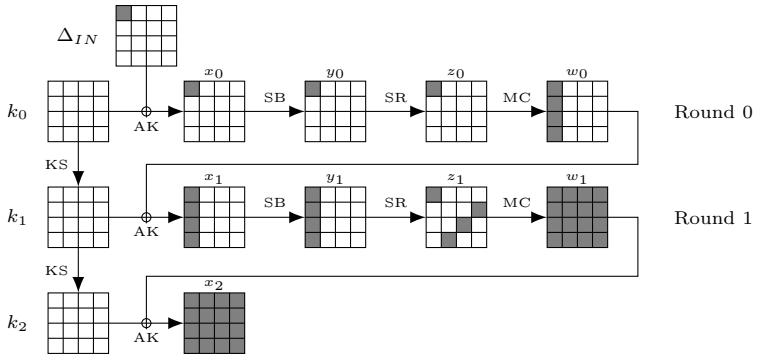


Figure 4: Propagation of an input difference on a single byte, through 2 rounds of AES.

There is no attack on 10 rounds of AES-128 to this day. However cryptanalysts deeply studied reduced-round AES. Though it does not represent the real AES-128, it allows the cryptography community to think about ways to attack the full primitive, if there exists any. The most successful attacks are on 7 rounds. They are showed in Table 1, along with their complexities.

2.2 Description of KIASU-BC and ForkAES

KIASU-BC is a tweakable variant of AES-128 designed by J.Jean et al. [JNP14a]. It has the same number of round as AES-128 and the round functions are very similar, but has an additional 64-bit input. The only change between the two primitives is the AddTweak (AT) operation after the AddKey operation of each round, in which the 64-bit tweak is XORed to the first two rows of the state. Note that unlike the key of AES-128, the tweak does not go through a tweak schedule, and is the same on each round.

The tweak can be chosen by the attacker, therefore an attacker can offset a state difference with a tweak difference for example. Despite the designers initial claim, for most of the existing attacks on AES-128 there exists an equivalent attack on KIASU-BC reaching one more round. Table 1 sums up the attacks and their complexities

Algorithm	Attack Type	Rds.	Data	Time	Memory	Reference
AES-128	Impossible Diff.	7	$2^{106.2}$	$2^{110.2}$	$2^{90.2}$	[MDRMH10]
AES-128	Meet in the Middle	7	2^{97}	2^{99}	2^{98}	[DFJ13]
AES-128	Square	6	2^{32}	2^{71}	2^{32}	[DKR97]
KIASU-BC	Impossible Diff.	8	2^{118}	$2^{120.2}$	2^{102}	[DL17]
KIASU-BC	Boomerang	8	2^{103}	2^{103}	2^{60}	[DL17]
KIASU-BC	Meet in the Middle	8	2^{116}	2^{116}	2^{86}	[MAY16]
KIASU-BC	Square	7	$2^{48.5}$	$2^{43.6}$	$2^{41.7}$	[DEM16]

Table 1: Comparison of existing attacks on AES-128 and KIASU-BC.

ForkAES is an authentificated encryption algorithm, based on the forkcipher framework designed by Andreeva et al. [ARVV18] for very short messages. It takes a 128-bit plaintext, a 128-bit key and a 64-bit tweak input and returns two 128-bit ciphertexts C_0 and C_1 derived from the same plaintext. The encryption of the data works almost exactly like KIASU-BC. To compute C_0 and C_1, five Kiasu-BC rounds are applied to the plaintext. Then we duplicate the state and compute 5 more Kiasu-BC rounds with different round keys, ob-

tained with an extension of the Kiasu-BC tweakey schedule. This way, we efficiently obtain two ciphertexts that both went through 10 rounds of KIASU-BC, for a total cost of 15 KIASU-BC round encryptions.

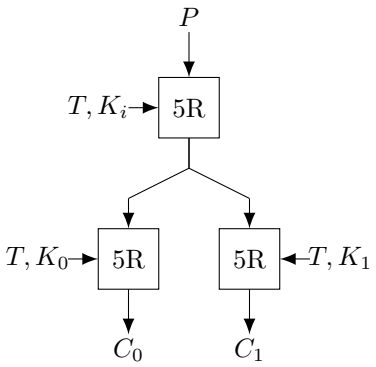


Figure 5: Illustration of an encryption by ForkAES. Each box corresponds to 5 Rounds of Kiasu-BC, with different round keys.

Although the designers of ForkAES state "Since we do not introduce any novel design complexities, the security of our forkcipher design can be reduced to the security of the AES and KIASU ciphers for further type of attacks", this type of encryption provides the attacker with a new oracle: the reconstruction oracle. In this model, the attacker can ask for the second ciphertext C_1 obtained from a chosen ciphertext C_0. Unlike KIASU-BC, the path from C_0 to C_1 consists of 5 decryption rounds followed by 5 encryption rounds, which lowers the diffusion around the middle round. We denote ForkAES-r_{init}-r_0-r_1 the round reduced ForkAES version composed of r_{init} rounds before the forking point, and respectively r_0 and r_1 rounds in each banch. Full ForkAES is therefore ForkAES-5-5-5. If r_{init} is not used and can take any value, we denote the round reduced version ForkAES-$*$-r_0-r_1. Banik et al. applied several known attacks on ForkAES [BBJ+19] and found interesting results, such as very low complexity differential attacks on ForkAES-$*$-4-4. They exploited the lack of diffusion, which inspired us for our improved differential attack. Our attack does directly endangers the full ForkAES level of security, but the complexity of the attack is high enough not to be fatal for the primitive. The different complexities of key recovery attacks for 128-bits key are lower than $2^{128} \times \epsilon$ where ϵ is the probability of success. These are the first attacks on full ForkAES.

ForkAES Version	Attack Type	Data	Time	Memory	Pr. of Success	Reference
ForkAES-$*$-4-4	Impossible Diff.	$2^{39.5}$	2^{47}	2^{35}	1	[BBJ+19]
ForkAES-$*$-4-4	Reflection Diff.	2^{35}	2^{35}	2^{33}	1	[BBJ+19]
ForkAES-$*$-5-5	Truncated Diff.	2^{73}	2^{73}	2^{58}	2^{-32}	Sect. 4
ForkAES-$*$-5-5	Truncated Diff.	$2^{97.6}$	$2^{117.6}$	2^{85}	$2^{-5.4}$	Sect. 5
ForkAES-$*$-5-5	Truncated Diff.	$2^{104.6}$	$2^{123.6}$	2^{96}	0.38	Sect. 6

Table 2: Comparison of complexities and probabilities of success of attacks on ForkAES versions using reconstruction queries.

2.3 Notations and Mathematics

Notations We denote x_i,y_i,z_i and w_i the states after respectively the AddTweaKey, SubByte, ShiftRows and MixColumn operations of round i. For a 128-bit state s, we denote $s[j]$ the j-th byte of the state, following the byte ordering given by:

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

The tweak byte ordering will follow the key byte ordering, as we do not use other tweak bytes than byte 0.

ForkAES forks the state after 5 rounds and transforms it through two different branches into two ciphertexts. The first branch takes round keys k_5 to k_{10} and the second branch takes round keys k_{11} to k_{16}, derived from k with an extended key schedule. We denote the ciphertext obtained from the first branch C_0 and the ciphertext obtained from the second branch C_1.

For the following attacks, we denote $\widehat{C_0}$ the state after partial inversion of the tenth round (we inverted AddTweak, MixColumn and ShiftRows), and $\widehat{k_{10}}$ the equivalent key of the real subround key k_{10}. The equivalent operation is denoted AK_{10}^{eq}. In other words,

$$\widehat{C_0} = SR^{-1}(MC^{-1}(AT(C_0)))$$
$$\widehat{k_{10}} = SR^{-1}(MC^{-1}(k_{10})).$$

Similarly, we denote $\widehat{C_1}$ the equivalent output ciphertext and $\widehat{k_{16}}$ the equivalent ultimate round key. The equivalent operation is denoted AK_{16}^{eq}. They verify :

$$\widehat{C_1} = SR^{-1}(MC^{-1}(AT(C_1)))$$
$$\widehat{k_{16}} = SR^{-1}(MC^{-1}(k_{16})).$$

We denote by $(\widehat{C_0}, T)$ the ciphertext $\widehat{C_0}$ with the tweak T. We denote ATK_i the AddTweaKey operation, which gathers the AddKey and the AddTweak operations of round i. We denote the associated tweakey $tk_i = T + k_i$ where T is the tweak and k_i the round key of round i. We denote C_x a column such that it contains 0 on the three last rows, and a certain value x in the first row.

Mathematics Every byte of the states we consider are elements from the field

$$\mathbb{F}_{2^8} = \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1).$$

Additions, multiplications and divisions of bytes should always be interpreted as operations in the field $\mathbb{F}_{2^8}$. Explicit elements of the field are represented as integers. For instance 2 represents the polynom X of the field.

We denote by $\mathcal{P}(\delta_i, \delta_o)$ the probability of having an output difference through the AES S-Box of δ_o if the input difference is δ_i. If SB denotes the AES SubByte permutation, we have:

$$\mathcal{P}(\delta_i, \delta_o) = |\{x \in \mathbb{F}_{2^8}, SB(x) + SB(x + \delta_i) = \delta_o\}| \times 2^{-8}.$$

Note that an important cryptographic property of SB is that for $\delta_i \neq 0$, $\mathcal{P}(\delta_i, \delta_o)$ is either $2^{-7}, 2^{-6}$ or 0, and for every non zero δ_i, there exists a unique $\delta_o \in \mathbb{F}_{2^8}$ such that $\mathcal{P}(\delta_i, \delta_o) = 2^{-6}$. SB^{-1} also has this property.

In the following proof of the attack, Θ will denote a tweak difference such that Θ is only active on the first byte and $\mathcal{P}(\Theta[0], \Theta[0]/2) = 2^{-6}$. Three such differences exist.

The three possibles values for $\Theta[0]$ are 33, 127 and 227.

We will denote $\Pr(s \to s')$ the probability of going from state s to state s' of the characteristic.

For the rest of the proof a_0, a_1, a_2, a_3 will be the 4 distinct values such that $a_0 + a_1 = a_2 + a_3 = \Theta[0]/2$ and $SB^{-1}(a_0) + SB^{-1}(a_1) = SB^{-1}(a_2) + SB^{-1}(a_3) = \Theta[0]$, and such that $a_0 < a_1$, $a_2 < a_3$ and $a_0 < a_2$ where the comparisons are operated on integer representations of the bytes. This way, the four values are uniquely defined. Let us denote $\delta_x = a_0 + a_2$. Furthermore, we denote $b_i = SB^{-1}(a_i)$ for $i \in \{0, 1, 2, 3\}$.

We denote τ the tweak active only on the first byte, such that $\tau[0] = \Theta[0]/28$.

For example, for $\Theta[0] = 33$, we have $\Theta[0]/2 = 158$, $a_0 = 99, a_1 = 253, a_2 = 104, a_3 = 246$, $\delta_x = 11$.

3 EXISTING ATTACK AGAINST FORKAES-∗-4-4

In this section, we describe the existing reflection differential attack presented by Banik et al. [BBJ+19]. This attack on round reduced ForkAES inspired us for our attacks presented in Section 4, 5 and 6. This attack is a truncated differential.

3.1 Presentation of Truncated Differential Characteristics

A differential characteristic is a specific trail of differences from an input difference to an output difference, going through each round of the block cipher. We choose a pair of input plaintexts such that the plaintext difference equals the input difference of the characteristic. The pair has a certain probability of satisfying the characteristic, say $\epsilon > 2^{-k}$, where k is the size of the key. We then ask for their encryption. If the encrypted pair have the expected difference, we save the pair. If we repeat this operation $1/\epsilon$ times, we have in average one pair that satisfies the characteristic. Then, we need a way to deduce some key bytes from a pair satisfying the characteristic. Once we have reduced the key space, we can proceed to an exhaustive search and test all possible keys over the reduced key space. That operation needs less computations than the exhaustive search on the entire key space, so we can recover the key more efficiently than the exhaustive search.

Differential characteristics are interesting in AES because most of the operations are linear and differences in output of linear operations can be computed from differences in input with probability 1. The SubByte operation is the only one that is not linear, and the uncertainty of the differential characteristic comes from this operation.

Truncated differential characteristics regroup several differential characteristics. Usually, for AES and its variants, we do not specify the value of the difference, but we only distinguish inactive (zero difference) bytes and active (non-zero difference) bytes. Not only this significantly increases the probability of the characteristic, but this allows us to build some plaintext sets, in which most of the pairs of plaintexts have the truncated input difference of the characteristic. For a set of n plaintexts, we create $\binom{n}{2}$ pairs. Therefore, we need a lower number of encrypted data to find one pair that satisfies the characteristic.

3.2 Differential Trail and Probability

Because we consider ForkAES-∗-4-4, round keys of the first branch are k_5,k_6,k_7,k_8 and k_9, and round keys of the second branch are k_{10},k_{11},k_{12},k_{13} and k_{14}. We define $\widehat{k_9}$ and $\widehat{k_{14}}$ similarely to Section 2.3:

$$\widehat{k_9} = SR^{-1}(MC^{-1}(k_9))$$
$$\widehat{k_{14}} = SR^{-1}(MC^{-1}(k_{14})).$$

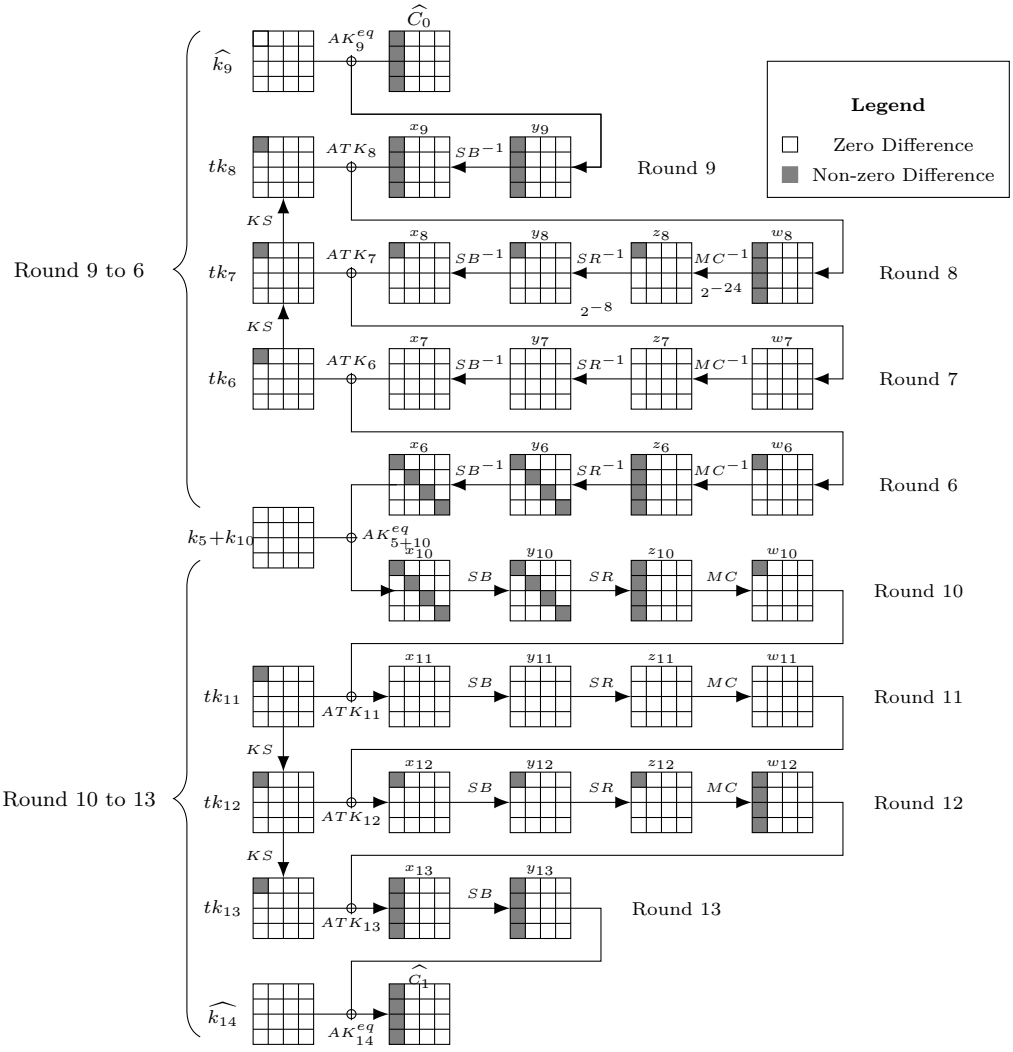


Figure 6: Truncated differential characteristic scheme for ForkAES-∗-4-4.

The input difference of the characteristic is only active on bytes 0,1,2,3, and the difference in tweaks is active only on the first byte.

- $\Pr(w_8 \to z_8) = 2^{-24}$ since three bytes become inactive after the inverted MixColumn operation.
- $\Pr(x_8 \to w_7) = 2^{-8}$ since the byte difference in position 0 cancels out with the tweak difference.
- $\Pr(z_{10} \to w_{10}) = 2^{-24}$ since three bytes become inactive after the MixColumn operation.
- $\Pr(w_{10} \to x_{11}) = 2^{-8}$ since the byte difference in position 0 cancels out with the tweak difference.

In total, the probability of the characteristic is 2^{-64}.

3.3 Attack Procedure

Unlike the attack described in "Cryptanalysis of ForkAES" [BBJ+19], we will not consider the tweak difference as fixed. Fixing the tweak, as we will show in Section 5.1, does not ensure the above probability of going through the characteristic, as the middle rounds can only be satisfied if the tweak and the key are compatible.

Construction of the pairs: First, we choose arbitrary values for the last three columns of the plaintext, along with arbritrary values for all bytes of the tweak except the first one. Iterating over the first byte of the tweak and the values of the first column of the plaintext gives us a set of 2^{40} tweak/plaintexts. Among them, we query $2^{32.5}$ randomly chosen plaintexts. This gives us $2^{32.5} \times (2^{32.5} - 1)/2 = 2^{64}$ pairs of input verifying the characteristic input difference. In average, one pair satisfies the characteristic.

Filtering the pairs: We can distinguish the right pair from the wrong ones by storing the corresponding ciphertexts into a hashtable, indexed by the values of the three last columns. If a pair of plaintext satisfies the characteristic, there is a collision in the hashtable between them. For random pairs of ciphertexts, there is a collision with probability 2^{-96}. As we have 2^{64} pairs stored in the hashtable, a random collision occurs with probability 2^{-32}, which can be neglected. The only collision is therefore the right pair.

Reducing the key space: The pair went through the characteristic so the difference in state x_{12} is exactly the tweak difference. Because of a property of S-boxes in $\mathbb{F}_{2^8}$, there are only 2^7 (and not 2^8) possible differences of the output values through SB if the input values have a fixed difference. We can compute these differences from the known tweak difference. We can propagate these differences through SR, MC and ATK_{13} since they are linear operations, in order to get 2^7 candidates for x_{13}. We also know the difference in $\widehat{C_1}$, therefore the difference in y_{13}. The following property of S-boxes in $\mathbb{F}_{2^8}$ reduces the key space: for random non-zero δ_i and δ_o, there is in average one solution $x \in \mathbb{F}_{2^8}$ to the equation

$$SB(x) + SB(x + \delta_i) = \delta_o.$$

Thus, for each 2^7 column difference in x_{13}, there is in average one column value that verifies the difference conditions around the S-box. Knowing the first column value in y_{13} and in $\widehat{C_1}$, we deduce in average one value for the first column of $\widehat{k_{14}}$. In total, we end up with 2^7 possible values for the first column of $\widehat{k_{14}}$.

End of the attack No operation depends specifically on the column position in AES, KIASU-BC and ForkAES. Consequently, we can iterate this attack by shifting the entire characteristic to another column. We can therefore recover 2^7 candidates for each column of $\widehat{k_{14}}$. We can invert the key schedule to recover the master key from $\widehat{k_{14}}$ The 2^{28} candidates are ultimately tested exhaustively.

3.4 Complexity Evaluation

We proceed to $2^{32.5}$ data queries per column, which induces a data complexity of $2^{34.5}$. The memory complexity is $2^{32.5}$ AES states to store $2^{32.5}$ values of $\widehat{C_0}$ in the hashtable. The time complexity is 2^{28} encryptions for the final exhaustive search and $2^{34.5}$ memory accesses for the data processing. Eventually the (Data,Time,Memory) complexity will be:

$$(D, T, M) = (2^{34.5}, 2^{34.5}, 2^{32.5}).$$

4
ATTACK AGAINST FULL FORKAES FOR 2^{96} WEAK KEYS

4.1 Core of the Attack

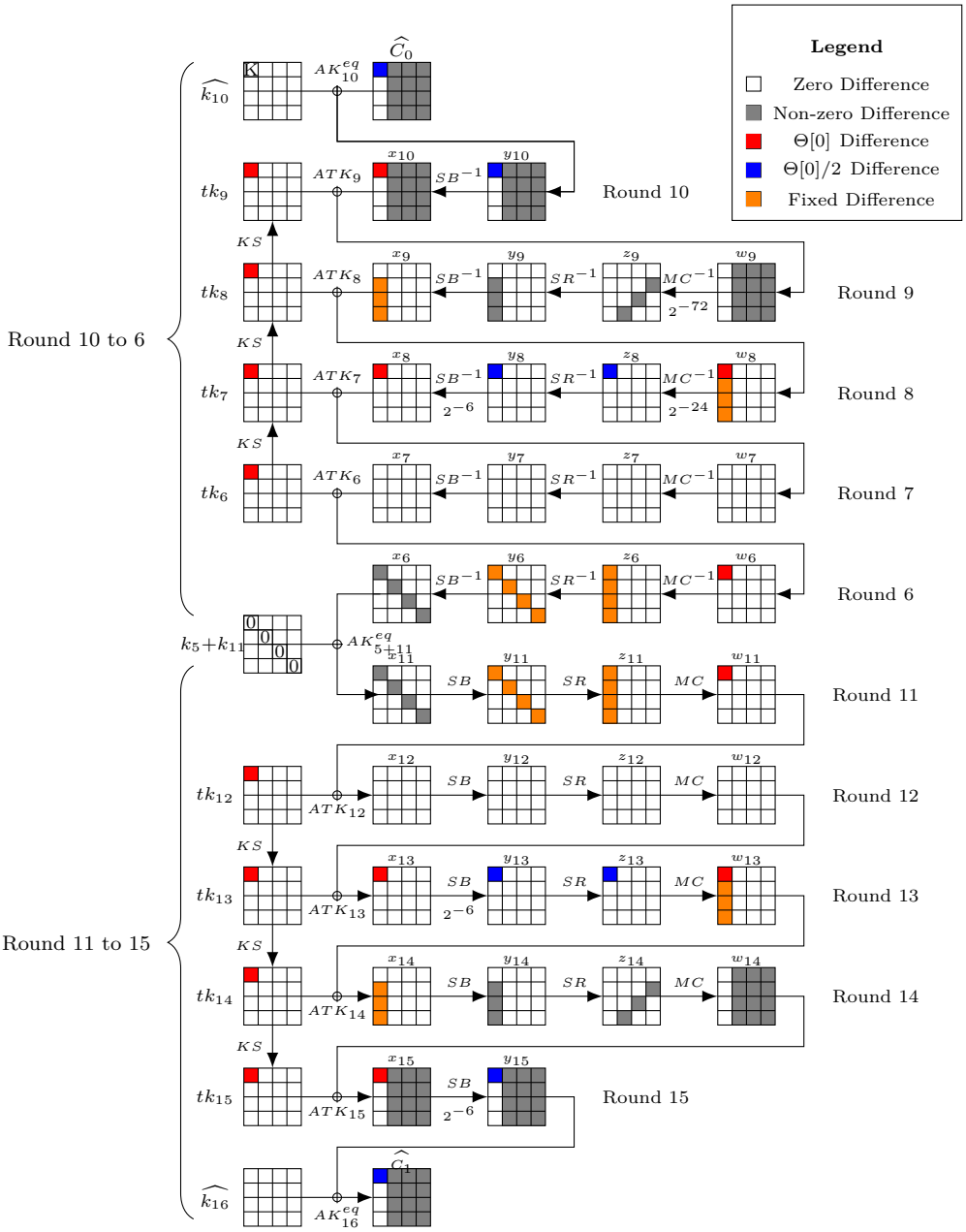


Figure 7: Differential characteristic for very weak keys.

This attack targets weak keys such that $k_5 + k_{11}$ only has zero values on the diagonal. This happens with probability 2^{-32}. We will show an attack on such keys with complexity under $2^{96} = 2^{128} \times 2^{-32}$.

This attack uses a differential characteristic with high probability. The particularity of this characteristic is that if one pair satisfies this characteristic, then it is easy to construct another pair that has a very high chance of satisfying the characteristic as well.

4.2 Differential Characteristic for the First Pair

First, we guess the first byte of the equivalent key $\widehat{k_{10}}$ and denote it K. The pairs that we are going to focus on are pairs of the form :

$$p = \left(\left(\begin{pmatrix} a_0 + K & u_0 & u_4 & u_8 \\ 0 & u_1 & u_5 & u_9 \\ 0 & u_2 & u_6 & u_{10} \\ 0 & u_3 & u_7 & u_{11} \end{pmatrix}, 0 \right), \left(\begin{pmatrix} a_1 + K & v_0 & v_4 & v_8 \\ 0 & v_1 & v_5 & v_9 \\ 0 & v_2 & v_6 & v_{10} \\ 0 & v_3 & v_7 & v_{11} \end{pmatrix}, \Theta \right) \right)$$

for any 96-bits vectors u and v. We built the pair so that the first byte difference at state x_{10} is exactly the tweak difference, so that the state w_9 has an inactive first column.

- $\Pr(w_9 \to z_9) = 2^{-72}$ since 9 bytes become inactive after the MixColumn operation.
- $\Pr(w_8 \to z_8) = 2^{-24}$ since 3 bytes become inactive after the MixColumn operation.
- $\Pr(y_8 \to x_8) = \mathcal{P}(\Theta[0], \Theta[0]/2) = 2^{-6}$ by definition of the tweak.
- Round 7 is then inactive.
- Values in the diagonals of both elements of the pair in state x_6 do not change with the AK^{eq}_{5+11} operation, because of the hypothesis we made on the key. Consequently, both elements of the pair have the same diagonal values in state y_6 and in state y_{11}. The difference in state w_{11} is therefore the tweak difference with probability 1.
- Round 12 is then inactive.
- $\Pr(x_{13} \to y_{13}) = 2^{-6}$ by definition of the tweak.
- $\Pr(x_{15} \to y_{15}) = 2^{-6}$ because we filter the pair on the first byte difference and we want a $\Theta[0]/2$ difference.

In summary, for the first pair, $\Pr(\widehat{C_0} \to \widehat{C_1}) = 2^{-72-24-3\times 6} = 2^{-114}$

4.3 Construction of the Twin Pair

The twin pair p_{twin} associated with the pair p will be constructed so that the first three rounds follows the characteristic with probability 1 knowing that p follows it. The probability of the total characteristic for p_{twin} will therefore be 2^{-12}.

$$p_{twin} = \left(\left(\begin{pmatrix} SB(b_0 + \tau[0]) + K & u_0 & u_4 & u_8 \\ 0 & u_1 & u_5 & u_9 \\ 0 & u_2 & u_6 & u_{10} \\ 0 & u_3 & u_7 & u_{11} \end{pmatrix}, \tau \right), \left(\begin{pmatrix} SB(b_1 + \tau[0]) + K & v_0 & v_4 & v_8 \\ 0 & v_1 & v_5 & v_9 \\ 0 & v_2 & v_6 & v_{10} \\ 0 & v_3 & v_7 & v_{11} \end{pmatrix}, \tau + \Theta \right) \right).$$

As a twin pair p_{twin} associated with a pair p is introduced, $s, s', \overline{s}, \overline{s'}$ will respectively denote the values of state s for the first element of p, the second element of p, the first element of p_{twin} and the second element of p_{twin}.

We remind the reader that τ is a tweak active only on the first byte such that $\tau[0] = \Theta[0]/28$. p_{twin} is constructed such that

$$\overline{w_9}[0] = ATK_9(SB^{-1}(AK_{10}^{eq}(SB(a_0 + X) + K))) = a_0 + k_9[0] = w_9[0]$$
$$\overline{w'_9}[0] = ATK_9(SB^{-1}(AK_{10}^{eq}(SB(a_1 + X) + K))) = a_1 + k_9[0] = w'_9[0].$$

Up to this point, neither MixColumn nor ShiftRows have been applied on the states, therefore each byte of the state has a specific value depending only on the corresponding byte of the input. We chose p_{twin} such that all bytes but the first one have exactly the same input value as the first pair, so we know that $\overline{w_9}[1, ..., 15] = w_9[1, ..., 15]$ and $\overline{w'_9}[1, ..., 15] = w'_9[1, ..., 15]$, which implies

$$\overline{w_9} = w_9$$
$$\overline{w'_9} = w'_9.$$

This equality of states is satisfied for the whole round 9, up to x_9. After the next tweak addition, we have the following property :

$$\overline{w_8} = w_8 + \tau[0]$$
$$\overline{w'_8} = w'_8 + \tau[0].$$

The values of state w_8 of both elements of p_{twin} only differ from those of p on the first byte. Since p satisfies the characteristic, we have $z_8[0] + z'_8[0] = \Theta[0]/2$ This implies:

$$\overline{z_8}[0] = z_8[0] + MC^{-1}(C_{\tau[0]})[0] = z_8[0] + 14 * \tau[0] = z_8[0] + \Theta[0]/2 = z'_8[0]$$
$$\overline{z'_8}[0] = z'_8[0] + MC^{-1}(C_{\tau[0]})[0] = z'_8[0] + 14 * \tau[0] = z'_8[0] + \Theta[0]/2 = z_8[0].$$

Ultimately, we have

$$\overline{x_8}[0] = x'_8[0] = \Theta[0] + x_8[0] = \Theta[0] + \overline{x'_8}[0]. \tag{2}$$

Furthermore, since $\overline{w_9} = w_9$ and $\overline{w'_9} = w'_9$, the differences of both pairs in state w_9 are equal. Because $w_9 \to z_8$ consists only in linear operations, the differences in state z_8 of both pairs are equal, in particular the difference $\overline{z_8} + \overline{z'_8}$ is only active on the first byte.

The twin pair has therefore a difference in state x_8 active only on the first byte, and the first byte difference is exactly the tweak difference $\Theta[0]$ as we showed in Equation (2). So the twin pair has an inactive round 7, and satisfies the first three rounds of the characteristics with probability 1.

4.4 Filtering Twin Pairs

The differential characteristic for twin pairs has a total probability of 2^{-126}. Now, we want to distinguish the pairs that satisfy the characteristic from other pairs. First, both pairs must have inactive bytes 1,2 and 3 in the final state, and the first byte difference must be exactly $\Theta[0]/2$. These conditions are verified with probability 2^{-64} on random pairs of pairs. Furthermore, we will add another condition to the filter : for both pairs, we can deduce the possible values of $\widehat{k_{16}}[0]$, and they should have a common value for $\widehat{k_{16}}[0]$. The differential condition between x_{15} and y_{15} has 4 possible output values, and these values are a_0, a_1, a_2 and a_3 as seen in section 2.3. If both pairs went through the characteristic, the ouput of every element of each pair will be one of the 4 possible

values $a_i + AK_{16}^{eq}[0]$ for $i \in \{0,1,2,3\}$. The set $\{a_0 + AK_{16}^{eq}[0], a_1 + AK_{16}^{eq}[0], a_2 + AK_{16}^{eq}[0], a_3 + AK_{16}^{eq}[0]\}$ is furthermore uniquely determined by the polynomial defined on $\mathbb{F}_{2^8}$

$$Q(X) = X \times (X + \Theta[0]/2)(X + \delta_x)(X + \Theta[0]/2 + \delta_x) \tag{3}$$

applied on any element $a_i + AK_{16}^{eq}[0]$ of the set. Indeed $\forall i \in \{0,1,2,3\}$,

$$Q(a_i + AK_{16}^{eq}[0]) = (a_0 + AK_{16}^{eq}[0])(a_1 + AK_{16}^{eq}[0])(a_2 + AK_{16}^{eq}[0])(a_3 + AK_{16}^{eq}[0]).$$

No other element that one of the $a_i + AK_{eq}[0]$ have such a value through Q, because a non constant polynomial of degree 4 does not have more than 4 roots. By checking if those polynomial values are the same for values of the first element of each pair, we can check whereas $AK_{eq}[0]$ can possibly be the same for both pairs, and a random pair of pairs passes this filter with probability 2^{-6}.

Note that after simple computations,

$$Q[X] = X^4 + ((\Theta[0]/2)^2 + \delta_x^2 + \delta_x \times \Theta[0]/2) \times X^2 + \delta_x \times \Theta[0]/2 \times (\delta_x + \Theta[0]/2) \times X.$$

Now, because polynomials X^4 and X^2 are linear on $\mathbb{F}_{2^8}$, $Q(X)$ is linear and can be computed efficiently with a 64 bit matrix.

Random twin pairs pass the complete filter with probability 2^{-70}.

4.5 Efficient Implementation of the Filter

We will dress two hashtables and find collision between the two tables to find twin pairs that satisfy the above filter.

For every vector u, we first ask for the encryptions of the inputs:

- $\widehat{C_0}(u) = (C_{a_0+K}|u, 0)$ denoted $\widehat{C_1}(u)$,
- $\widehat{C'_0}(u) = (C_{a_1+K}|u, \Theta)$ denoted $\widehat{C'_1}(u)$,
- $\overline{\widehat{C_0}(u)} = (C_{SB(b_0+\tau[0])+K}|u, \tau)$ denoted $\overline{\widehat{C_1}(u)}$,
- $\overline{\widehat{C'_0}(u)} = (C_{SB(b_1+\tau[0])+K}|u, \Theta + \tau)$ denoted $\overline{\widehat{C'_1}(u)}$.

First, we compare the values of $Q(\widehat{C_1}(u)[0])$ and $Q(\overline{\widehat{C_1}(u)}[0])$, where Q is the previously defined polynomial. We add the following hash to the first hashtable:

$$H_0(u) = \widehat{C_1}(u)[0,1,2,3] \| \overline{\widehat{C_1}(u)}[0,1,2,3]. \tag{4}$$

We then compare the values of $Q(\widehat{C'_1}(u)[0])$ and $Q(\overline{\widehat{C'_1}(u)}[0])$. If the values are equal, we have the same property as above, and we add the following hash to the second HashTable :

$$H_1(u) = (\widehat{C'_1}(u)[0] + \Theta[0]/2) \| \widehat{C'_1}(u)[1,2,3] \| (\overline{\widehat{C'_1}(u)}[0] + \Theta[0]/2) \| \overline{\widehat{C'_1}(u)}[1,2,3]. \tag{5}$$

There is, for every 96-bit vector u that we take, a 2^{-6} chance that we write it inside the first table, and the same probability for the second table, which produces a 12-bits filter on the total number of pairs.

Let's suppose we have a match between an element of the first table and an element of the second one, corresponding to vectors u and v. Our twin pairs will be

$$\begin{aligned} p &= [\widehat{C_0}(u), \widehat{C'_0}(v)] \\ p_{\text{twin}} &= [\overline{\widehat{C_0}(u)}, \overline{\widehat{C'_0}(v)}]. \end{aligned} \tag{6}$$

Having $H_0(u) = H_1(v)$ implies that the difference in bytes 1,2,3 of both pairs are inactive. Furthermore, we have the right difference of $\Theta[0]/2$ in the first byte of each pair. Colliding pairs satisfy the 70-bit filter.

Both hashtables have 64-bit hash keys. However, they only filter 58 bits of data due to redundancy, in addition to the 12 bits filter corresponding to values excluded from the tables. Indeed, knowing that

$$\widehat{C_1}(u)[0] = \widehat{C_1'}(v)[0] + \Theta[0]/2$$
$$Q(\widehat{C_1}(u)[0]) = Q(\overline{\widehat{C_1}(u)}[0])$$
$$Q(\widehat{C_1'}(v)[0]) = Q(\overline{\widehat{C_1'}(v)}[0])$$

implies that there exists $\delta \in \{0, \Theta[0]/2, \delta_x, \Theta[0]/2 + \delta_x\}$ such that

$$\overline{\widehat{C_1}(u)}[0] = \overline{\widehat{C_1'}(v)}[0] + \delta.$$

Due to previously checked conditions, $\overline{\widehat{C_1}(u)}[0] = \overline{\widehat{C_1'}(v)}[0]$ therefore has a 2^{-2} probability of happening for a random pair of pairs instead of the naively expected 2^{-8} probability.

We finally have our 70-bit filter implemented with a first 12-bit condition filter and two 64-bit hashtables. The filter has a memory complexity of only $2 \times n \times 2^{-6}$ 96 bits blocks if n is the number of candidate vectors u.

4.6 Processing the Remaining Pairs

The probability that twin pairs satisfy the characteristic is 2^{-126}. Therefore, we need 2^{63} 96-bit vectors to form 2^{126} twin pairs. We expect one right pair. After the 70-bit filter, we will still have to process $2^{126-70} = 2^{56}$ pairs. Further filtering is required in order to distinguish the right pair from the wrong ones. The difference value of z_{13} is $\Theta[0]/2$, and $z_{13} \to x_{14}$ is composed of linear operations, so we can compute the fixed difference values in state x_{14}. Then, we preventively compute the 2^7 possible difference values of a byte difference in state y_{14}, say $y_{14}[1]$. We then compute the possible differences of the last column of the state x_{15}. Knowing the differences in $\widehat{C_1}$, we can deduce 2^7 possible values for the last column of AK_{eq}. Doing that for pairs p and p_{twin}, we can check whether the intersection of these possible values is empty or not. If it is, the pairs are incompatible. This costs 2^8 operations at most, and the probability of having a non-empty intersection is appoximately $2^7 \times 2^7 \times 2^{-32} = 2^{-18}$. When this happens, there is in average one element in the intersection. Doing this with the last column, then the two other columns, we have another filter, which a random pair passes with probability $2^{3\times(-18)} = 2^{-54}$.

After this extended filter, we still have $2^{56-54} = 2^2$ pairs of pairs. For each of them, there are in average one key guess for the three last columns and four key guesses for the first byte. We then iterate over the bytes $AK_{eq}[1, 2, 3]$ to proceed in an exhaustive search for the key. This has a complexity of $2^{2+2+24} = 2^{28}$ which is clearly lower than the complexities of other steps of this attack.

4.7 Complexity of the Attack

First, we guessed the value K of the first byte of the equivalent round key $\widehat{k_{10}}$. Then, for each value, we needed to create 2^{126} pairs and filter them. We therefore needed 2^{63} vectors u, and asked for 2^{65} encryptions, which makes a total data complexity of 2^{73}. The memory complexity is 2^{58}, because the average number of vectors written in each hashtable is 2^{57}. The time complexity is 2^{64} for pair processing, therefore the final time complexity lies in the data processing which gives us a time complexity of 2^{73}.

Our (Data,Time,Memory) complexity will therefore be:

$$(D, T, M) = (2^{73}, 2^{73}, 2^{58}).$$

5
ATTACK AGAINST FULL FORKAES FOR 2^{122} WEAK KEYS

5.1 A Focus on the Middle Rounds

The authors of the paper "Cryptanalysis of ForkAES" [ARVV18] showed a truncated differential attack on ForkAES-∗-4-4, using the same scheme for middle rounds. Their attack used a unique tweak difference. However, this unique tweak difference can possibly never go through the middle rounds:

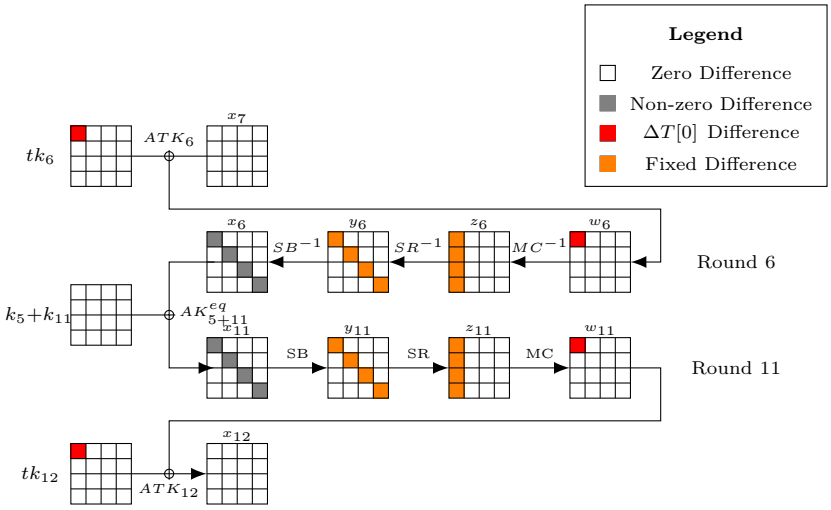


Figure 8: Middle rounds of the characteristic.

Let us note the values of the state s respectively s and s' for the first and the second element of the pair and $k = k_5 + k_{11}$. Let us suppose that we have a tweak difference ΔT between the elements of the pair, only active on the first byte. Additionaly, the state difference in x_7 is inactive. We want the whole state difference to be inactive again in x_{12}. These two conditions imply that the differences in w_6 and w_{11} are exactly the tweak difference ΔT. Therefore, the differences in states z_6 and z_{11} are equal. Hence, y_6 and y_{11} should also have the same difference, which can directly be computed from ΔT. Only diagonal bytes are active, and

$$(y_{11} + y'_{11})[0, 5, 10, 15] = MC^{-1}(C_{\Delta T})[0, 1, 2, 3] = (y_6 + y'_6)[0, 5, 10, 15]. \tag{7}$$

Let us denote $\delta_{\Delta T} \overset{\text{def}}{=} [i](y_{11} + y'_{11})[i] = (y_6 + y'_6)[i] = MC^{-1}(C_{\Delta T})[i/5]$ for $i \in \{0, 5, 10, 15\}$. If the pair goes from differential state y_6 to y_{11}, there are certain values $x_6[i]$, for $i \in \{0, 5, 10, 15\}$ such that:

$$\begin{aligned} SB(x_6[i]) &= y_6[i] & SB(x'_6[i]) &= y'_6[i] \\ SB(k[i] + x_6[i]) &= y_{11}[i] & SB(k[i] + x'_6[i]) &= y'_{11}[i]. \end{aligned}$$

Now, because of equation (7), a simple combination of the above equations gives the following property :

$$\begin{aligned} SB(x_6[i]) + SB(x'_6[i])) &= \delta_{\Delta T}[i] \\ SB(k[i] + x_6[i]) + SB(k[i] + x'_6[i]) &= \delta_{\Delta T}[i]. \end{aligned} \tag{8}$$

Therefore the pairs of byte $p_1 = (x_6[i], x'_6[i])$ and $p_2 = (x_6[i] + k[i], x'_6[i] + k[i])$ have the same difference $x_6[i] + x'_6[i]$ and their output difference through the AES S-box is for both pairs $\delta_{\Delta T}[i]$.

Three possibilities emerge:

- $\mathcal{P}(x_6[i] + x'_6[i], \delta_{\Delta T}[i]) = 0$. In this case, the pair can obviously not satisfy the characteristic.
- $\mathcal{P}(x_6[i] + x'_6[i], \delta_{\Delta T}[i]) = 2^{-7}$. There is a unique pair verifying an input difference of $x_6[i] + x'_6[i]$ and an output difference of $\delta_{\Delta T}[i]$ through the AES S-box. In order to pass the characteristic, p_1 and p_2 should have the same elements (switched or not). Therefore, $x_6[i] = k[i] + x_6[i]$ or $x_6[i] = k[i] + x'_6[i]$ which implies $k[i] = 0$ or $k[i] = x_6[i] + x'_6[i]$.
- $\mathcal{P}(x_6[i] + x'_6[i], \delta_{\Delta T}[i]) = 2^{-6}$. There are two pairs verifying an input difference of $x_6[i] + x'_6[i]$ and an output difference of $\delta_{\Delta T}[i]$ through the AES S-box. Suppose (q_0, q_1) and (q_2, q_3) are such pairs. Let's pose $\delta_p = q_0 + q_2$. Then either $x_6[i] = k[i] + x_6[i]$, $x_6[i] = k[i] + x'_6[i]$, $x_6[i] = k[i] + x_6[i] + \delta_p$, $x_6[i] = k[i] + x'_6[i] + \delta_p$. Therefore $k[i]$ can take 4 different values in order for the pair to possibly pass the characteristic.

For each active value of $\delta_{\Delta T}[i]$, there are exactly 2^7 possible key bytes $k[i]$ such that there exist values $x_6[i]$ and $x'_6[i]$ verifying Equation (8). That implies that if we choose a unique ΔT, then there is a probability of 2^{-4} that the key is compatible with the tweak (2^{-1} for each diagonal byte). In other cases, the probability of passing the total characteristic is 0. Additionally, if diagonal key bytes are compatible with the tweak, $\Pr(x_7 \to x_{12}) >= 2^{-28}$, because at least two values $x_6[i]$ satisfy Equation 8 for $i \in \{0, 5, 10, 15\}$.

In summary, if we take a random tweak difference and a random key, there is an average probability of 2^{-32} of satisfying the characteristic on the middle rounds. However, for a fixed tweak difference and a fixed key, there is a 2^{-4} chance that they are compatible, in which case the probability of going through the middle rounds is 2^{-28}. If they are not compatible, the pair can not satisfy the characteristic.

5.2 Our New Weak Key Hypothesis

Now, we will suppose that the key has at least one diagonal byte equal to zero. This happens with probability 2^{-6}. To avoid irrelevant complications, we will suppose that the first byte of the key is 0. We will also suppose that the tweak difference taken is compatible with the key, as seen in Section 5.1. This compatibility happens with probability 2^{-3} (2^{-1} for each non zero diagonal key byte).

5.3 The New Differential Characteristic

Like for the previous attack, we start by guessing the first byte of $\widehat{k_{10}}$, denoted K. The main change between this characteristic and the previous one is the addition of a 2^{-21} probability of passing the middle rounds (2^{-7} for each of the three rightmost columns). The probability of passing the characteristic is 2^{-135} for the first pair and 2^{-33} for its twin pair, which induces a total probability of 2^{-168}. We therefore need 2^{84} 96-bits vectors u to expect that one pair of pairs passes the characteristic. The same filter filters 70 bits to give us 2^{98} remaining pairs. The same advanced filtering costing 2^8 operations per pair, filters 54 other bits, which gives us 2^{44} remaining twin pairs. In average, each remaining twin pair has one key guess for the three last columns, four key guesses for the first byte, and 2^{24} key guesses for bytes 1,2,3. Exhaustively testing the remaining possible keys has a complexity of $2^{40+2+24} = 2^{70}$.

5.4 Complexity of the Attack

If the key value $k_5 + k_{11}$ has a zero value on the diagonal and the other diagonal values are compatible with the tweak difference, we have a complexity in data of $2^{84} \times 4$ per key guess, a complexity in memory of 2^{85} 96-bits blocks, and a time complexity of $2^8 \times 2^{98}$ per key guess. This attack works with probability 2^{-9}. In addition,

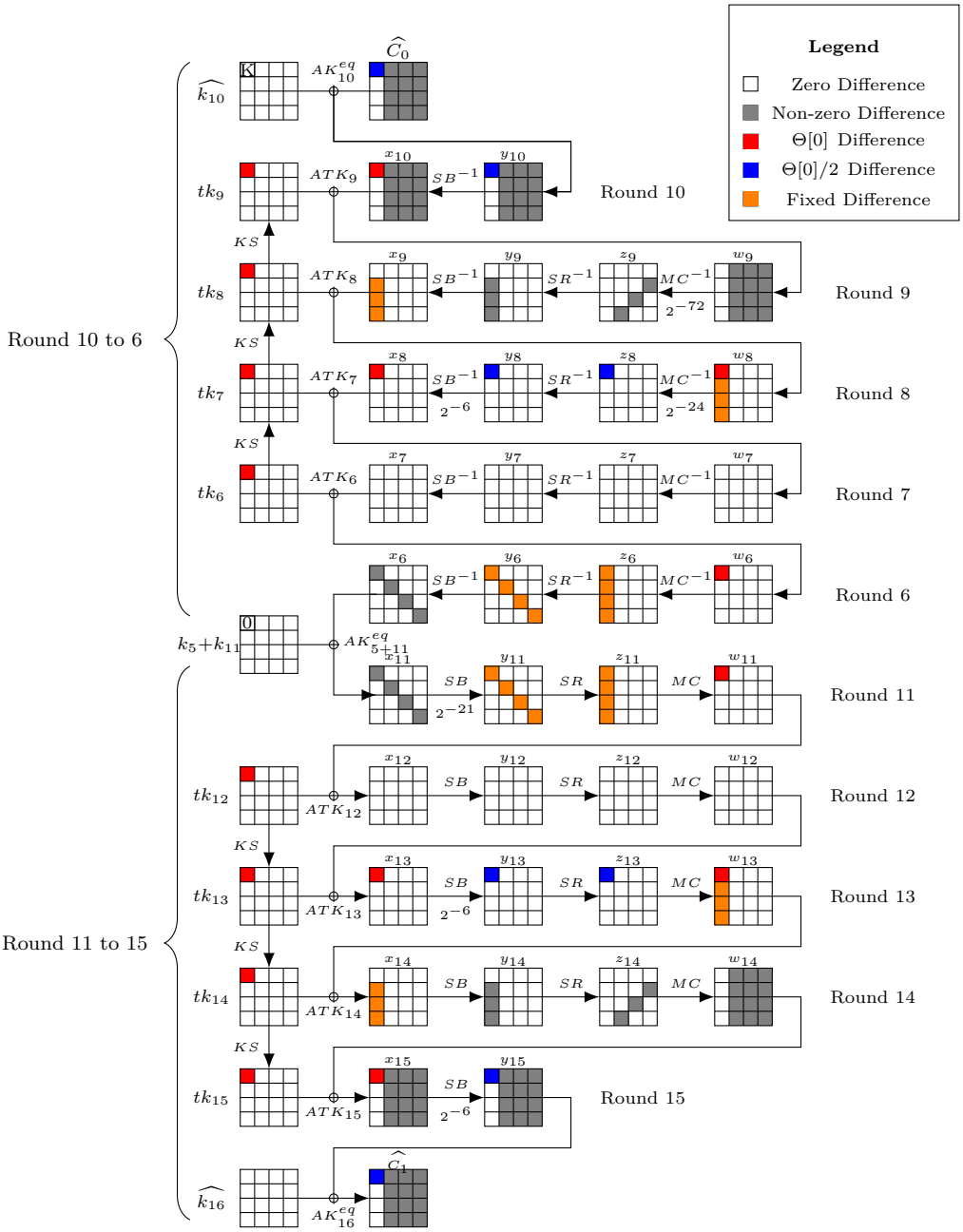


Figure 9: New differential characteristic for weak keys.

we can try this attack with the 3 tweak differences values that verify $\Delta(T[0], T[0]/2) = 2^{-6}$, and we can rotate the columns, to finally give a probability of success of $3/2^{-7}$, for a total (Data,Time,Memory) complexity of $(D, T, M) = (2^{97.58}, 2^{117.58}, 2^{85})$.

We can still trade 2^8 in time complexity for a 2^{11} factor in memory with the intermediate filter described in the next attack.

6
ATTACK AGAINST FULL FORKAES FOR 2^{127} KEYS

6.1 Our New Key Hypothesis

In this attack, we want to attack any key. The only condition on the key is that it should be compatible with the tweak used during the attack. This has a probability of 2^{-1} on every diagonal byte, which makes a total probability of 2^{-4}.

6.2 The New Differential Characteristic

The characteristic is similar to attack of Section 5, with an additionnal 2^{-7} probability of passing the middle rounds. We first guess the value K of the first key byte of $\widehat{k_{10}}$. The first pair satisfies the total characteristic with probability 2^{-142}, and the twin pair with probability 2^{-40}. The pair of pairs satisfies the characteristic with probability 2^{-182}. We therefore need to create 2^{182} pairs with the hope that one satisfies the characteristic, requiring a set of 2^{91} 96-bits vectors. In this case, we are left with 2^{112} twin pairs after the 70 bits filter. We will introduce a new efficient intermediate filter that will filter enough pairs to apply the complete filter on every remaining pair without exceeding 2^{128} computations.

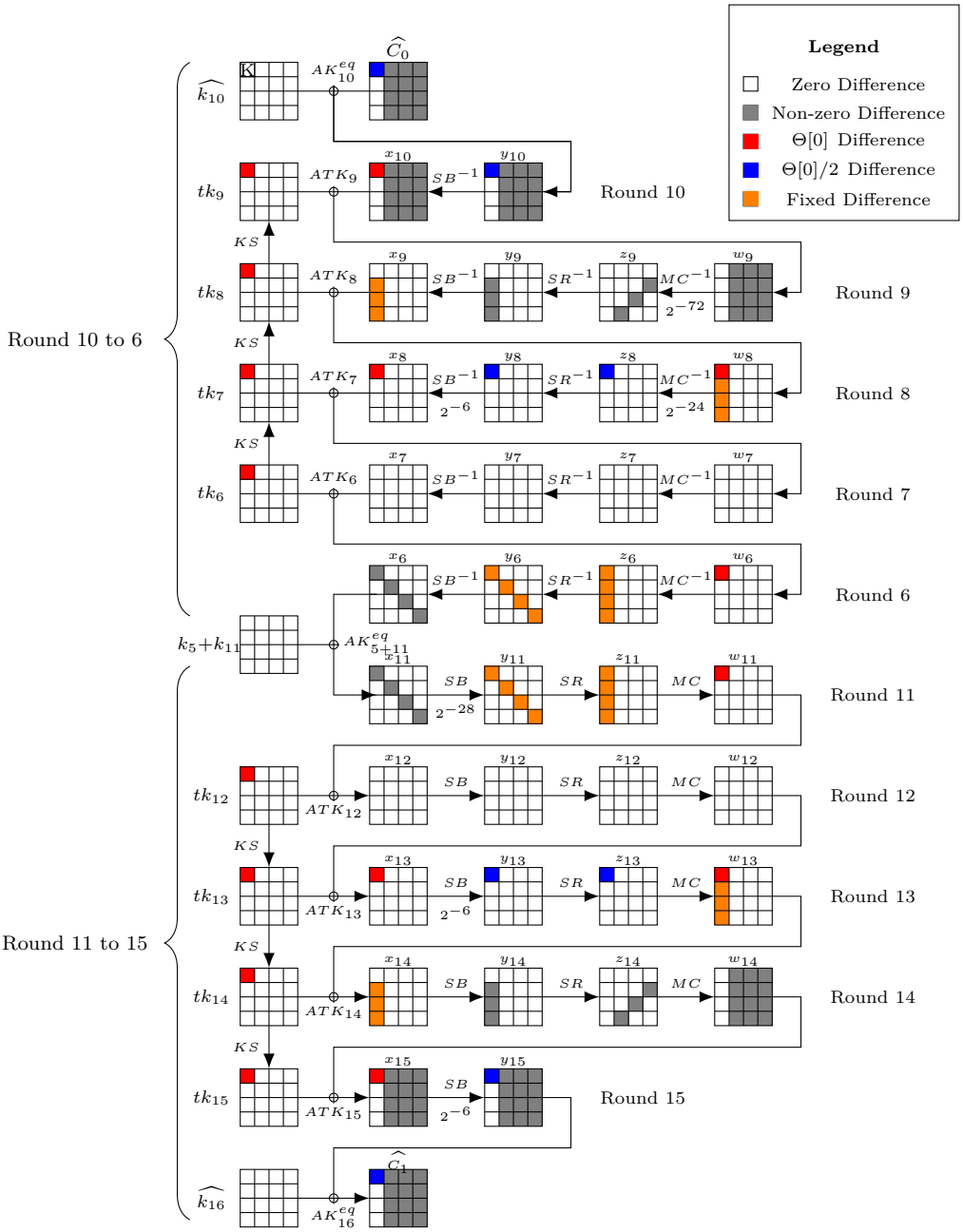


Figure 10: Differential characteristic under the hypothesis that the tweak is compatible with the key.

6.3 Intermediate Filter

This intermediate filter aims at reducing the number of possible pairs that we get at the end of the 70 bits filter, with a small cost per pair. We construct a table as follows :

1. First, we consider the 2^7 possible difference values of $y_{14}[1]$, considering that the difference $x_{14}[1]$ is fixed and can be computed with linear operations from the tweak difference. After spreading this difference to the last column of x_{15}, we come out with a table of 2^7 column differences. Now, we only keep the first 3 bytes of each column difference.

2. The second step consists in iterating over all possible values of bytes 12,13,14 of both elements of a pair of ciphertext. In total, there are $2^{6\times 8}$ possible values. For every possible value and for each column difference computed in the last step, we store possible three-byte key values $AK_{eq}[12, 13, 14]$, deduced from the differential condition before and after the S-box operation $x_{15} \rightarrow y_{15}$ with in average one solution. For each 48-bit value, we store in average 2^7 guesses for $AK_{eq}[12, 13, 14]$.

3. Eventually, we create a hashtable taking 12 bytes of input, corresponding to a 6-byte value for each pair, and returning one bit of output, 1 if and only if the two pairs have a common three byte value $AK_{eq}[12, 13, 14]$. If this is not the case, the pairs are incompatible. Because in average every pair has 2^7 possible values for the 3 bytes $AK_{eq}[12, 13, 14]$, two random pairs have a common key value with probability $2^7 \times 2^7 \times 2^{-24} = 2^{-10}$. This table is precomputed at the beginning of the algorithm and uses 2^{96} bits of memory.

6.4 End of the Attack

For the 2^{112} remaining twin pairs that we have at the end of the 70 bits filter, we call the above table to check if they are compatible. This intermediate filter directly reduces the number of remaining pair to 2^{102}, and at this point we use the previous complete filter. The intermediate filter being a subfilter of the complete filter, we are left with $2^{112} \times 2^{-54} = 2^{58}$ remaining pairs. We proceed to a final exhaustive search of complexity $2^{58} \times 2^{26} = 2^{84}$.

6.5 Complexity of the Attack

The data complexity of the attack is $4 \times 2^{91} \times 2^8$ to create the 2^{182} pairs of pairs for each key guess. The memory complexity is 2^{96} bits for the precomputed table, and 2^{86} for the hashtables storage. The time complexity corresponds to the beginning of the pair processing, which is 2^{112} memory accesses per key guess, which makes 2^{120} memory accesses.

In addition, this attack has a 2^{-4} probability of success, because the key must be compatible with the tweak difference in the middle round. We can use this attack with the three different tweaks values seen in Section 2.3, and we can rotate the columns to have 12 tries in total. We have a time complexity of 12×2^{120} memory accesses. The time complexity of the complete attack is therefore $2^{120} \times 12 = 2^{123.58}$ memory accesses. The average total data complexity of the attack is $2^{101} \times 12 = 2^{104.58}$. The memory complexity does not change.

$(D, T, M) = (2^{104.58}, 2^{123.58}, 2^{96})$.

6.6 Probability of Success

The probability that a tweak difference is compatible with the key is 2^{-4}, and the probability to have a right pair among our 2^{182} pairs is $1-(1-2^{-182})^{2^{182}} \approx 1-1/e$. Therefore, the probability to find a right key is $(1-1/e)\times 2^{-4}$

for each of our twelve tweak differences, so in total the probability of success is $1-(1-(1-1/e)\times 2^{-4})^{12} \approx 0.38$ for a total time complexity of $2^{123.58}$. Note that there is a possible tradeoff between data/time complexities and the probability of success. A fixed key has a probability $1-(1-1\times 2^{-4})^{12} \approx 0.54$ of being compatible with one of the 12 tweak differences. Increasing the data increases the probability that a compatible tweak leads to a right pair. To reach a success probability of 1/2, we need to increase the data complexity to $2^{105.2}$ and the time complexity to $2^{124.8}$.

REFERENCES

[ARVV18] Elena Andreeva, Reza Reyhanitabar, Kerem Varici, and Damian Vizár. Forking a blockcipher for authentificated encryption of very short messages. Cryptology ePrint Archive, Report 2018/916, 2018. `http://eprint.iacr.org/2018/916`.

[BBJ+19] Subhadeep Banik, Jannis Bossert, Amit Jana, Eik List, Stefan Lucks, Willi Meier, Mostafizar Rahman, Dhiman Saha, and Yu Sasaki. Cryptanalysis of ForkAES. In Robert H. Deng, Valérie Gauthier-Umaña, MartÃŋn Ochoa, and Moti Yung, editors, *ACNS 19*, volume 11464 of *LNCS*, pages 43–63, Bogota, Colombia, June 5–7, 2019. Springer, Heidelberg, Germany.

[DEM16] Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Square attack on 7-round kiasu-BC. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 16*, volume 9696 of *LNCS*, pages 500–517, Guildford, UK, June 19–22, 2016. Springer, Heidelberg, Germany.

[DFJ13] Patrick Derbez, Pierre-Alain Fouque, and Jérémy Jean. Improved key recovery attacks on reduced-round AES in the single-key setting. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 371–387, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher Square. In Eli Biham, editor, *FSE'97*, volume 1267 of *LNCS*, pages 149–165, Haifa, Israel, January 20–22, 1997. Springer, Heidelberg, Germany.

[DL17] Christoph Dobraunig and Eik List. Impossible-differential and boomerang cryptanalysis of round-reduced kiasu-BC. In Helena Handschuh, editor, *CT-RSA 2017*, volume 10159 of *LNCS*, pages 207–222, San Francisco, CA, USA, February 14–17, 2017. Springer, Heidelberg, Germany.

[DR01] Joan Daemen and Vincent Rijmen. Specification for the advanced encryption standard (aes). Federal Information Processing Standards Publication 197, 2001. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[JNP14a] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Kiasu v1.1. First-round submission to the CAESAR competition, 2014. `https://competitions.cr.yp.to/round1/kiasuv1.pdf`.

[JNP14b] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 274–288, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany.

[LRW11] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. *Journal of Cryptology*, 24(3):588–613, July 2011.

[MAY16] Tolba Mohamed, Abdelkhalek Ahmed, and Amr Youssef. A meet in the middle attack on reduced round kiasu-bc, 2016.

[MDRMH10] Hamid Mala, Mohammad Dakhilalian, Vincent Rijmen, and Mahmoud Modarres-Hashemi. Improved impossible differential cryptanalysis of 7-round AES-128. In Guang Gong and Kishan Chand Gupta, editors, *INDOCRYPT 2010*, volume 6498 of *LNCS*, pages 282–291, Hyderabad, India, December 12–15, 2010. Springer, Heidelberg, Germany.