Improved Attacks on Full GOST

Itai Dinur[1], Orr Dunkelman[1,2] and Adi Shamir[1]

[1] Computer Science department, The Weizmann Institute, Rehovot, Israel
[2] Computer Science Department, University of Haifa, Israel

Abstract. GOST is a well known block cipher which was developed in the Soviet Union during the 1970's as an alternative to the US-developed DES. In spite of considerable cryptanalytic effort, until very recently there were no published single key attacks against its full 32-round version which were faster than the 2^{256} time complexity of exhaustive search. In February 2011, Isobe used the previously discovered reflection property in order to develop the first such attack, which requires 2^{32} data, 2^{64} memory and 2^{224} time. In this paper we introduce a new fixed point property and a better way to attack 8-round GOST in order to find improved attacks on full GOST: Given 2^{32} data we can reduce the memory complexity from an impractical 2^{64} to a practical 2^{36} without changing the 2^{224} time complexity, and given 2^{64} data we can simultaneously reduce the time complexity to 2^{192} and the memory complexity to 2^{36}.

Keywords: Block cipher, cryptanalysis, GOST, reflection property, fixed point property, 2D meet in the middle attack

1 Introduction

During the 1970's, the US decided to publicly develop the Data Encryption Standard (DES), which was the first standardized block cipher intended for civilian applications. At roughly the same time, the Soviet Union decided to secretly develop GOST [14], which was supposed to be used in civilian applications as well but in a more controlled way. The general design of GOST was finally published in 1994, but even today some of the crucial elements (e.g., the choice of Sboxes) do not appear in the public description, and a different choice can be made for each application.

GOST is a Feistel structure over 64-bit blocks. The round function consists of adding (modulo 2^{32}) a 32-bit round key to the right half of the block, and then applying the function f described in Figure 1. This function has an Sbox layer consisting of eight different 4×4 Sboxes, followed by a rotation of the 32-bit result by 11 bits to the left using the little-endian format (i.e. the LSB of the 32-bit word enters the rightmost entry of the first Sbox).

The full GOST has 32 rounds, and its key schedule is extremely simple: the 256-bit key is divided into eight 32-bit words $(K_1, K_2, ..., K_8)$. Each round of GOST uses one of these words as a round key in the following order: in the first 24 rounds, the keys are used in their cyclic order (i.e. K_1 in rounds 1,9,17, K_2 in rounds 2,10,18, and so forth). In the final 8 rounds (25–32), the round keys are used in reverse order (K_8 in round 25, K_7 in round 26, and so forth).

A major difference between the design philosophies of DES and GOST was that the publicly available DES was intentionally chosen with marginal parameters (16 rounds, 56-bit keys), whereas the secretive GOST used larger parameters (32 rounds, 256-bit keys) which seemed to offer an extra margin of security. As a result, DES was broken theoretically (by using differential and linear techniques) and practically (by using special purpose hardware) about 20 years ago, whereas in the case of GOST, all the single key attacks [1, 9, 17] published before 2011 were only applicable to reduced-round versions of the cipher.[1]

The first single key attack on the full 32-round version of GOST was published by Isobe at FSE'11 [8]. It exploited a surprising reflection property which was first pointed out by Kara [9] in 2008: Whenever the left and right halves of the state after 24 rounds are equal (which happens

[1] Attacks on full GOST in the stronger related-key model are known for about a decade, see [7, 10, 11, 16, 17].

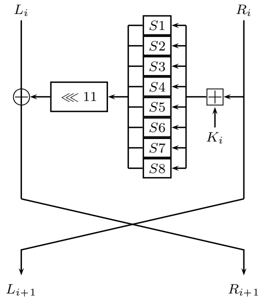


Fig. 1. One round of GOST

with probability 2^{-32}), the last 16 rounds become the identity mapping, and thus the effective number of rounds is reduced from 32 to 16. Isobe developed a new key-extraction algorithm for the remaining 16 rounds of GOST which required 2^{192} time and 2^{64} memory, and used it 2^{32} times for different plaintext/ciphertext pairs in order to get the full 256-bit key using a total of 2^{32} data, 2^{64} memory, and 2^{224} time. This is much faster than exhaustive search, but neither the time complexity nor the memory complexity are even close to being practical.

Shortly afterwards, Courtois [4] published on ePrint a new attack on the full GOST. It uses a very different algebraic approach, but had an inferior complexity of 2^{64} data, 2^{64} memory, and 2^{248} time. Later, Courtois and Misztal [5] described a differential attack which again used 2^{64} data and memory, but reduced the time complexity to 2^{226}.

In this paper we improve several aspects of these previously published attacks. We describe a new *fixed point property*, and show how to use either the previous reflection property or the new fixed point property in order to reduce the general cryptanalytic problem of attacking the full 32-round GOST into an attack on 8-round GOST with two known input-output pairs. We then develop a new way to extract all the 2^{128} possible values of the full 256-bit key given only two known 64-bit input-output pairs of 8-round GOST, which requires 2^{128} time and 2^{36} memory[2] (all the previously published attacks on 8-round GOST have an impractical memory complexity of at least 2^{64}). By combining these improved elements, we can get the best known attacks on GOST for the two previously considered data complexities of 2^{32} and 2^{64}.

Our new results on GOST (including the fixed point based attack) use well known and easy to analyze cryptanalytic techniques such as "Guess and Determine" and "meet-in-the-middle". A month after this paper appeared on eprint [6] (and more than four months after its results were publicly disclosed in a public talk by Adi Shamir at MIT), Courtois posted to ePrint his independently discovered attacks [3], which use several different algebraic techniques. Some of his attacks are also based on the fixed point property, but all of them have higher claimed complexities: Given 2^{32} data, the best attack in [3] has a time complexity of 2^{224} and a memory complexity of 2^{128}, and given 2^{64} data, the best attack in [3] has a time complexity of 2^{216} and a negligible memory complexity. We include the results of [3] in Table 1 (which summarizes all the previously known single-key attacks on the full GOST, our new results, and Courtois' subsequent results) for the sake of completeness.

An important observation about Isobe's attack is that it uses in an essential way the assumption that the Sboxes are invertible. Since the GOST standard does not specify the Sboxes, and there is no need to make them invertible in a Feistel structure, Isobe's attack might not

[2] We can reduce the memory complexity by an additional factor of 2^{17} (to 2^{19}) if we are willing to increase the time by a factor of 2^{12} (to 2^{140}). This may seem like an unattractive tradeoff since the 2^{36} memory complexity is already practical, but one can argue that 2^{19} words fit into the cache whereas 2^{36} do not, which may result in a big performance penalty.

Reference	Data (KP)††	Memory	Time	Self-Similarity Property	8-Round Attack	Sboxes
[8]	2^{32}	2^{64}	2^{224}	Reflection	-	Bijective
[4]	2^{64}	2^{64}	2^{248}	Other (unnamed)	Algebraic	Russian Banks [15]
[5]	2^{64}	2^{64}	2^{226}	None (differential attack)	-	Russian Banks [15]
[3]†††	2^{32}	2^{128}	2^{224}	Reflection	-	any
[3]†††	2^{64}	Negligible	2^{216}	fixed point	Algebraic	Russian Banks [15]
This paper	2^{64}	2^{36}	2^{192}†	fixed point	2DMITM	any
This paper	2^{64}	2^{19}	2^{204}†	fixed point	low-memory	any
This paper	2^{32}	2^{36}	2^{224}†	Reflection	2DMITM	any
This paper	2^{32}	2^{19}	2^{236}†	Reflection	low-memory	any

† The time complexity can be slightly reduced by exploiting GOST's complementation properties (as described in Appendix C)

†† Known plaintext

††† Published on ePrint after the original version of this paper [6].

Table 1. Single-key Attacks on the Full GOST

be applicable to some valid incarnations of this standard. A similar problem occurs in most of Courtois' attacks [3–5], as their complexities are only estimated for one particular choice of Sboxes described in [15] which is used in the Russian banking system, and it is possible that for other choices of Sboxes the complexities will be different. Our new attacks do not suffer from these limitations, since they can be applied with the same complexity to any given set of Sboxes.

2 Overview of Our New Attacks on the Full GOST

The 32 rounds of GOST can be described using only two closely related 8-round encryption functions. Let $G_{K_{i_1},...,K_{i_j}}$ be j rounds of GOST under the subkeys $K_{i_1},...,K_{i_j}$ (where $i_1,...,i_j \in \{1,2,...,8\}$), and let (P_L,P_R) be a 64-bit plaintext, such its right half, P_R, enters the first round. Then $GOST_K(P_L,P_R) = G_{K_8,...,K_1}(G_{K_1,...,K_8}(G_{K_1,...,K_8}(G_{K_1,...,K_8}(P_L,P_R))))$.

Our new attacks on the full GOST exploit its high degree of self-similarity using a general framework which is shared by other attacks: the algorithm of each attack consists of an outer loop which iterates over the given 32-round plaintext-ciphertext pairs, and uses each one of them to obtain suggestions for two input-output pairs for $G_{K_1,...,K_8}$. For each suggestion of the 8-round input-output pairs, we apply an 8-round attack which gives suggestions for the 256-bit GOST key. We then verify the key suggestions by using some of the other plaintext-ciphertext pairs. The self-similarity properties of GOST ensure that the 8-round attack needs to be applied a relatively small number of times, leading to attacks which are much faster than exhaustive search.

We describe several attacks on the full GOST which belong to this common framework but differ according to the property and the type of 8-round attack we use. The two self-similarity properties are:

1. The *reflection property* which was first described in [9], where it was used to attack 30 rounds of GOST (and 2^{224} weak keys of the full GOST). This property was later exploited in [8] to attack the full GOST for all keys. We describe this property again in Section 3.1 for the sake of completeness.

2. A new *fixed point property* which is described in Section 3.2.

The two properties differ according to the amount of data required to satisfy them, and thus offer different points on a time/data tradeoff curve.

Given two 8-round input-output pairs, we describe in this paper several possible attacks of increasing sophistication:

1. A very basic meet-in-the-middle (MITM) attack [2], which is described in Section 4.1.
2. An improved MITM attack, described in Section 4.2, which uses the idea of equivalent keys (first described by Isobe in [8]).
3. A low-memory attack, described in Section 5, which requires 2^{19} memory and 2^{140} time.
4. A new *2-dimensional meet-in-the-middle* (2DMITM) attack, described in Section 6, which requires 2^{36} memory and 2^{128} time.

In order to attack the full GOST, we first select one of the two self-similarity properties to use in the outer loop of the attack according to the number of plaintext-ciphertext pairs available: in case we have 2^{64} pairs available, we select the fixed point property, and if we only have 2^{32} pairs, we select the reflection property. We then select one of last two 8-round attacks according to the amount of available memory: in case we have 2^{36} memory available, we select the 2DMITM attack, and if we only have 2^{19} memory, we select the low-memory attack. The outcome of this selection is an attack algorithm of the form:

1. For each plaintext-ciphertext pair (P, C):
   (a) Assuming that (P, C) satisfies the conditions of the self-similarity property, derive suggestions for two 8-round input-output pairs (I, O) and (I^*, O^*).
   (b) For each suggestion for (I, O) and (I^*, O^*):
      i. Execute the 8-round attack on (I, O) and (I^*, O^*) in order to derive suggestions for the key, and test each suggestion by performing trial encryptions on the remaining plaintext-ciphertext pairs.

The total time complexity of our attacks is calculated by multiplying the complexity of the 8-round attack by the expected number of times it needs to be applied according to the self-similarity property: An arbitrary (P, C) pair satisfies the fixed point property with probability of about 2^{-64}. Thus, it requires about 2^{64} known (P, C) pairs to succeed with high probability, and since we do not know in advance which pair satisfies the property, we need to repeat step 1 of the attack 2^{64} times. For each (P, C) pair, the fixed point property immediately suggests two 8-round input-output pairs (which are correct if the pair indeed satisfies the property). Hence, we need to perform step 1.(b) of the attack only once per (P, C) pair. In total, we need to execute the 8-round attack about 2^{64} times. On the other hand, an arbitrary (P, C) pair satisfies the reflection property with a much higher probability of about 2^{-32}. Thus, it requires about 2^{32} known (P, C) pairs, and we need to repeat the attack only 2^{32} times. However, for each (P, C) pair, the reflection property suggests a large number of 2^{64} values for (I, O) and (I^*, O^*) (out of which only one is correct if the pair indeed satisfies the property). Hence, we need to perform step 1.(b) of the attack 2^{64} times per (P, C) pair. In total, we need to execute the 8-round attack about $2^{32+64} = 2^{96}$ times.

Altogether, we obtain four new attacks on the full GOST. In three out of the four cases, we obtain better combinations of complexities than in all the previously published attacks. In the remaining case, we use the reflection property and the low-memory 8-round attack to significantly reduce the memory requirements of Isobe's attack [8], at the expense of a small time complexity penalty. We note that the computation required by each one of our attacks can be easily parallelized, and thus using x CPUs reduces the expected running time of the attack by a factor of x.

As described in Appendix C, the time complexity of all these attacks can be slightly reduced by exploiting GOST's complementation properties. However, in some of these improved attacks we have to use chosen rather than known plaintexts, which reduces their attractiveness.

3 Obtaining Two 8-Round Input-Output Pairs for GOST

In this section, we describe the two self-similarity properties of GOST which we exploit in order to obtain two 8-round input-output pairs.

3.1 The Reflection Property [8, 9]

Assume that the encryption of a plaintext P after 24 rounds of GOST results in a 64-bit value Y, such that the 32-bit right and left halves of Y are equal (i.e. $Y_R = Y_L$). Thus, exchanging the two halves of Y at the end of round 24 does not change the intermediate encryption value. In rounds 25–32, the round keys K_1–K_8 are applied in the reverse order, and Y undergoes the same operations as in rounds 17–24, but in the reverse order. As a result, the encryption of P after 32 rounds, which is the ciphertext C, is equal to its encryption after 16 rounds (see Figure 2). By guessing the state of the encryption of P after 8 rounds, denoted by the 64-bit value X, we obtain two 8-round input-output pairs (P, X) and (X, C). For an arbitrary key, the probability that a random plaintext gives such a symmetric value Y after 24 rounds is 2^{-32}, implying that we have to try about 2^{32} known plaintexts (in addition to guessing X) in order to obtain the two pairs. Note that the reflection property actually gives us another "half pair" $(\widehat{C}, Y)$, where the 64-bit word $\widehat{C}$ is obtained from C by exchanging the right and left 32-bit halves of C, and the 32-bit right and left halves of Y are equal.[3] However, it is not clear how to exploit this additional knowledge in order to significantly improve the running time of our attacks on the full GOST which are based on the reflection property.

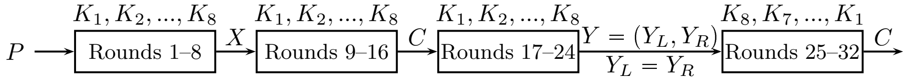


Fig. 2. The Reflection Property of GOST

3.2 The Fixed Point Property

Assume that for a plaintext P, $G_{K_8,...,K_1}(P) = P$. Since rounds 9–16 and 17–24 are identical to rounds 1–8, we obtain P after 16 and 24 rounds as well. In rounds 25–32, the round keys K_1–K_8 are applied in the reverse order, and we obtain some arbitrary ciphertext C (see Figure 3). The knowledge of P and C immediately gives us the 8-round input-output pairs (P, P) and $(\widehat{C}, \widehat{P})$ (in which the right and left 32-bit halves of P and C are exchanged).

For an arbitrary key, the probability that a random plaintext is a fixed point is about 2^{-64}, implying that we need about 2^{64} known plaintexts to have a single fixed point, from which we

[3] In our attacks, we use 8-round input-output pairs whose encryption starts with K_1 and thus need to apply the Feistel structure in the reverse order (starting from round 32) for input-output pairs obtained for rounds 25–32. Since in Feistel structures the right and left halves of the block are exchanged at the end (rather than at the beginning) of the round function, we exchange the right and left sides of the input and the output of the input-output pairs obtained for rounds 25–32. We call $(\widehat{C}, Y)$ a "half pair" since we have to guess only 32 additional bits in order to find it, once (P,C) is known.

obtain the two input-output pairs needed in our attack. If we have only $c{\cdot}2^{64}$ known plaintexts for some fraction c, we expect this fixed point to occur among the given plaintexts with probability c, and thus the time complexity, the data complexity, and the success probability are all reduced by the same linear factor c. Consequently, it makes sense to try the fixed point based attack even when we are given only a small fraction of the entire code book of GOST. Such a graceful degradation when we are given fewer plaintexts (which also occurs for the reflection property) should be contrasted with other attacks such as slide attacks, in which we have to wait for some random birthday phenomenon to occur among the given data points. Since the existence of birthdays has a much sharper threshold, the probability of finding an appropriate pair of points goes down quadratically rather than linearly in c, and thus they are much more likely to fail in such situations.

We note that our fixed point property is closely related to a previously published property which (in addition to the assumption the P is an 8-round fixed point) also assumes that the right and left halves of P are equal. Such a plaintext exists for an arbitrary key with probability 2^{-32} and thus was used in [9] to attack 2^{224} weak keys of the full GOST. The same property was also used later in [13] in cryptanalysis of the GOST hash function.

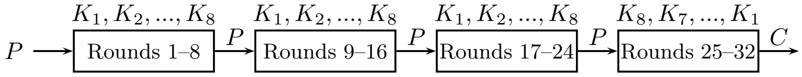


Fig. 3. The fixed point property of GOST

4 Simple Meet-in-the-middle Attacks on 8 Rounds of GOST

Meet-in-the-middle (MITM) attacks can be efficiently applied to block ciphers in which some intermediate encryption variables (bits, or combinations of bits) depend only on a subset of key bits from the encryption side and on another subset of key bits from the decryption side: the attacker guesses the relevant key bits from the encryption and the decryption sides independently, and tries only keys in which the values suggested by the computed intermediate variables match. While the full 32-round GOST resists such attacks, 8-round GOST uses completely independent round keys. Thus, the full 64-bit value after 4 encryption rounds depends only on round keys K_1–K_4 from the encryption side and on round keys K_5–K_8 from the decryption side.

4.1 The Basic Meet-in-the-middle Attack

We describe how to mount a simple meet-in-the-middle attack on 8 rounds of GOST given two 8-round input-output pairs and several additional 32-round plaintext-ciphertext pairs:

1. For each of the 2^{128} possible values of K_1–K_4, encrypt both inputs and obtain two 64-bit intermediate encryption values after 4 rounds of GOST (i.e., 2^{128} intermediate values of 128 bits each). Store the intermediate values in a list, sorted according to these 128 bits, along with the corresponding value of K_1–K_4.
2. For each of the 2^{128} possible values of K_5–K_8, decrypt both outputs, obtain two 64-bit intermediate values and search the sorted list for these two values.
3. For each match, obtain the corresponding value of K_1–K_4 from the sorted list and derive a full 256-bit key by concatenating the value of value of K_1–K_4 with the value of K_5–K_8 of the previous step. Using the full key, perform a trial encryption of several plaintexts and return the full key, i.e., the one that remains after successfully testing the given 32-round pairs.

We expect to try about $2^{128+128-128} = 2^{128}$ full keys in step 3 of the attack, out of which only the correct key is expected to pass the exhaustive search of step 3. Including the 2^{128} 8-round encryptions which are performed in each of the first two steps of the attack, the total time complexity of the attack is slightly more than 2^{128} GOST encryptions. The memory complexity of the attack is about 2^{128} words of 256 bits.[4]

4.2 An Improved Meet-in-the-middle Attack Using Equivalent Keys

In this section, we use a more general variant of Isobe's equivalent keys idea [8] to significantly improve the memory complexity of the attack. Both our and Isobe's MITM attacks are based on a 4-round attack that uses one 4-round input-output pair to find all the 2^{64} possible values of subkeys K_1–K_4 that yield this pair. However, our MITM attack is more general since we can attack all possible incarnations of the GOST standard, whereas Isobe's attack works only on those which use bijective Sboxes.[5] An additional advantage of our MITM attack over Isobe's one, is that our attack can use any two input-output pairs for 8-round GOST, regardless of how they are obtained. We can thus use the same algorithm to exploit both the reflection and the fixed point properties. On the other hand, Isobe's attack is restricted to the case of a single input-output pair obtained for the first 16 rounds of GOST (by guessing the intermediate values obtained after 4 and 12 rounds) and thus can be combined with the reflection property, but cannot be directly applied to the two input-output pairs produced by the fixed point property.

We now describe Isobe's 4-round attack procedure: Denote the 4-round input (divided into two 32-bit words) by (X_L, X_R) and the output by (Y_L, Y_R). Denote the middle values (after the second round) by (Z_L, Z_R) (see Figure 4). Then:

$$Z_L = X_L \oplus f(X_R \boxplus K_1)$$

$$Z_R = Y_R \oplus f(Y_L \boxplus K_4)$$

$$Y_L \oplus Z_L = f(Z_R \boxplus K_3)$$

$$X_R \oplus Z_R = f(Z_L \boxplus K_2)$$

Isobe's attack assumes bijective Sboxes (making f invertible), and finds the equivalent keys as follows:[6] for each value of K_1, K_2, compute Z_L from the first equation and Z_R from the fourth equation. From the second equation we have: $K_4 = f^{-1}(Z_R \oplus Y_R) \boxminus Y_L$ and from the third equation: $K_3 = f^{-1}(Z_L \oplus Y_L) \boxminus Y_R$.

Our 8-round attack is a variant of Isobe's MITM attack, given two 8-round input-output pairs (I, O) and (I^*, O^*):

1. For each possible value of the 64-bit word $Y = (Y_L, Y_R)$ obtained after 4 encryption rounds of I:
   (a) Apply the 4-round attack on (I, Y) to obtain 2^{64} candidates for K_1–K_4.
   (b) Partially encrypt I^* using the 2^{64} candidates and store $Y^* = (Y^*_L, Y^*_R)$ in a list with K_1–K_4.

[4] Note that it is possible obtain a time-memory tradeoff: we partition the 2^{128} possible values of K_1–K_4 into 2^x sets of size 2^{128-x} (for $0 \leq x \leq 128$), and run the second and third steps of the attack independently for each set. Thus, the memory complexity decreases by a factor 2^x to 2^{128-x}, and the time complexity increases by a factor of 2^x to 2^{128+x}.

[5] The Feistel structure of GOST does not require bijective Sboxes and the published standard does not discuss this issue, but all the known choices of Sboxes happen to be bijective (perhaps due to the weakness of non-bijective Sboxes against differential cryptanalysis).

[6] In case f is not bijective, then for a random (X_L, X_R) and (Y_L, Y_R) there exist an average of 2^{64} equivalent keys which can be found using a simple preprocessing MITM algorithm that requires about 2^{64} time and memory.

(c) Apply the 4-round attack on (Y,O) to obtain 2^{64} candidates for K_5–K_8.
(d) Partially decrypt O^* using each one of the 2^{64} candidates and obtain $Y^* = (Y^*_L, Y^*_R)$.
(e) Search the list obtained in step (b) for Y^*, and test the full 256-bit keys for which there is a match.

The expected time complexity of steps (a–d) is about 2^{64} (regardless of the algorithm that is used to find the equivalent keys). The time complexity of step (e) is also about 2^{64} since we expect to try about $2^{64+64-64} = 2^{64}$ full keys. Steps (a–e) are performed 2^{64} times, hence the total time complexity of the attack is about 2^{128} GOST encryptions, which is similar to the first attack. However, the memory complexity is significantly reduced from 2^{128} to slightly more than 2^{64} words of 64 bits.

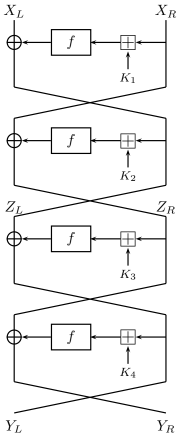


Fig. 4. Four Rounds of GOST

5 A New Attack on 8 Rounds of GOST with Lower Memory Complexity

Simple meet-in-the-middle attacks, such as the ones described in Sections 4.1 and 4.2 are much faster than exhaustive search for the entire 256-bit key. However, they do not fully exploit the slow diffusion of the key bits in 4 rounds of GOST. As a result, these MITM attacks use a large amount of memory to store the many intermediate encryption values obtained for all the possible values of large sets of key bits. In this section, we describe an improved 8-round attack which exploits the slow diffusion properties of 4 rounds of GOST in order to reduce the memory complexity from the impractical value of 2^{64} to the very practical value of 2^{19} words of memory, with a very small time complexity penalty. The main idea of this attack is to guess the 4 round keys K_5–K_8 and apply an optimized "Guess and Determine" attack on the remaining 4 rounds using two input-output pairs. In the 4-round attacks we have 128-bits of unknown key and 128 bits of input-output pairs. Thus, we expect that only one value for K_1–K_4 exists (although there are likely to be input-output pairs for which the encryptions of the inputs does not match the outputs for any of the keys, and input-output pairs for which the encryptions of the inputs matches the outputs for several values of K_1–K_4).

In the rest of this section we describe the algorithm for deriving the 32 bits of K_1 and the 32 bits of K_4. Afterwards, deriving the values of K_2 and K_3 is immediate using the third and forth equations of Section 4.2 (Z_L and Z_R are known from the first and second equations).

5.1 Overview of the “Guess and Determine” Attack on 4-Round GOST

Now that we deal with 4-round GOST, we apply a typical “Guess and Determine” attack which traverses a tree of partial guesses for the round keys K_1 and K_4 and intermediate encryption values. The tree is composed of layers of nodes ℓ_i for integral $0 \leq i \leq k$, where each layer contains nodes that specify the potential values (i.e. guesses) for a certain subset of key and intermediate encryption values. In each layer we expand each node by guessing the values of a small number of additional key bits and state bits that are needed to calculate some intermediate encryption bits, both from the encryption and the decryption sides. We then calculate the bits by evaluating the Feistel structure from both sides on a small number of bits, compare the values obtained, and discard guesses in which the values do not match (i.e., we discard child nodes that do not satisfy a predicate which checks the consistency of intermediate encryption values).

We traverse the partial guess tree starting from the root using DFS (which requires only a small amount of memory). In our attack, the nodes of the last layer of the tree ℓ_k contain guesses for the full key, which can be verified using trial encryptions.

The total number of operations performed during the traversal is proportional to the total number of nodes in the tree. However, the operations performed when expanding a single node work only on a few bits (rather than on full words). At the same time, when expanding a full path of nodes in the tree from the root to the last layer, we work on the full-size Feistel structure to obtain a guess for the full key. Hence, we estimate the time complexity of expanding a full path by a single Feistel structure evaluation on a full 64-bit input. Using this estimation, we can upper bound the time complexity of the tree traversal (in terms of Feistel structure evaluations) as the width of the tree, or the size of the layer which contains the highest number of nodes. Note that when counting the number of nodes in a layer for the time complexity analysis, we must also include nodes that were expanded and discarded since they do not satisfy the predicate of the previous layer.

5.2 Notations

Assume that we have two input-output pairs for 4 encryption rounds of GOST under the subkeys K_1, K_2, K_3, K_4. Similarly to Section 4.2, denote the input, output and middle values (after using K_2) for the first pair by (X_L, X_R), (Y_L, Y_R) and (Z_L, Z_R), respectively. For the second pair, denote these values by (X^*_L, X^*_R), (Y^*_L, Y^*_R) and (Z^*_L, Z^*_R) respectively.

Since our attack analyzes 4-bit words (which are outputs of single Sboxes), we introduce additional notations: Define the functions $f^0, f^1, ..., f^7$ where each f^i takes a 4-bit word as an input, and outputs a 4-bit word by applying Sbox i to the input. Denote by W^i the i’th bit of the 32-bit word W, and by $W^{i,j}$ the $(j - i + 1)$-bit word composed of consecutive bits of W starting from bit i and ending at bit j. We treat W as a cyclic word, and thus $W^{24,3}$ contains 12 bits which are bits 24 to 31 and 0 to 3 of W.

5.3 An Attack on 4 Rounds of Simplified GOST

We start by describing an attack on 4 rounds of a simplified variant of GOST (which we call S-GOST), in which the round-key addition is replaced by XOR, and the 11-bit rotation is replaced by 12-bit rotation. The simplified variant is easier to analyze since it provides much slower diffusion of the key bits compared to full GOST: unlike addition, the XOR operation does not produce carries, and since 12 is a multiple of 4, rotating by 12 bits implies that the output of any Sbox effects the input of only a single Sbox in the next round.

We now describe the basic procedure preformed by a node in layer 0 of our guess tree for S-GOST. The procedure requires the value of $K_1^{0,3}$ (whose value we guess before executing the procedure), and expands nodes in the next layer, which suggest a value for the additional 4 bits of $K_4^{20,23}$. The steps of this procedure can be easily verified using a variant of Figure 4 where the addition is replaced by XOR.

1. Given $K_1^{0,3}$ and $X_R^{0,3}$, compute $Z_L^{12,15} \equiv f^0(X_R^{0,3} \oplus K_1^{0,3})$ for both pairs (i.e., given $K_1^{0,3}$ and $X_R^{*0,3}$, compute $Z_L^{*12,15} \equiv f^0(X_R^{*0,3} \oplus K_1^{0,3})$).
2. Obtain $f^0(Z_R^{0,3} \oplus K_3^{0,3}) \equiv Z_L^{12,15} \oplus Y_L^{12,15}$ for both pairs. Then, invert[7] f^0 to obtain $Z_R^{0,3} \oplus K_3^{0,3}$ and $Z_R^{*0,3} \oplus K_3^{0,3}$.
3. XOR the two expressions calculated in step 2, to eliminate $K_3^{0,3}$, and obtain the value of $Z_R^{0,3} \oplus Z_R^{*0,3}$.
4. XOR the 4-bit difference obtained in step 3 to the difference $Y_R^{0,3} \oplus Y_R^{*0,3}$ and obtain the value of $T = Z_R^{0,3} \oplus Y_R^{0,3} \oplus Z_R^{*0,3} \oplus Y_R^{*0,3} \equiv (f(Y_L \oplus K_4) \oplus f(Y_L^* \oplus K_4))^{0,3}$ (from the encryption side).
5. For each of the 2^4 possible values of $K_4^{20,23}$:
   (a) Allocate a node in the next layer.
   (b) Evaluate the expression $f^5(Y_L^{20,23} \oplus K_4^{20,23}) \oplus f^5(Y_L^{*20,23} \oplus K_4^{20,23})$ from the decryption side by plugging the current value of $K_4^{20,23}$ into the expression. Discard nodes which do not agree with the value T.

Note that given $K_1^{0,3}$, we expect the procedure above to process a single child in the next layer: in step 5 we have a 4-bit condition on 4 bits of the key $K_4^{20,23}$, and thus we expect one node to satisfy the predicate. Moreover, step 5 can be optimized by using a small amount of precomputation and memory in order to calculate in advance the solutions to the 4-bit condition (as described in Appendix A.1).

We now generalize the procedure above in order to derive more key bits in a similar way:

- Since encryption and decryption are completely symmetric (except the order of the subkeys), steps 1–5 can also be performed from the decryption side: in steps 1–5 we use the value of $K_1^{0,3}$ in order to obtain the value of $K_4^{20,23}$, and thus we define the symmetric steps 6–10 which use the value of $K_4^{20,23}$ in order to obtain the value of $K_1^{20+20,23+20}$, i.e. $K_1^{8,11}$.
- Given any integer $0 \leq i \leq 7$, we can rotate the indices of all the 32-bit words in steps 1–10 by $4i$ bits. Namely, given i, we define analogues steps 1–10 which use the value of $K_1^{4i,4i+3}$ to obtain the value of $K_4^{4i+20,4i+23}$ and $K_1^{4i+8,4i+11}$.

In order to derive the full 32-bit values of K_1 and K_4, we define a tree which contains 9 layers $\ell_0, \ell_1, ..., \ell_8$ (and an additional root node). The nodes of each layer are expanded using the generalized procedure which uses 4 bits of K_1 in order to derive 4 additional bits of K_1 and 4 additional bits of K_4. Since the 10 steps of the procedure for expanding the nodes of layers 0–7 are basically the same, we call this procedure an *iteration*, and index it according to the value of i (which determines the 4-bit chunks that we work on).

5.4 Extending the Attack to 4 Rounds of the Real GOST

In order to extend the iteration procedure from S-GOST to full GOST, we need to make several adjustments. The most significant adjustments are given below:

- Since the round keys are added (and not XORed) to the state, we have to guess the carry bits into the LSBs of several addition operations of 4-bit words. For example, in the expression $f^5(Y_L^{20,23} \boxplus K_4^{20,23}) \oplus f^5(Y_L^{*20,23} \boxplus K_4^{20,23})$ evaluated in step 5, we have to guess two carry bits (one for $Y_L^{20,23}$ and one for $Y_L^{*20,23}$).

[7] We expect one solution on average. However, in case the inversion has more than one solution, we need to try each one. In case the inversion has no solution, we can discard the node.

- GOST uses 11-bit rotation (instead of 12-bit rotation), and thus the 4-bit chunks that we work on in each iteration are not aligned. Consequently, we have to guess additional state bits in order to compare the evaluation of the 4-bit predicates from both sides. For example, since $20 + 11 = 31$, in step 5 of the iteration we actually calculate $(f(Y_L \oplus K_4) \oplus f(Y_L^* \oplus K_4))^{31,2}$ from the decryption side. Thus, we additionally guess bit 31 of this expression from the encryption side.

These adjustment create strong dependencies between iterations with consecutive indexes (i.e., i and $i + 1$), namely:

- The carry bits required by iteration $i + 1$ are known after iteration i. For example, iteration 1 requires the carry into bit 24 of the addition operation $Y_L \boxplus K_4$ (in order to calculate $f^6(Y_L^{24,27} \boxplus K_4^{24,27}) \oplus f^6(Y_L^{*24,27} \boxplus K_4^{24,27})$ in step 5). This bit can be calculated after step 5 of iteration 0, where the 4-bit value of $Y_L^{20,23} \boxplus K_4^{20,23}$ is calculated in order to evaluate the predicate.
- The state bits required by iteration $i+1$ are known after iteration i. For example, iteration 1 requires calculation of bit 3 of the expression $f(Y_L \boxplus K_4) \oplus f(Y_L^* \boxplus K_4)$ from the encryption side. However, this bit is already guessed in step 4 of iteration 0.

This suggests that we perform the iterations in their natural order, namely assign layer ℓ_i iteration i for $0 \leq i \leq 7$. As a result, we need to guess carry and state bits only in the first iteration. Afterwards, the required carry and state bits for each iteration can be calculated by the knowledge from the previous one. On the other hand, we pay a (relatively small) penalty on key bit guesses since key bits required by iteration $i + 2$ are derived in iteration i (and not in iteration $i + 1$). Since iteration i requires key bits $K_1^{4i,4i+3}$, we need to guess 4 key bits in both iterations 0 and 1 ($K_1^{0,3}$ and $K_1^{4,7}$). For iterations $i \geq 2$, the required key bits are already derived in previous iterations (as shown in Table 2).

We note that since there is no carry into the LSBs of addition operations, starting the process with iteration 0 has the advantage that we do not need to guess the carries for all the addition operations (e.g., we do not need to guess the carry into the addition $f^0(X_R^{0,3} \boxplus K_1^{0,3})$ in step 1).

The full details and analysis of the "Guess and Determine" attack are given in Appendix A, most of which is not required in order to understand the rest of this paper. It shows that the expected number of nodes in the widest layer of the partial guess tree is 2^{14}, and it is obtained at iterations 1 to 5 (this was also verified using simulations performed on a PC). Basically, the number 2^{14} is obtained due to the 8 key-bit guesses ($K_1^{0,3}$ and $K_1^{4,7}$) and 6 additional carry and state bit guesses in iteration 0. This gives an expected time complexity of about 2^{14} 4-round Feistel structure evaluations for two input-output pairs, which is equivalent to about 2^{12} full GOST evaluations. Since we apply this 4-round attack 2^{128} times, the time complexity of the 8-round attack is about $2^{128+12} = 2^{140}$ GOST evaluations. In terms of memory, the attack has a completely practical complexity of 2^{25} bits, which is equivalent to 2^{19} 64-bit words.

Iteration	0	1	2	3	4	5	6	7
K_1 bits derived	0–3 8–11	4-7 12–15	$\underline{\text{8–11}}$ 16–19	$\underline{\text{12–15}}$ 20–23	$\underline{\text{16–19}}$ 24–27	$\underline{\text{20–23}}$ 28–31	$\underline{\text{24–27}}$ $\underline{\text{0–3}}$	$\underline{\text{28–31}}$ $\underline{\text{4–7}}$
K_4 bits derived	20–23	24–27	28–31	0–3	4–7	8–11	12–15	16–19

The key bits which are already known from previous iterations are underlined.

Table 2. The key bits derived in each iteration

6 A New 2-Dimensional Meet-in-the-middle Attack on 8 Rounds of GOST

In this section, we present a new attack on 8 rounds of GOST given two input-output pairs, which combines the ideas of the "Guess and Determine" attack (which progresses horizontally across the state) and the MITM attack (which progresses vertically across the rounds). Unlike the attack of the previous section, we do not guess the last 4 round keys in advance. Instead, we divide the 8-round Feistel structure horizontally by splitting it into a *top part*, which uses round keys K_1–K_4, and a *bottom part*, which uses round keys K_5–K_8.

Our main observation is that due to the slow diffusion of the data bits into the state, we can run a substantial part of the "Guess and Determine" attack of Section 5 with very partial knowledge of Y and Y^* (obtained after 4 rounds of encryption). This allows us to split the "Guess and Determine" attack into two partial 4-round attacks which we run a relatively small number of times (once for each value of the bits of Y and Y^* that it requires). Our full 4-round attacks on the top and bottom parts combine the suggestions of the partial attacks in order to suggest values for the 4-round keys. Finally, we use an 8-round attack which joins the suggestions of the two partial attacks in order to obtain suggestions for the full 256-bit key.

Schematically, we split the top and bottom parts of the block cipher vertically into two (potentially overlapping) cells, such that on each cell we execute an independent partial attack to obtain suggestions for some subset of key bits. We then join all the suggestions to obtain suggestions for the full key using three MITM attacks. This can be visualized using a 2 × 2 matrix (as shown in Figure 5), where the horizontal line separates the four initial and final rounds of the 8-round block cipher, and the dashed vertical line separates the left and right cells in each one of the top and bottom parts.

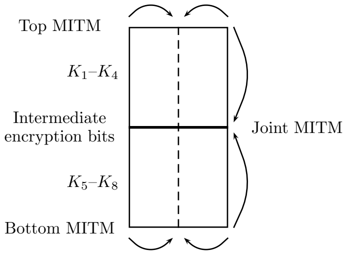


Fig. 5. The general framework of the 2-dimensional meet-in-the-middle attack

After the MITM attacks on the top and bottom parts of the Feistel structure, we obtain 2^{128} suggestions for K_1–K_4 and 2^{128} suggestions for K_5–K_8, each accompanied by corresponding 128-bit values of Y and Y^*. Note that so far we did not filter out any possible keys, and thus the final MITM attack, which compares the 128-bit values of Y and Y^* to obtain about 2^{128} suggestions for the full key, is essentially the basic MITM attack of Section 4.1, which would normally require 2^{128} memory.

To reduce the memory consumption, we guess many of the 128 bits of Y and Y^* in advance (in the outer loop of the 8-round attack). For each possible value of those bits, we execute the 2DMITM (2-dimensional MITM) attack described above, but obtain fewer suggestions for the key which we have to store. This increases the number of times that we execute the partial 4-round attacks and could potentially increase the overall time complexity of the full 8-round attack. However, this is not the case, as the partial 4-round attacks are relatively efficient (the time complexity of each one is at most 2^{18}) and is executed only 2^{82} times. Thus, the partial 4-round attacks are not the bottleneck of the time complexity of the attack.[8]

6.1 Details of the 8-Round Attack

Formally, we define the following sets which contain bits of Y and Y^*:

- S_1 is the set of bits that we guess in the outer loop of the 8-round attack.
- S_2 is chosen such that $S_1 \bigcap S_2 = \emptyset$, and $S_1 \bigcup S_2$ is the minimal set that contains all the bits of Y and Y^* which are required by the partial 4-round attack on the left cell of the top part.
- S_3 is chosen such that $S_1 \bigcap S_3 = \emptyset$, and $S_1 \bigcup S_3$ is the minimal set of bits which are required by the partial 4-round attack on the right cell of the top part.

For the bottom MITM attack, we define S_4 and S_5 in a similar way to S_2 and S_3, respectively. Note that since the 4-round attacks on both the top and bottom parts require all the 128 intermediate bits, $S_2 \bigcup S_3 = S_4 \bigcup S_5$.

The details of the 4-round attacks are given in the next section. We now refer to them as black boxes, and give the algorithm of the full 8-round attack:

1. For each value of the bits of the set S_1:
   (a) Perform the 4-round attack on the top part of the Feistel structure, and obtain a list with values of K_1–K_4, sorted according to the value of the bits of $S_2 \bigcup S_3$.
   (b) Perform the 4-round attack on the bottom part of the Feistel structure. For each value of $S_4 \bigcup S_5 = S_2 \bigcup S_3$ (given along with the value of K_5–K_8), search the list obtained in the previous step of matches. For each match test the full key K_1–K_8.

6.2 Details of the 4-Round Attacks

We concentrate first on the top part of the 8-round Feistel structure: each one of the two partial 4-round attacks on the top part sequentially executes a subset of the iterations defined in Section 5, and is called an iteration *batch*. The first (left) iteration batch executes iterations 0–3, and the second (right) executes iterations 4–7.

After performing iteration batches 0–3 and 4–7 independently, we get suggestions for the values of some key bits, along with some carry and state bits. We then discard inconsistent suggestions by comparing the values of the common bits that are derived by batches. We partition these bits into three groups (which are fully specified in Appendix B):

- G_1 contains 16 key bits which are derived by both of the left and right batches.
- G_2 contains 6 carry and state input bits that we guess in iteration 0. These bits are also contained in the set of output bits of iteration 7 (of the right batch), and can thus be used to discard inconsistent suggestions made by the two batches.
- G_3 contains 10 carry and state input bits that we guess in iteration 4. This bits are also contained in the set of iteration output bits of iteration 3 (of the left batch), and can thus be used to discard inconsistent suggestions made by the two batches.

Assume that the values of all the bits of S_1 are known. We now give the algorithm of the MITM attack performed on the top part of the 8-round Feistel structure:

[8] Note again that we expect about 2^{128} keys to fulfill the filtering conditions of the two input-output pairs. Thus, the time required for the attack to list all of them cannot be reduced below 2^{128} (without exploiting additional filtering conditions).

1. For each value of the bits of S_2, perform the left batch. Save all the nodes of the final layer in a list. These nodes contain the values 40 bits of K_1 and K_4 (including the values of the bits of G_1), and also the values of the bits of G_3. In addition to the information obtained by each node, save the value of the initial guess of the bits of G_2, and the value of the bits of S_2 per node. Sort the list according to the values of G_1,G_2 and G_3.
2. For each value of the bits of S_3, perform the right batch. For each node in the final layer obtain the value of the bits of G_1,G_2 and G_3 and search the list obtained in the first step for their value. For each match, save the value of the full K_1–K_4 in a sorted list according to the value of the bits of $S_2 \bigcup S_3$.

The iteration batches of the MITM attack on the bottom part of the Feistel structure are performed from the decryption side and are completely analogous to the iteration batches on the top part (i.e. in iteration 0, we start by guessing $K_8^{0,3}$, and derive $K_5^{20,23}$ and $K_8^{8,11}$). We also define analogous sets to G_1,G_2 and G_3 for the bottom part.

The specific choices of S_1–S_5 are given in Appendix B. This choice of sets satisfies $|S_1| = 92$ and $|S_2| = |S_3| = |S_4| = |S_5| = 18$.

We now analyze the complexity of the MITM attack on the top part of the Feistel structure: as specified in Section A.2, when starting the iteration batch from iteration 0, the expected maximal size of the tree is 2^{14}. It is obtained after iteration 1, and is maintained until the end of iteration 5 (even though we do not perform 5 consecutive iterations in this attack). The time complexity of the first step of the attack is thus about $2^{|S_2|+14} = 2^{14+18} = 2^{32}$, and this is also the size of the sorted list at the end of the first step. The maximal size of the tree of the iteration batch 4–7 is $2^{14+4} = 2^{18}$ (as described in Appendix B, we have to guess 4 more carry bits compared to iterations 0–3). Thus, the time complexity of expanding the tree in the second step is $2^{|S_3|+18} = 2^{36}$. The time and memory complexities of the remainder of step 2 (in which we match the batches) are $2^{|S_2|+|S_3|+14+18-(|G_1|+|G_2|+|G_3|)} = 2^{|S_2|+|S_3|+14+18-(16+6+10)} = 2^{|S_2|+|S_3|} = 2^{36}$. Note that it is not surprising that the time and memory complexities of the matching part of the attack reduce to $2^{|S_2|+|S_3|}$, since given the full 128-bit intermediate value, we expect that only one key survives the filtering conditions. Altogether, the memory complexity of the top MITM attack is about 2^{36} 64-bit words. The time complexity is dominated by step 2 and is equivalent to about 2^{36} 4-round Feistel structure evaluations, which is equivalent to about 2^{33} evaluations of the full GOST cryptosystem. For the bottom MITM attack, we obtain the same time and memory complexities, since the sizes of S_4 and S_5 are equal to the sizes of S_2 and S_3, and the sets corresponding to G_1, G_2 and G_3 are completely symmetrical.

6.3 The Complexity of the 8-Round Attack on GOST

We can now analyze the complexity of the attack described in Section 6.1: The time complexities of each of the MITM attacks on the bottom and top parts in steps (a) and (b) are equivalent to about 2^{36} 4-round Feistel structure evaluations, as calculated above. The number of expected matches for which we run the full cipher in step (b) is $2^{36+36-36} = 2^{36}$. Hence, the time complexity of these steps is equivalent to a bit more than 2^{36} full GOST evaluations. Since $|S_1| = 92$, the total time complexity of the attack is equivalent to about $2^{92+36} = 2^{128}$ GOST evaluations. The total memory complexity of the attack is about 2^{36} 64-bit words, and is dominated by the sorted list calculated in step (a).

7 Conclusions and Open Problem

In this paper we introduced several new techniques such as the fixed point property and two dimensional meet in the middle attacks, and used them to greatly improve the best known attacks on the full 32-round GOST. In particular, we reduced the memory complexity of the attacks from an impractical 2^{64} to a practical 2^{36} (and to an even more practical 2^{19} complexity, which can fit into the cache of modern microprocessors, with a small penalty in the running time). The lowest time complexity of our attacks is 2^{192}, which is 2^{32} times better than previously published attacks but still very far from being practical. Consequently, we are concerned about the demonstrated weaknesses in the design of GOST (especially in its simplistic key schedule), but do not advocate that its current users should stop using it right away.

The main open problems left in this paper are whether it is possible to find faster attacks, and how to better exploit other amounts of available data (in addition to the 2^{32} and 2^{64} complexities considered in this paper, which are the natural thresholds for our techniques).

Acknowledgements: The authors thank Nathan Keller, Pierre-Alain Fouque and Charles Bouillaguet for useful discussions on this work, and the anonymous referees for their helpful comments on this paper which greatly improved the presentation of our results.

References

1. Eli Biham, Orr Dunkelman, and Nathan Keller. Improved Slide Attacks. In Alex Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 153–166. Springer, 2007.
2. David Chaum and Jan-Hendfik Evertse. Cryptanalysis of DES with a Reduced Number Of Rounds: Sequences of Linear Factors in Block Ciphers. In *Advances in Cryptology, CRYPTO 85*, pages 192–211. Springer-Verlag, 1986.
3. Nicolas T. Courtois. Algebraic Complexity Reduction and Cryptanalysis of GOST. Cryptology ePrint Archive, Report 2011/626, 2011. `http://eprint.iacr.org/`.
4. Nicolas T. Courtois. Security Evaluation of GOST 28147-89 in View of International Standardisation. Cryptology ePrint Archive, Report 2011/211, 2011. `http://eprint.iacr.org/`.
5. Nicolas T. Courtois and Michał Misztal. Differential Cryptanalysis of GOST. Cryptology ePrint Archive, Report 2011/312, 2011. `http://eprint.iacr.org/`.
6. Itai Dinur, Orr Dunkelman, and Adi Shamir. Improved Attacks on Full GOST. Cryptology ePrint Archive, Report 2011/558, 2011. `http://eprint.iacr.org/`.
7. Ewan Fleischmann, Michael Gorski, Jan-Hendrik Huehne, and Stefan Lucks. Key Recovery Attack on full GOST Block Cipher with Negligible Time and Memory. Presented at Western European Workshop on Research in Cryptology (WEWoRC), 2009.
8. Takanori Isobe. A Single-Key Attack on the Full GOST Block Cipher. In Antoine Joux, editor, *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2011.
9. Orhun Kara. Reflection Cryptanalysis of Some Ciphers. In Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, *INDOCRYPT*, volume 5365 of *Lecture Notes in Computer Science*, pages 294–307. Springer, 2008.
10. John Kelsey, Bruce Schneier, and David Wagner. Key-Schedule Cryptoanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 1996.
11. Youngdai Ko, Seokhie Hong, Wonil Lee, Sangjin Lee, and Ju-Sung Kang. Related Key Differential Attacks on 27 Rounds of XTEA and Full-Round GOST. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 299–316. Springer, 2004.
12. Florian Mendel, Norbert Pramstaller, and Christian Rechberger. A (Second) Preimage Attack on the GOST Hash Function. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *Lecture Notes in Computer Science*, pages 224–234. Springer, 2008.
13. Florian Mendel, Norbert Pramstaller, Christian Rechberger, Marcin Kontak, and Janusz Szmidt. Cryptanalysis of the GOST Hash Function. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 162–178. Springer, 2008.
14. National Bureau of Standards. Federal Information Processing Standard-Cryptographic Protection - Cryptographic Algorithm. GOST 28147-89, 1989.
15. OpenSSL. A Reference Implementation of GOST. `http://www.openssl.org/source/`.
16. Vladimir Rudskoy. On Zero Practical Significance of Key Recovery Attack on Full GOST Block Cipher with Zero Time and Memory. Cryptology ePrint Archive, Report 2010/111, 2010. `http://eprint.iacr.org/`.
17. Haruki Seki and Toshinobu Kaneko. Differential Cryptanalysis of Reduced Rounds of GOST. In Douglas R. Stinson and Stafford E. Tavares, editors, *Selected Areas in Cryptography*, volume 2012 of *Lecture Notes in Computer Science*, pages 315–323. Springer, 2000.

A Appendix: The Full 4-Round "Guess and Determine" Attack

In this section we give the full details of the the 4-Round "Guess and Determine" attack.

A.1 Optimization Methods Used in the 4-Round Attack

Since the time complexity of the attack is determined by the widest layer of the tree, we use several optimizations in order to obtain effective filtering conditions while guessing the smallest possible number of bits in each layer. These optimizations ensure that the expected number of children per node at a given layer is small, and thus reduce the expected width of the next layer. As a result, we can recover the possible keys that generate the given input-output pairs more efficiently. The optimizations that we use are described below (some of which were already used in the attack on S-GOST in Section 5.3):

1. We optimize the basic process of expanding a node described in Section 5.3 by using a more direct approach which gives the same result. For example, we calculate the values of the 4 intermediate encryption bits, $Z_R^{31,2}$, from the encryption side of the Feistel structure. The consistency predicate on these bits can now be viewed as an equation on 4 bits of K_4 ($K_4^{20,23}$) from the decryption side. In the basic approach we solve this equation by exhaustive search on the 16 possible values of $K_4^{20,23}$. Instead, we precompute and store the solutions to the equation for all its 2^{20} possible values, and use this small precomputed table to directly derive the values of $K_4^{20,23}$ (i.e. we expand only the nodes that satisfy the consistency predicate in advance). This direct approach is more efficient than the basic approach since the size of the layers of the guess tree (including the size of the widest layer) is reduced in exchange for a small amount of precomputation and memory.
2. Given the case of two input-output pairs, we use differential methods in order to simultaneously reduce the number of unknown bits and constraint bits in our equations. For example, refer to step 3 in Section 5.3, where we eliminate the 4 bits of $K_3^{0,3}$ from the predicate. As a result, the size of our precomputed tables can be reduced.
3. To minimize the number of required carry and state bit guesses, we work on consecutive chunks of bits from right to left (i.e., we perform the iterations sequentially as justified in Section 5.4).
4. Initially, we guess values that are required to calculate the four LSBs of several addition operations (i.e., we start the process from iteration 0, as justified in Section 5.4).

A.2 Details and Analysis of the 4-Round Attack

Consider the equations of Section 4.2 for the first pair, and similar equations for the second pair. Let C_i for be the 32-bit carry word produced by the addition of the round key K_i to the corresponding state word (note that $C_i^0 = 0$ and we can ignore the last carry produced at bit 31 which has no effect on the encryption). From each one of these four 32-bit equations, we derive eight equations which equate 4-bit words, and are indexed by $i \in \{0, 1, ..., 7\}$:
(E_1^i): $Z_L^{4i+11,4i+14} = X_L^{4i+11,4i+14} \oplus f^i(X_R^{4i,4i+3} \boxplus K_1^{4i,4i+3} \boxplus C_1^{4i})$
(E_2^i): $Z_R^{4i+11,4i+14} = Y_R^{4i+11,4i+14} \oplus f^i(Y_L^{4i,4i+3} \boxplus K_4^{4i,4i+3} \boxplus C_4^{4i})$
(E_3^i): $Y_L^{4i+11,4i+14} \oplus Z_L^{4i+11,4i+14} = f^i(Z_R^{4i,4i+3} \boxplus K_3^{4i,4i+3} \boxplus C_3^{4i})$
(E_4^i): $X_R^{4i+11,4i+14} \oplus Z_R^{4i+11,4i+14} = f^i(Z_L^{4i,4i+3} \boxplus K_2^{4i,4i+3} \boxplus C_2^{4i})$

In addition to the carry words defined above, we define CS_2 and CS_3 as the 32-bit words $(Z_L \boxminus Z_L^*) \oplus Z_L \oplus Z_L^*$ and $(Z_R \boxminus Z_R^*) \oplus Z_R \oplus Z_R^*$ respectively.

As described in Section 5, our tree contains 9 layers ($\ell_0, \ell_1, ..., \ell_8$). The procedure for expanding the nodes of layer $i \in \{0, 1, ..., 7\}$ uses equations E_1^i, E_1^{i+2}, E_2^{i+5}, E_3^i and E_4^{i+5} (the index additions are performed numerically modulo 8).

Table 3 gives the iteration inputs and outputs calculated in each step of the iteration algorithm for $i \in \{0, 1, ..., 7\}$. Note that the carry and state bits and expressions which are outputs of the iteration i, serve as inputs to iteration $i + 1$.

Step	Key input	Carry input	State input	Key output	Carry output	State output
(1)	$K_1^{4i,4i+3}$	C_1^{4i}, C_1^{*4i}	-	-	C_1^{4i+4}, C_1^{*4i+4}	-
(3)	-	-	-	-	$C_3^{4i+4} \boxminus$ $C_3^{*4i+4} \boxminus$ CS_3^{4i+4}	$(Z_R \boxminus Z_R^*)^{4i+3}$
(4)	-	$C_3^{4i} \boxminus$ $C_3^{*4i} \boxminus$ CS_3^{4i}	$(Z_R \boxminus Z_R^*)^{4i+31}$	-	-	-
(5)	-	C_4^{4i+20}, C_4^{*4i+20}	-	$K_4^{4i+20,4i+23}$	-	-
(6)	-	C_4^{4i+20}, C_4^{*4i+20}	-	-	C_4^{4i+24}, C_4^{*4i+24}	-
(8)	-	-	-	-	$C_2^{4i+24} \boxminus$ $C_2^{*4i+24} \boxminus$ CS_2^{4i+24}	$(Z_L \boxminus Z_L^*)^{4i+23}$
(9)	-	$C_2^{4i+20} \boxminus$ $C_2^{*4i+20} \boxminus$ CS_2^{4i+20}	$(Z_L \boxminus Z_L^*)^{4i+19}$	-	-	-
(10)	-	C_1^{4i+8}, C_1^{*4i+8}	-	$K_1^{4i+8,4i+11}$	C_1^{4i+12}, C_1^{*4i+12}	-

Steps (2) and (7) do not use any iteration input or calculate any iteration output.

Table 3. Iteration inputs used and iteration outputs calculated in each step of the iteration algorithm for $i \in \{0, 1, ..., 7\}$

The steps of iteration $i \in \{0, 1, ..., 7\}$ are given below.[9] Note that steps 6–10 are analogous to steps 1–5, but are performed from the decryption side.

1. Given the inputs $K_1^{4i,4i+3}, C_1^{4i}, C_1^{*4i}$, use equation E_1^i to calculate $Z_L^{4i+11,4i+14}$ for both pairs.
2. Given $Z_L^{4i+11,4i+14}$ (from step (1)), use equation E_3^i to calculate $Z_R^{4i,4i+3} \boxplus K_3^{4i,4i+3} \boxplus C_3^{4i}$ for both pairs.
3. Subtract the expressions calculated in step (2), $Z_R^{4i,4i+3} \boxplus K_3^{4i,4i+3} \boxplus C_3^{4i}$ and $Z_R^{*4i,4i+3} \boxplus K_3^{4i,4i+3} \boxplus C_3^{*4i}$, to eliminate $K_3^{4i,4i+3}$, and obtain the value of $(Z_R^{4i,4i+3} \boxminus Z_R^{*4i,4i+3}) \boxplus (C_3^{4i} \boxminus C_3^{*4i} \boxminus CS_3^{4i})$.
4. Subtract the input $C_3^{4i} \boxminus C_3^{*4i} \boxminus CS_3^{4i}$ from the 3 LSBs of the expression calculated in step (3), and concatenate the 3-bit result with the input $(Z_R \boxminus Z_R^*)^{4i+31}$ to obtain $(Z_R \boxminus Z_R^*)^{4i+31,4i+2}$.
5. Given $(Z_R \boxminus Z_R^*)^{4i+31,4i+2}$ (from step (4)) and the carries $C_4^{4i+20}, C_4^{*4i+20}$, solve the equation obtained by subtracting right hand side of E_2^{i+5} to obtain $K_4^{4i+20,4i+23}$.
6. Given $C_4^{4i+20}, C_4^{*4i+20}$, and $K_4^{4i+20,4i+23}$ (derived in step (5)), use equation E_2^{i+5} to calculate $Z_R^{4i+31,4i+2}$ for both pairs.

[9] For the sake of simplicity, we do not mention the carry and state output bits in the description of the steps, and just list them in Table 3.

7. Given $Z_R^{4i+31,4i+2}$ (from step (6)), use equation E_4^{i+5} to calculate $Z_L^{4i+20,4i+23} \boxplus K_2^{4i+20,4i+23} \boxplus C_2^{4i+20}$ for both pairs.
8. Subtract the expressions calculated in step (7), $Z_L^{4i+20,4i+23} \boxplus K_2^{4i+20,4i+23} \boxplus C_2^{4i+20}$ and $Z_L^{*4i+20,4i+23} \boxplus K_2^{4i+20,4i+23} \boxplus C_2^{*4i+20}$ to eliminate $K_2^{4i+20,4i+23}$, and obtain the value of $(Z_L^{4i+20,4i+23} \boxminus Z_L^{*4i+20,4i+23}) \boxplus (C_2^{4i+20} \boxminus C_2^{*4i+20} \boxminus CS_2^{4i+20})$.
9. Subtract the input $C_2^{4i+20} \boxminus C_2^{*4i+20} \boxminus CS_2^{4i+20}$ from the 3 LSBs of the expression calculated in step (8), and concatenate the 3-bit result with the input $(Z_L \boxminus Z_L^*)^{4i+19}$ to obtain $(Z_L \boxminus Z_L^*)^{4i+19,4i+22}$.
10. Given $(Z_L \boxminus Z_L^*)^{4i+19,4i+22}$ (from step (9)) and the inputs C_1^{4i+8}, C_1^{*4i+8}, solve the equation obtained by subtracting right hand side of E_1^{i+2} to obtain $K_1^{4i+8,4i+11}$.

All the steps of this iteration algorithm involve simple operations on 4-bit words (addition, subtraction, XOR and application of a 4 × 4 Sbox, or its inverse). The exceptional steps are (5) and (10), where we have to solve the equations obtained by subtracting the right hand sides of E_2^{i+5} and E_1^{i+2}, respectively. Each equation adds a 4-bit constraint on 4 unknown bits of the key, and thus we expect a single solution on average. The solutions to each equation can be derived by using the basic approach of exhaustive search over the 2^4 possible values of the 4 key bits. However, we speed up the process for each equation by precomputing and storing the solutions for each of the 2^4 possible values of the equation and for each of the 2^{16} values of the 16 relevant input or output bits that participate in the equation. A table for a single equation has $2^{4+16} = 2^{20}$ entries, where each entry has an average of a single 4-bit solution (2^{22} bits, or 2^{16} words of 64 bits in total per table), and requires a negligible precomputation time compared to the complexity of the full attack on GOST.

We now analyze the expected time complexity of the algorithm by calculating the width of the layers of the tree according to the expected number of guesses required at each stage of the algorithm: in general, iteration i requires the following input bits (as specified in Table 3): 4 bits of K_1 in step (1), 6 single carry bits in steps (1),(5),(6) and (8) (note that steps (5) and (6) require the same carry bits), 4 carry expression bits in steps (4) and (9) (note that the value of each carry expression is either -2,-1,0 or 1) and 2 state bit expressions in steps (4) and (9). Altogether, iteration i requires $4 + 6 + 4 + 2 = 16$ input bits. However, in iteration 0 (which is the first iteration performed), the carry inputs required in step (1) and the carry expression required in step (4) are known to be zero. Thus, iteration 0 requires only 12 unknown iteration input bits which we have to guess, thus the expected size of the second layer is 2^{12}. Note that the inverse Sbox computed in steps (2) and (7) is expected to provide a single output value per input (i.e. step (2) and (7) are not expected to increase the width of the guess tree). In addition, the equations solved in steps (5) and (10), are expected to have a single solution, as explained above.

In iteration 1 (where we derive layer 2 of the tree), iteration inputs which are carry and state bits are already known from the output of iteration 0. Moreover, after step (10) of the first iteration, we know the values of C_1^8 and C_1^{*8}. This gives us a 2-bit filtering condition on $K_1^{4,7}$ (we only try values of $K_1^{4,7}$ which are consistent with the carries). In this sense, the carries guessed in step (10) of the first iteration are "consumed" by the second iteration. Thus, after the first two iterations, we obtain $K_4^{20,27}$ and $K_1^{8,15}$ from guessing 8 bits of the first key, $K_1^{0,7}$. In addition, we have an expected number of $2^{8-2} = 2^6$ additional guesses (counting the carry and state bit guesses of steps (2)–(10) of iteration 0, without the 2-bit guess of step (10)). Thus, the expected size of layer 2 is $2^{8+6} = 2^{14}$, which is larger than the 2^{12} expected size of layer 1, but not by a large factor.

In iteration 2, we derive $K_4^{28,31}$ and $K_1^{16,19}$ from $K_1^{8,11}$. Since $K_1^{8,11}$ is already known at this stage, we do not need to guess it again. Thus, the size of layer 3 remains the same as in layer 2, namely 2^{14} possible solutions. This pattern continues until the end of iteration 5, where our partial guess nodes include the values of $K_1^{0,31}$ and $K_4^{20,11}$ (as shown in Table 2). In iterations 6 and 7, we derive the remaining bits of K_4 ($K_4^{12,19}$) and the bits of K_1 ($K_1^{0,7}$) which were already guessed, and give us additional 4-bit filtering conditions on the guesses in each of these iterations. Thus, layer 7 of the tree in expected to contain $2^{14-4} = 2^{10}$ nodes. Iteration 7 is the final iteration, in which besides the 4-bit filtering condition on $K_1^{4,8}$, we also obtain the remaining 6 iteration inputs guessed in iteration 0. We thus receive additional filtering conditions of 6 bits and expect the final layer to contain $2^{10-4-6} = 1$ node (a single value for K_1 and K_4, as expected when we compare the total number of key and input-output constraint bits).

The expected number of nodes in the widest layer of the partial guess tree is 2^{14}, and it is obtained at iterations 1 to 5 (which define layers 2 to 6 in the tree). Thus, the time complexity of the algorithm is about 2^{14} Feistel structure evaluations for each one of the two input-output pairs, and 2^{15} evaluations altogether. Since we work on a 4-round Feistel structure which contains a fraction of 2^{-3} of the 32 rounds of the full GOST, we estimate that the expected time complexity of this attack is equivalent to about $2^{15-3} = 2^{12}$ GOST evaluations. We apply this 4-round attack for each one of the 2^{128} possible values of the last 4 round keys (K_5, K_6, K_7 and K_8), and thus the time complexity of the 8-round attack is about $2^{128+12} = 2^{140}$ GOST evaluations.

In terms of memory, we store precomputed tables for steps (5) and (10) in each iteration. The equations solved in these two steps are of the same structure for each one of the 8 iterations and differ only according to the Sbox used. Thus, we need 8 such tables (one for each Sbox), which require $8 \cdot 2^{22} = 2^{25}$ bits of memory. The additional memory required to store other intermediate variables and to store our state in the DFS traversal is negligible compared to the space consumed by the precomputed tables. Hence, the attack has a completely practical memory complexity of 2^{25} bits, which is equivalent to 2^{19} 64-bit words.

B Appendix: Parameters for the 2-Dimensional Meet-in-the-middle Attack

In this section, we specify our choices of G_1–G_3 and S_1–S_5:

- G_1 contains the 16 key bits which are derived by both the left batch (iterations 0–3) and the right batch (iterations 4–7), as specified in Table 4.
- G_2 contains the carry and state iteration input bits that we guess in iteration 0, not including step (10) (the bits that we guess in step (10) are already used as filtering conditions in iteration 1). Using Table 3, we get $|G_2| = 6$ (using the fact that the carry bits are known to be zero).
- G_3 contains the carry and state iteration input bits that we guess in iteration 4 (the first iteration of the right batch), not including the bits that we guess in step (10). Using Table 3, we get that $|G_3| = 10$ (unlike iteration 0, in iteration 4 no carry bits and expressions are known in advance).

In order to determine the sets S_1–S_5 we refer to Table 4, which gives the indices of the intermediate encryption bits required by iterations 0–7 of the top part of the 8-round Feistel structure. In order to calculate the indices of these bits, recall from Section A.2 that iteration $i \in \{0, 1, ..., 7\}$ uses equations $E_1^i, E_1^{i+2}, E_2^{i+5}, E_3^i$ and E_4^{i+5}, out of which only E_2^{i+5} and E_3^i require bits of Y and Y^*: E_2^{i+5} requires $Y_R^{4i+31,4i+2}$ and $Y_L^{4i+20,4i+23}$, and E_3^i requires $Y_L^{4i+11,4i+14}$ (note that iteration i also requires the same indices for Y^*). Altogether, iterations 0–3 require the 82 intermediate bits $Y_R^{31,14}$, $Y_L^{11,3}$, $Y_R^{*31,14}$ and $Y_L^{*11,3}$, and iterations 4–7 require the 82 intermediate bits of $Y_R^{15,30}$, $Y_L^{27,19}$, $Y_R^{*15,30}$ and $Y_L^{*27,19}$. After calculating the indices of the intermediate encryption bits that the iteration batches of the top part require, we can easily derive the analogous indices that the iteration batches of the bottom part require, taking into account that the right and left 32-bit halves of Y and Y^* are exchanged at the end of round 4. Thus, we

Iteration	0	1	2	3	4	5	6	7
K_1 bits derived	(0–3) 8–11	(4–7) 12–15	$\underline{8\text{–}11}$ (16–19)	$\underline{12\text{–}15}$ (20–23)	(16–19) 24–27	(20–23) 28–31	$\underline{24\text{–}27}$ (0–3)	$\underline{28\text{–}31}$ (4–7)
K_4 bits derived	20–23	24–27	28–31	0–3	4–7	8–11	12–15	16–19
Bits of Y and Y^* required	$R[31,2]$ $L[11,14]$ $L[20,23]$	$R[3,6]$ $L[15,18]$ $L[24,27]$	$R[7,10]$ $L[19,22]$ $L[28,31]$	$R[11,14]$ $L[23,26]$ $L[0,3]$	$R[15,18]$ $L[27,30]$ $L[4,7]$	$R[19,22]$ $L[31,2]$ $L[8,11]$	$R[23,26]$ $L[3,6]$ $L[12,15]$	$R[27,30]$ $L[7,10]$ $L[15,19]$

Key bits which are known from previous iterations of the batch are underlined. Key bits of G_1 (derived by both of the iteration batches) appear is parenthesis. The bits of Y and Y^* are denoted as follows: $R[i,j]$ denotes $Y_R^{i,j}$ and $Y_R^{*i,j}$, $L[i,j]$ denotes $Y_L^{i,j}$ and $Y_L^{*i,j}$.

Table 4. The key bits derived and the intermediate encryption bits required in each iteration of the left and right batches

need to exchange the right and left halves of the bits calculated for the top part: for the bottom part, the left batch requires the 82 intermediate encryption bit values of $Y_L^{31,14}$,$Y_R^{11,3}$,$Y_L^{*31,14}$ and $Y_R^{*11,3}$ and the right batch requires the 82 bits of $Y_L^{15,30}$,$Y_R^{27,19}$,$Y_L^{*15,30}$ and $Y_R^{*27,19}$.

The sets S_1–S_5 that we choose are given in table 5. Note that since the right and left 32-bit halves of Y and Y^* are exchanged at the end of round 4, we choose S_1 so that it contains the same bit indices from both halves of Y and Y^*. As a result, the sets used during the iteration batches are of the same size ($|S_2| = |S_3| = |S_4| = |S_5| = 18$). This implies that the iteration batches of both the top and the bottom parts are performed the same number of times (2^{18}) for a given value of the 92 bits of S_1.

S_1	$Y_L^{10,19}$, $Y_L^{23,3}$, $Y_R^{10,19}$, $Y_R^{23,3}$, $Y_L^{*10,19}$, $Y_L^{*23,3}$, $Y_R^{*10,19}$, $Y_R^{*23,3}$
S_2	$Y_L^{20,22}$, $Y_R^{4,9}$, $Y_L^{*20,22}$, $Y_R^{*4,9}$
S_3	$Y_L^{4,9}$, $Y_R^{20,22}$, $Y_L^{*4,9}$, $Y_R^{*20,22}$
S_4	$Y_R^{20,22}$, $Y_L^{4,9}$, $Y_R^{*20,22}$, $Y_L^{*4,9}$
S_5	$Y_R^{4,9}$, $Y_L^{20,22}$, $Y_R^{*4,9}$, $Y_L^{*20,22}$

Table 5. The sets S_1–S_5

C Appendix: Exploiting GOST's Complementation Property

The full GOST block cipher has a well-known complementation property. If the plaintext $P = (P_L, P_R)$ is encrypted under $K = (K_1, K_2, \ldots, K_8)$ to the ciphertext $C = (C_1, C_2)$, then the encryption of $P^* = (P_L \oplus e_{31}, P_R \oplus e_{31})$ under $K = (K_1 \oplus e_{31}, K_2 \oplus e_{31}, \ldots, K_8 \oplus e_{31})$ is $C^* = (C_1 \oplus e_{31}, C_2 \oplus e_{31})$ (where e_{31} is the 32-bit vector whose entries are all zero, except the MSB, which is one.).

At the same time, in our attacks on reduced-round GOST, we notice the existence of two less known complementation properties: for $G_{K_1,K_2,K_3,K_4}(P_L, P_R) = (T_L, T_R)$, $G_{K_1 \oplus e_{31},K_2,K_3 \oplus e_{31},K_4}(P_L, P_R \oplus e_{31}) = (T_L, T_R \oplus e_{31})$ and $G_{K_1,K_2 \oplus e_{31},K_3,K_4 \oplus e_{31}}(P_L \oplus e_{31}, P_R) = (T_L \oplus e_{31}, T_R)$.

One can use these three complementation properties in all of our attacks (even though each one of them leads to a different improvement factor). For example, consider the meet-in-the-middle attack suggested in Section 4.2. In this attack, we obtain two 8-round input-output pairs (I, O) and (I^*, O^*). The attack starts by guessing Y (the partial encryption of I after four rounds). The naive way to implement the search loop is to try any possible value of Y, and then any value of K_3, K_4 to obtain the candidate values of K_1, K_2. However, for each guess of Y, I, K_3, K_4, consider the 2^{64} candidates for K_1, K_2. If we consider the list of candidates for $Y \oplus (e_{31}, e_{31}), I \oplus (e_{31}, e_{31}), K_3 \oplus e_{31}, K_4 \oplus e_{31}$, it is the same as the previous one (up to the MSBs of K_1 and K_2). The same is true for the other two complementation properties.

In other words, instead of computing the three additional lists (for each of the three complementation properties) we can perform this step only once. As there are four 4-round steps (we need to deal with (I, Y), (Y, O), (I^*, Y^*) and (Y^*, O^*)), we can save three out of the 16 4-round steps (i.e., for each I, $I \oplus (0, e_{31})$, $I \oplus (e_{31}, 0)$ and $I \oplus (e_{31}, e_{31})$ with all the corresponding Y's we compute the list only once).

We note that in the attacks based on the fix point point property, the first input-output pair is actually (I, I), hence, one can use the complementation property again (once for $(I, I \oplus (e_{31}, e_{31}))$ and once for $(I \oplus (0, e_{31}), I \oplus (e_{31}, 0))$. Additionally, as O^* is I (up to a swap), one can again save two out of the four rounds computations. In total, this improvement results in an overall saving of 7/16 in the 8-round attack.

In the unoptimized fixed-point attack there are 2^{192} steps of full-GOST trial encryptions, and 2^{192} executions of the 8-round attack, which result in a total time complexity equivalent to $(32 + 16) \cdot 2^{192} = 48 \cdot 2^{192}$ rounds of GOST. Using this improvement, the total running time is reduced to $(32 + 9) \cdot 2^{192} = 41 \cdot 2^{192}$ rounds of GOST, a speed up of about 14.6%.

In the reflection-based attacks one can optimize the trial encryptions: instead of performing 2^{224} full-GOST trial encryptions, it is possible to exploit the additional "half pair" and obtain an additional 32-bit filtering condition by running 8 rounds of GOST. As a result, the trial encryptions require less than 2^{224} full-GOST evaluations, while the 8-round attacks take more than that. Thus, unlike the fixed-point-based attacks, in the reflection-based attacks the 8-round attacks form the bottleneck, and reducing their complexity gives a more significant savings. We note that the complex attack procedure of Section 6 can also be improved by changing the order of the loop. To do so, one needs to reorder the guess of X, and Y accordingly. Therefore, using a chosen plaintext model for the reflection-based attacks (to obtain 2^{32} appropriate plaintext-ciphertext pairs), it is possible to perform the analysis for three out of the four 4-round phases only once. This reduces the running time to 7/16 of the original time complexity. The total running time of the improved attack is thus reduced to $2^{222.8}$ applications of the 8-round attack.

Zero Correlation Linear Cryptanalysis with Reduced Data Complexity

Andrey Bogdanov[1⋆] and Meiqin Wang[1,2⋆]

[1] KU Leuven, ESAT/COSIC and IBBT, Belgium
[2] Shandong University, Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan 250100,China

Abstract. Zero correlation linear cryptanalysis is a novel key recovery technique for block ciphers proposed in [5]. It is based on linear approximations with probability of exactly 1/2 (which corresponds to the zero correlation). Some block ciphers turn out to have multiple linear approximations with correlation zero for each key over a considerable number of rounds. Zero correlation linear cryptanalysis is the counterpart of impossible differential cryptanalysis in the domain of linear cryptanalysis, though having many technical distinctions and sometimes resulting in stronger attacks.

In this paper, we propose a statistical technique to significantly reduce the data complexity using the high number of zero correlation linear approximations available. We also identify zero correlation linear approximations for 14 and 15 rounds of TEA and XTEA. Those result in key-recovery attacks for 21-round TEA and 25-round XTEA, while requiring less data than the full code book. In the single secret key setting, these are structural attacks breaking the highest number of rounds for both ciphers.

The findings of this paper demonstrate that the prohibitive data complexity requirements are not inherent in the zero correlation linear cryptanalysis and can be overcome. Moreover, our results suggest that zero correlation linear cryptanalysis can actually break more rounds than the best known impossible differential cryptanalysis does for relevant block ciphers. This might make a security re-evaluation of some ciphers necessary in the view of the new attack.

Keywords: block ciphers, key recovery, linear cryptanalysis, zero correlation linear cryptanalysis, data complexity, TEA, XTEA

1 Introduction

1.1 Motivation

Differential and linear cryptanalyses [3, 30] are the two basic tools for evaluating the security of block ciphers such as the former U.S. encryption standard DES as well as its successor AES. While DES was developed at the time when differential and linear cryptanalyses were not publicly known, the design of AES provably addresses these attacks.

Design strategies have been proposed such as the wide-trail design strategy [13] or decorrelation theory [42] to make ciphers resistant to the basic flavours of differential and linear cryptanalysis. However, a proof of resistance according to these strategies does not necessarily imply resistance to the extensions of these techniques such as impossible differential cryptanalysis [1, 6] and the recently proposed zero correlation linear cryptanalysis [5].

Standard differential cryptanalysis uses differentials with probabilities significantly higher than those expected for a randomly drawn permutation. Similarly, basic linear cryptanalysis uses linear approximations whose probabilities detectably deviate from 1/2. At the same time, impossible differential cryptanalysis and zero correlation linear cryptanalysis are based on structural deviations of another kind: Differentials with zero probability are targeted in impossible differential cryptanalysis and linear approximations with probability of exactly 1/2 correlation are exploited in zero correlation linear cryptanalysis. Thus, zero correlation linear cryptanalysis

[⋆] Both authors are corresponding authors.

can be seen as the counterpart of impossible differential cryptanalysis in the domain of linear cryptanalysis.

The name of the attack originated from the notion of *correlation* [11,34]: If $\frac{1+c}{2}$ is the probability for a linear approximation to hold, c is called the correlation of this linear approximation. Clearly, putting $c = 0$ yields an unbiased linear approximation of probability 1/2, or a *zero correlation linear approximation.*

Impossible differential cryptanalysis has been known to the cryptographic community since over a decade now. It has turned out a highly useful tool of attacking block ciphers [2,15,27–29, 41]. In fact, among meet-in-the-middle [14] and multiset-type attacks [18], it is the impossible differential cryptanalysis [28] that breaks the highest numbers of rounds of AES-128 and AES-256 in the classical single-key attack model as to date, the recent biclique cryptanalysis [4] being the notable exception though.

Zero correlation linear cryptanalysis is a novel promising attack technique that bears some technical similarities to impossible differential cryptanalysis but has its theoretical foundation in a different mathematical theory. Despite its newness, it has already been demonstrated to successfully apply to round-reduced AES and CLEFIA even in its basic form [5], which is highly motivating for further studies.

In this paper, we show how to remove the data requirement of the full codebook which was the major limitation of basic zero correlation linear cryptanalysis [5]. As an application of zero correlation linear cryptanalysis and this data complexity reduction technique, we propose attacks against round-reduced TEA and XTEA. For both ciphers, we can cryptanalyze more rounds than it was previously possible using less than the full code book.

1.2 Contributions

The work at hand has two major contributions.

Data complexity reduction for zero correlation linear cryptanalysis. The data requirements of the full codebook have been a crucial limitation for the recent zero correlation linear cryptanalysis to become a major cryptanalytic technique, though the length of the fundamental property (the length of the zero correlation linear approximation) was demonstrated to be comparable to that of impossible differentials for several cipher structures [5]. Overcoming this annoying limitation, a statistical technique of data complexity reduction for zero correlation linear cryptanalysis is the first contribution of this paper.

The data complexity reduction technique is based on the fact that, like any exploitable impossible differential, a typical zero correlation linear approximation is *truncated*: That is, once a zero correlation linear approximation has been identified that holds for all keys, it will as a rule imply an entire class of similar zero correlation linear approximations to exist. Those can be typically obtained by just changing several bits of the input mask, output mask or both. In other words, in most practical cases, there will be *multiple* zero correlation linear approximations available to the adversary which has been ignored by the previous analysis.

However, unlike in impossible differential cryptanalysis, the actual value of the correlation has to be estimated in zero correlation linear cryptanalysis and it is not enough to just wait for the impossible event to occur. In fact, the idea we use for zero correlation linear cryptanalysis is more similar to that of multiple linear cryptanalysis: We estimate the correlation of each individual linear approximation using a limited number of texts. Then, for a group of zero correlation linear approximations (i.e. for the right key), we expect the cumulative deviation of those estimations from 0 to be lower than that for a group of randomly chosen linear approximations (i.e. for a wrong key). Given the statistical behaviour of correlation for a randomly drawn permutation [12, 35], this consideration results in a χ^2 statistic and allows for a theoretical analysis of the complexity and error probabilities of a zero correlation linear attack that are confirmed by experiments.

Table 1. Summary of cryptanalytic results on round-reduced TEA* and XTEA in the single-key setting

attack	#rounds	data	comp. compl.	memory	Pr[success]	ref.
TEA						
impossible differential	11	$2^{52.5}$ CP	2^{84}	NA	NA	[32]
truncated differential	17	1920 CP	$2^{123.37}$	NA	NA	[20]
impossible differential	17	2^{57} CP	$2^{106.6}$	2^{49}	NA	[8]
zero correlation linear	**21**	$\mathbf{2^{62.62}}$ **KP**	$\mathbf{2^{121.52}}$	**negligible**	**0.846**	**this paper**
zero correlation linear	**23**	$\mathbf{2^{64}}$	$\mathbf{2^{119.64}}$	**negligible**	**1**	**this paper**
XTEA						
impossible differential	14	$2^{62.5}$ CP	2^{85}	NA	NA	[32]
truncated differential	23	$2^{20.55}$ CP	$2^{120.65}$	NA	0.969	[20]
meet-in-the-middle	23	18 KP	2^{117}		$1 - 2^{-1025}$	[37]
impossible differential	23	$2^{62.3}$ CP	$2^{114.9}$	$2^{94.3}$	NA	[8]
impossible differential	23	2^{63}	2^{101} MA $+2^{105.6}$	2^{103}	NA	[8]
zero correlation linear	**25**	$\mathbf{2^{62.62}}$ **KP**	$\mathbf{2^{124.53}}$	$\mathbf{2^{30}}$	**0.846**	**this paper**
zero correlation linear	**27**	$\mathbf{2^{64}}$	$\mathbf{2^{120.71}}$	**negligible**	**1**	**this paper**

CP: Chosen Plaintexts, KP: Known Plaintexts.
Memory: the number of 32-bit words.
*The effective key length for TEA is 126 bit

Zero correlation linear cryptanalysis of round-reduced TEA and XTEA. TEA (Tiny Encryption Algorithm) is one of the first lightweight block ciphers. It is a 64-bit block cipher based on a balanced Feistel-type network with a simple ARX round function. TEA has 64 rounds and accepts a key of 128 bits. It favours both efficient hardware [22] and software implementations. TEA was designed by Wheeler and Needham and proposed at FSE'94 [43]. It was used in Microsoft's Xbox gaming console for checking software authenticity until its weakness as a hash function was used [40] to compromise the chain of trust. The block cipher XTEA [33] is the fixed version of TEA eliminating this property (having the same number rounds, block size, and key size). TEA and XTEA being rather popular ciphers, both are implemented in the Linux kernel.

Similarly to the complementation property of DES, TEA has an equivalent key property and its effective key size is 126 bits (compared to 128 bits suggested by the nominal key input size) [23]. Kelsey, Scheier and Wagner [24] proposed a practical related-key attack on the full TEA. Using complementation cryptanalysis [7], up to 36 rounds of XTEA can be attacked with related keys for all keys. The work [7] also contains related-key attacks for up to 50 rounds of XTEA working for a weak key class.

In the classical single-key setting, however, by far not all rounds of TEA are broken by structural attacks (whereas the effective key size is 126 bits for the full cipher). The truncated differential result on 17 rounds remains the best cryptanalysis of TEA [20]. Impossible differential cryptanalysis [8] has yielded a faster attack against 17 rounds of TEA. Similarly, 23 rounds of XTEA have been cryptanalyzed so far using truncated differential [20], impossible differential [8] and well as meet-in-the-middle attacks [37]. That is, for both TEA and XTEA, there has been no progress in terms of the number of attacked rounds since 2003.

In this paper, using zero correlation linear cryptanalysis, we cryptanalyze 21 rounds of TEA and 25 rounds of XTEA with $2^{62.62}$ *known* plaintexts (in contrast to *chosen* texts required in impossible differential cryptanalysis). Certainly, zero correlation linear cryptanalysis for lower number of rounds yields a lower data complexity for both TEA and XTEA. Moreover, unlike most impossible differential attacks including those on TEA and XTEA [8], zero correlation linear cryptanalysis is able to profit from the full code available. If all 2^{64} texts are available to the adversary, we propose zero correlation linear cryptanalysis for 23 rounds of TEA and 27 rounds of XTEA. Our cryptanalytic results are summarized and compared to previous cryptanalysis in Table 1.

As opposed to the initial intuition expressed in [5], both major contributions of this work — the data complexity reduction and the new attacks on more rounds of TEA and XTEA — demonstrate that zero correlation linear cryptanalysis can actually perform better than impossible differential cryptanalysis. Moreover, we expect the security of more ciphers to be reevaluated under the consideration of zero correlation linear cryptanalysis.

1.3 Outline

We start with a review of the basic zero correlation linear cryptanalysis for block ciphers in Section 2. In Section 3, we introduce a χ^2 statistical technique for reducing the data requirements of zero correlation linear cryptanalysis and thoroughly investigate its complexity. In Section 4, the 14- and 15-round zero correlation linear approximations are demonstrated for block ciphers TEA and XTEA. Section 5 gives several zero correlation key recoveries for round-reduced TEA and XTEA. We conclude in Section 6. Appendices contain proofs of some technical statements as well as further zero correlation linear attacks on round-reduced TEA and XTEA.

2 Basic zero correlation linear cryptanalysis

Zero correlation linear cryptanalysis has been introduced in [5]. Below we briefly review its basic ideas and methods.

2.1 Linear approximations with correlation zero

Consider an n-bit block cipher f_K with key K. Let P denote a plaintext which is mapped to ciphertext C under key K, $C = f_K(P)$. If Γ_P and Γ_C are nonzero plaintext and ciphertext linear masks of n bit each, we denote by $\Gamma_P \to \Gamma_C$ the linear approximation

$$\Gamma_P^T P \oplus \Gamma_C^T C = 0.$$

Here, $\Gamma_A^T A$ denotes the multiplication of the transposed bit vector Γ_A (linear mask for A) by a column bit vector A over $\mathbb{F}_2$. The linear approximation $\Gamma_P \to \Gamma_C$ has probability

$$p_{\Gamma_P,\Gamma_C} = \Pr_{P \in \mathbb{F}_2^n}\{\Gamma_P^T P \oplus \Gamma_C^T C = 0\}. \tag{1}$$

The value

$$c_{\Gamma_P,\Gamma_C} = 2p_{\Gamma_P,\Gamma_C} - 1 \tag{2}$$

is called the *correlation (or bias)* of linear approximation $\Gamma_P \to \Gamma_C$. Note that $p_{\Gamma_P,\Gamma_C} = 1/2$ is equivalent to *zero correlation* $c_{\Gamma_P,\Gamma_C} = 0$:

$$p_{\Gamma_P,\Gamma_C} = \Pr_{P \in \mathbb{F}_2^n}\{\Gamma_P^T P \oplus \Gamma_C^T C = 0\} = 1/2. \tag{3}$$

In fact, for a randomly drawn permutation of sufficiently large bit size n, zero is the most frequent single value of correlation for a nontrivial linear approximation. Correlation goes to small values for increasing n, the probability to get exactly zero decreases as a function of n though. More precisely, the probability of the linear approximation $\Gamma_P \to \Gamma_C$ with $\Gamma_P, \Gamma_C \neq 0$ to have zero correlation has been shown [5, Proposition 2] to be approximated by

$$\frac{1}{\sqrt{2\pi}} 2^{\frac{4-n}{2}}. \tag{4}$$

2.2 Two examples

Given a randomly chosen permutation, however, it is hard to tell a priori which of its nontrivial linear approximations in particular has zero correlation. At the same time, it is often possible to identify groups of zero correlation linear approximations for a block cipher f_K once it has compact description with a distinct structure. Moreover, in many interesting cases, these linear approximations will have zero correlation *for any key* K. Here are two examples provided in [5]:

- **AES:** The data transform of AES has a set of zero correlation linear approximations over 4 rounds (3 full rounds appended by 1 incomplete rounds with MixColumns omitted). If Γ and Γ' are 4-byte column linear masks with exactly one nonzero byte, then each of the linear approximations $(\Gamma, 0, 0, 0) \rightarrow (\Gamma', 0, 0, 0)$ over 4 AES rounds has zero correlation [5, Theorem 2].
- **CLEFIA-type GFNs:** CLEFIA-type generalized Feistel networks [39] (also known as type-2 GFNs with 4 lines [44]) have zero correlation linear approximations over 9 rounds, if the underlying F-functions of the Feistel construction are invertible. For $a \neq 0$, the linear approximations $(a, 0, 0, 0) \rightarrow (0, 0, 0, a)$ and $(0, 0, a, 0) \rightarrow (0, a, 0, 0)$ over 9 rounds have zero correlation [5, Theorem 1].

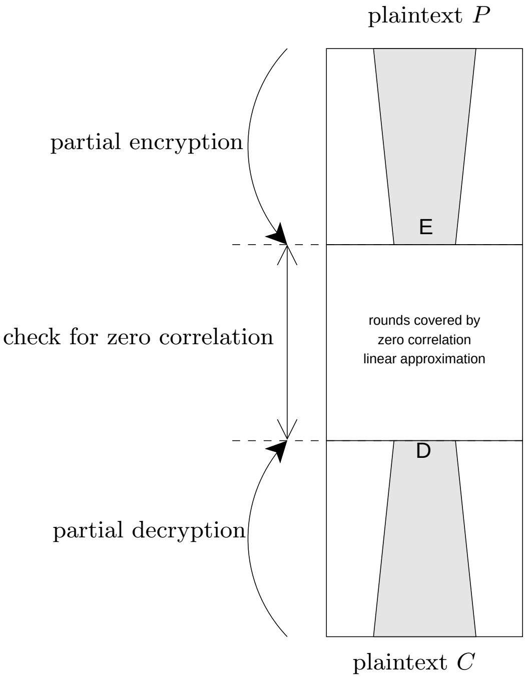


Fig. 1. High-level view of key recovery in zero correlation linear cryptanalysis

2.3 Key recovery with zero correlation linear approximations

Based on linear approximations of correlation zero, a technique similar to Matsui's Algorithm 2 [30] can be used for key recovery. Let the adversary have N known plaintext-ciphertexts and ℓ zero correlation linear approximations $\{\Gamma_E \rightarrow \Gamma_D\}$ for a part of the cipher, with $\ell = |\{\Gamma_E \rightarrow \Gamma_D\}|$. The linear approximations $\{\Gamma_E \rightarrow \Gamma_D\}$ are placed in the middle of the attacked cipher. Let E and D be the partial intermediate states of the data transform at the boundaries of the linear approximations.

Then the key can be recovered using the following approach (see also Figure 1):

1. Guess the bits of the key needed to compute E and D. For each guess:

(a) Partially encrypt the plaintexts and partially decrypt the ciphertexts up to the boundaries of the zero correlation linear approximation $\Gamma_E \to \Gamma_D$.

(b) Estimate the correlations $\{\hat{c}_{\Gamma_E,\Gamma_D}\}$ of all linear approximations in $\{\Gamma_E \to \Gamma_D\}$ for the key guess using the partially encrypted and decrypted values E and D by counting how many times $\Gamma_E^T E \oplus \Gamma_D^T D$ is zero over N input/output pairs, see (1) and (2).

(c) Perform a test on the estimated correlations $\{\hat{c}_{\Gamma_E,\Gamma_D}\}$ for $\{\Gamma_E \to \Gamma_D\}$ to tell of the estimated values of $\{\hat{c}_{\Gamma_E,\Gamma_D}\}$ are compatible with the hypothesis that all of the actual values of $\{c_{\Gamma_E,\Gamma_D}\}$ are zero.

2. Test the surviving key candidates against a necessary number of plaintext-ciphertext pairs according to the unicity distance for the attacked cipher.

Step 1(c) of the technique above relies on an efficient test distinguishing between the hypothesis that $\{c_{\Gamma_E,\Gamma_D}\}$ are all zero and the alternative hypothesis. The work [5] requires the exact evaluation of the correlation value (defined by the probability of a linear approximation) and the data complexity is restricted to $N = 2^n$ in [5]. Thus, a small number ℓ of linear approximations is usually enough in [5] and $\hat{c}_{\Gamma_E,\Gamma_D} = c_{\Gamma_E,\Gamma_D}$, though the data complexity of the full codebook is too restrictive.

For most ciphers (including the examples of Subsection 2.2), however, a large number ℓ of zero correlation linear approximations is available. This freedom is not used in [5]. At the same time, it has been shown in the experimental work [9] that any value of correlation can be used for key recovery in a linear attack with reduced data complexity, once enough linear approximations are available. Despite its convincing experimental evidence, [9] gives no theoretical data complexity estimations and does not provide any ways of constructing linear approximations with certain properties.

In the next section of this paper, we provide a framework for reducing the data complexity N if many zero correlation linear approximations are known.

3 Reduction of data complexity with many approximations

3.1 Distinguishing between two normal distributions

Consider two normal distributions: $\mathcal{N}(\mu_0, \sigma_0)$ with mean μ_0 and standard deviation σ_0, and $\mathcal{N}(\mu_1, \sigma_1)$ with mean μ_1 and standard deviation σ_1. A sample s is drawn from either $\mathcal{N}(\mu_0, \sigma_0)$ or $\mathcal{N}(\mu_1, \sigma_1)$. It has to be decided if this sample is from $\mathcal{N}(\mu_0, \sigma_0)$ or from $\mathcal{N}(\mu_1, \sigma_1)$. The test is performed by comparing the value s to some threshold value t. Without loss of generality, assume that $\mu_0 < \mu_1$. If $s \le t$, the test returns "$s \in \mathcal{N}(\mu_0, \sigma_0)$". Otherwise, if $s > t$, the test returns "$s \in \mathcal{N}(\mu_1, \sigma_1)$". There will be error probabilities of two types:

$$\beta_0 = \Pr\{\text{"}s \in \mathcal{N}(\mu_1, \sigma_1)\text{"} | s \in \mathcal{N}(\mu_0, \sigma_0)\},$$
$$\beta_1 = \Pr\{\text{"}s \in \mathcal{N}(\mu_0, \sigma_0)\text{"} | s \in \mathcal{N}(\mu_1, \sigma_1)\}.$$

Here a condition is given on μ_0, μ_1, σ_0, and σ_1 such that the error probabilities are β_0 and β_1. The proof immediately follows from the basics of probability theory (see e.g. [17, 19]) and is given in Appendix A for completeness.

Proposition 1. *For the test to have error probabilities of at most β_0 and β_1, the parameters of the normal distributions $\mathcal{N}(\mu_0, \sigma_0)$ and $\mathcal{N}(\mu_1, \sigma_1)$ with $\mu_0 \ne \mu_1$ have to be such that*

$$\frac{z_{1-\beta_1}\sigma_1 + z_{1-\beta_0}\sigma_0}{|\mu_1 - \mu_0|} = 1,$$

where $z_{1-\beta_1}$ and $z_{1-\beta_0}$ are the quantiles of the standard normal distribution.

3.2 A known plaintext distinguisher with many zero correlation linear approximations

Let the adversary be given N known plaintext-ciphertext pairs and ℓ zero correlation linear approximations for an n-bit block cipher. The adversary aims to distinguish between this cipher and a randomly drawn permutation.

The procedure is as follows. For each of the ℓ given linear approximations, the adversary computes the number T_i of times the linear approximations are fulfilled on N plaintexts, $i \in \{1, \ldots, \ell\}$. Each T_i suggests an empirical correlation value $\hat{c}_i = 2\frac{T_i}{N} - 1$. Then, the adversary evaluates the statistic:

$$\sum_{i=1}^{\ell} \hat{c}_i^2 = \sum_{i=1}^{\ell} \left(2\frac{T_i}{N} - 1\right)^2. \tag{5}$$

It is expected that for the cipher with ℓ known zero correlation linear approximations, the value of statistic (5) will be lower than that for ℓ linear approximations of a randomly drawn permutation. In a key-recovery setting, the right key will result in statistic (5) being among the lowest values for all candidate keys if ℓ is high enough. In the sequel, we treat this more formally.

3.3 Correlation under right and wrong keys

Consider the key recovery procedure outlined in Subsection 2.3 given N known plaintext-ciphertext pairs. There will be two cases:

- *Right key guess:* Each of the values $\hat{c}_i$ in (5) approximately follows the normal distribution with zero mean and standard deviation $1/\sqrt{N}$ with good precision (c.f. e.g. [21, 38]) for sufficiently large N:

  $$\hat{c}_i \sim \mathcal{N}(0, 1/\sqrt{N}).$$

- *Wrong key guess:* Each of the values $\hat{c}_i$ in (5) approximately follows the normal distribution with mean c_i and standard deviation $1/\sqrt{N}$ for sufficiently large N:

  $$\hat{c}_i \sim \mathcal{N}(c_i, 1/\sqrt{N}) \text{ with } c_i \sim \mathcal{N}(0, 2^{-n/2}),$$

  where c_i is the exact value of the correlation which is itself distributed as $\mathcal{N}(0, 2^{-n/2})$ over random permutations with $n \geq 5$ — a result due to [12,35]. Thus, our wrong key hypothesis is that for each wrong key, the adversary obtains a permutation with linear properties close to those of a randomly chosen permutation.

3.4 Distribution of the statistic

Based on these distributions of $\hat{c}_i$, we now derive the distributions of statistic (5) in these two cases.

Right key guess. In this case, we deal with ℓ zero correlation linear approximations:

$$\sum_{i=1}^{\ell} \hat{c}_i^2 \sim \sum_{i=1}^{\ell} \mathcal{N}^2\left(0, 1/\sqrt{N}\right) = \frac{1}{N} \sum_{i=1}^{\ell} \mathcal{N}^2(0,1) = \frac{1}{N}\chi_\ell^2,$$

where χ_ℓ^2 is the χ^2-distribution with ℓ degrees of freedom which has mean ℓ and standard deviation $\sqrt{2\ell}$, assuming the independency of underlying distributions. For sufficiently large ℓ, χ_ℓ^2 converges to the normal distribution. That is, χ_ℓ^2 approximately follows $\mathcal{N}(\ell, \sqrt{2\ell})$, and:

$$\sum_{i=1}^{\ell} \hat{c}_i^2 \sim \frac{1}{N}\chi_\ell^2 \approx \frac{1}{N}\mathcal{N}\left(\ell, \sqrt{2\ell}\right) = \mathcal{N}\left(\frac{\ell}{N}, \frac{\sqrt{2\ell}}{N}\right). \tag{6}$$

Proposition 2. *Consider ℓ nontrivial zero correlation linear approximations for a block cipher with a fixed key. If N is the number of known plaintext-ciphertext pairs, T_i is the number of times such a linear approximation is fulfilled for $i \in \{1, \ldots, \ell\}$, and ℓ is high enough, then, assuming the counters T_i are independent, the following approximate distribution holds for sufficiently large N and n:*

$$\sum_{i=1}^{\ell} \left(2\frac{T_i}{N} - 1\right)^2 \sim \mathcal{N}\left(\frac{\ell}{N}, \frac{\sqrt{2\ell}}{N}\right).$$

Wrong key guess. The wrong key hypothesis is that we deal with pick a permutation at random for each wrong key. Therefore, the ℓ given linear approximations will have randomly drawn correlations, under this hypothesis. Thus, as mentioned above:

$$\sum_{i=1}^{\ell} \hat{c}_i^2 \sim \sum_{i=1}^{\ell} \mathcal{N}^2\left(c_i, 1/\sqrt{N}\right), \text{ where } c_i \sim \mathcal{N}\left(0, 2^{-n/2}\right).$$

First, we show that the underlying distribution of $\hat{c}_i$ is actually normal with mean 0. Then we show that the sum approximately follows χ^2-distribution assuming the independency of underlying distributions, and can be approximated by another normal distribution.

Since

$$\begin{aligned} \mathcal{N}\left(c_i, 1/\sqrt{N}\right) &= c_i + \mathcal{N}\left(0, 1/\sqrt{N}\right) \\ &= \mathcal{N}\left(0, 1/\sqrt{2^n}\right) + \mathcal{N}\left(0, 1/\sqrt{N}\right) \\ &= \mathcal{N}\left(0, \sqrt{1/N + 1/2^n}\right), \end{aligned}$$

the distribution above is a χ^2-distribution with ℓ degrees of freedom up to a factor, under the independency assumption:

$$\begin{aligned} \textstyle\sum_{i=1}^{\ell} \mathcal{N}^2\left(c_i, 1/\sqrt{N}\right) &= \textstyle\sum_{i=1}^{\ell} \mathcal{N}^2\left(0, \sqrt{\frac{1}{N} + \frac{1}{2^n}}\right) \\ &= \left(\tfrac{1}{N} + \tfrac{1}{2^n}\right) \textstyle\sum_{i=1}^{\ell} \mathcal{N}^2(0, 1) \\ &= \left(\tfrac{1}{N} + \tfrac{1}{2^n}\right) \chi_\ell^2. \end{aligned}$$

As for the right keys, for sufficiently large ℓ, χ_ℓ^2 can be approximated by the normal distribution with mean ℓ and standard deviation $\sqrt{2\ell}$. Thus:

$$\begin{aligned} \textstyle\sum_{i=1}^{\ell} \hat{c}_i^2 \sim \left(\tfrac{1}{N} + \tfrac{1}{2^n}\right) \chi_\ell^2 &\approx \left(\tfrac{1}{N} + \tfrac{1}{2^n}\right) \mathcal{N}\left(\ell, \sqrt{2\ell}\right) \\ &= \mathcal{N}\left(\tfrac{\ell}{N} + \tfrac{\ell}{2^n}, \tfrac{\sqrt{2\ell}}{N} + \tfrac{\sqrt{2\ell}}{2^n}\right). \end{aligned}$$

Proposition 3. *Consider ℓ nontrivial linear approximations for a randomly drawn permutation. If N is the number of known plaintext-ciphertext pairs, T_i is the number of times a linear approximation is fulfilled for $i \in \{1, \ldots, \ell\}$, and ℓ is high enough, then, assuming the independency of T_i, the following approximate distribution holds for sufficiently large N and n:*

$$\sum_{i=1}^{\ell} \left(2\frac{T_i}{N} - 1\right)^2 \sim \mathcal{N}\left(\frac{\ell}{N} + \frac{\ell}{2^n}, \frac{\sqrt{2\ell}}{N} + \frac{\sqrt{2\ell}}{2^n}\right).$$

3.5 Data complexity of the distinguisher

Combining Propositions 2 and 3 with Proposition 1, one obtains the condition:

$$\frac{z_{1-\beta_1}\left(\frac{\sqrt{2\ell}}{N} + \frac{\sqrt{2\ell}}{2^n}\right) + z_{1-\beta_0}\frac{\sqrt{2\ell}}{N}}{\left(\frac{\ell}{N} + \frac{\ell}{2^n}\right) - \frac{\ell}{N}} = 1.$$

The left part of this equation can be simplified to

$$\frac{2^{n+0.5}}{N\sqrt{\ell}}\left(z_{1-\beta_0}+z_{1-\beta_1}\right)+\frac{z_{1-\beta_1}\sqrt{2}}{\sqrt{\ell}},$$

which yields

Theorem 1. *With the assumptions of Propositions 1 to 3, using ℓ nontrivial zero correlation linear approximations, to distinguish between a wrong key and a right key with probability β_1 of false positives and probability β_0 of false negatives, a number N of known plaintext-ciphertext pairs is sufficient if the following condition is fulfilled:*

$$\frac{2^{n+0.5}}{N\sqrt{\ell}}\left(z_{1-\beta_0}+z_{1-\beta_1}\right)+\frac{z_{1-\beta_1}\sqrt{2}}{\sqrt{\ell}}=1.$$

The success probability of an attack is defined by the probability β_0 of false negatives. The probability β_1 of false positives determines the number of surviving key candidates and, thus, influences the computational complexity of the key recovery.

4 Linear approximations with correlation zero for TEA and XTEA

In [5], a sufficient condition is given for a linear approximation to have a correlation of zero. Namely, if for a linear approximation there exist no linear characteristics with non-zero correlation contributions, then the correlation of the linear approximation is exactly zero.

4.1 The block ciphers TEA and XTEA

TEA is a 64-round iterated block cipher with 64-bit block size and 128-bit key which consist of four 32-bit words $K[0], K[1], K[2]$ and $K[3]$. TEA does not have any iterative key schedule algorithm. Instead, the key words are used directly in round functions. The round constant is derived from the constant $\delta = 9e3779b9_x$ and the round number. We denote the input and the output of the r-th round for $1 \leq r \leq 64$ by (L_r, R_r) and (L_{r+1}, R_{r+1}), respectively. $L_{r+1} = R_r$ and R_{r+1} is computed as follows:

$$R_{r+1}=\begin{cases} L_r + (((R_r \ll 4) + K[0]) \oplus (R_r + i\cdot\delta) \oplus (R_r \gg 5 + K[1])) & r = 2i-1, 1 \leq i \leq 32,\\ L_r + (((R_r \ll 4) + K[2]) \oplus (R_r + i\cdot\delta) \oplus (R_r \gg 5 + K[3])) & r = 2i, 1 \leq i \leq 32.\end{cases}$$

Like TEA, XTEA is also a 64-round Feistel cipher with 64-bit block and 128-bit key. Its 128-bit secret key K is represented by four 32-bit words $K[0], K[1], K[2]$ and $K[3]$ as well. The derivation of the subkey word number is slightly more complex though. The input of the r-th round is (L_r, R_r) and the output is (L_{r+1}, R_{r+1}). Again, $L_{r+1} = R_r$ and R_{r+1} is derived as:

$$R_{r+1}=\begin{cases} L_r + (((R_r \ll 4 \oplus R_r \gg 5) + R_r) \oplus ((i-1)\cdot\delta + K[((i-1)\cdot\delta \ll 11)\&3])) & r = 2i-1, 1 \leq i \leq 32,\\ L_r + (((R_r \ll 4 \oplus R_r \gg 5) + R_r) \oplus (i\cdot\delta + K[(i\cdot\delta \ll 11)\&3])) & r = 2i, 1 \leq i \leq 32.\end{cases}$$

These round functions of TEA and XTEA are illustrated in Figure 2.

4.2 Notations

To demonstrate zero correlation linear approximations for TEA and XTEA, we will need the following notations (the least significant bit of a word has number 0):

- $e_{i,\sim}$ is a 32-bit word that has zeros in bits $(i+1)$ to 31, one in bit i and undefined values in bits 0 to $(i-1)$,

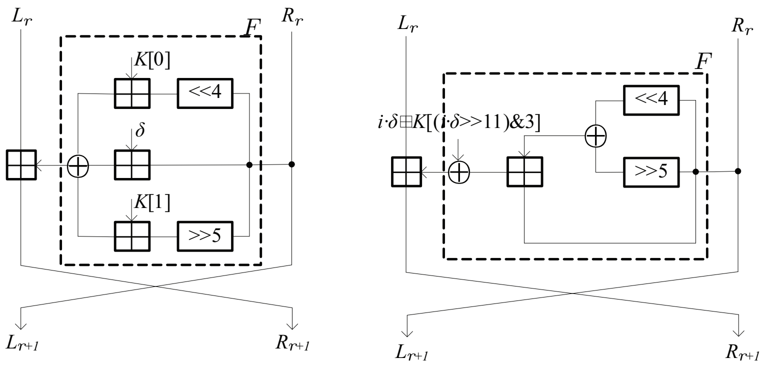


Fig. 2. Round function for TEA(left) and XTEA(right)

- $e_{i\sim j}$ is a 32-bit word that has zeros in bits $(i+1)$ to 31 and bits 0 to $(j-1)$, a one in bit i and undefined values in bits j to $(i-1)$ for $j < i$,
- $\bar{e}_{i,\sim}$ is a 32-bit word that has zeros in bits $(i+1)$ to 31, undefined values in bits 0 to i,
- ? is an undefined value,
- $X^{i\sim j}$ is bits from j to i of the value X, $j < i$, and
- X^i is the value of bit i of X.

4.3 Linear approximation of modular addition

Here, we first demonstrate the properties of linear approximations with non-zero correlation over the modular addition, which is the only nonlinear part of the TEA and XTEA transformation (summarized as Property 1). Then we use it to show a condition for linear approximation with non-zero correlation for one round of TEA and XTEA (stated as Property 2).

For the modular addition of two n-bit inputs x and y, the output z can be computed as:

$$z = (x + y) \mod 2^n.$$

We denote the mask values for x, y and z as $\Gamma x, \Gamma y$ and Γz, respectively ($x, y, z, \Gamma x, \Gamma y$, and $\Gamma z \in \mathbb{F}_2^n$). The linear approximation for the modular addition is then $\Gamma x^T \cdot x \oplus \Gamma y^T \cdot y = \Gamma z^T \cdot z$ and is referred to as

$$+ : (\Gamma x|\Gamma y) \to \Gamma z.$$

Property 1 (Modular addition). In any linear approximation $(\Gamma x|\Gamma y) \to \Gamma z$ of the modular addition with a non-zero correlation, the most significant non-zero mask bit for $\Gamma x, \Gamma y$ and Γz is the same.

Property 1 is proven in Appendix B.

4.4 Linear approximation of one TEA/XTEA round

Using Property 1 for modular addition, as all other operations in TEA and XTEA are linear, we can derive conditions on a special class of approximations with non-zero correlation for the round function of TEA and XTEA. See Figures 4 and 3 for an illustration.

As in Subsection 4.1, the input and output of round r in TEA and XTEA are $(L_r|R_r)$ and $(L_{r+1}|R_{r+1})$, respectively. Correspondingly, $(\Gamma_r^L|\Gamma_r^R)$ and $(\Gamma_{r+1}^L|\Gamma_{r+1}^R)$ are input and output linear masks of the round. So the linear approximation over the round is

$$\text{(X)TEA round } r : (\Gamma_r^L|\Gamma_r^R) \to (\Gamma_{r+1}^L|\Gamma_{r+1}^R)$$

and has the following

Property 2 (One round). If $\varGamma_r^L = e_{i,\sim}$ and $\varGamma_r^R = e_{j,\sim}$, $(j < i)$, then one needs $\varGamma_{r+1}^R = e_{i,\sim}$ and $\varGamma_{r+1}^L = e_{i,\sim} \oplus e_{i+5\sim5}$ for the approximation to have a non-zero correlation. Similarly, for the decryption round function of TEA, if the input mask and the output mask for round r are $(\varGamma_r^L|\varGamma_r^R)$ and $(\varGamma_{r+1}^L|\varGamma_{r+1}^R)$, respectively. If $\varGamma_r^R = e_{i,\sim}$ and $\varGamma_r^L = e_{j,\sim}$, $(j < i)$, then we have $\varGamma_{r+1}^L = e_{i,\sim}$ and $\varGamma_{r+1}^R = e_{i,\sim} \oplus e_{i+5\sim5}$.

4.5 Zero correlation approximations for 14 and 15 rounds of TEA/XTEA

With the one-round property of linear approximation in TEA and XTEA derived in the previous subsection, we can identify zero correlation approximations over 14 and 15 rounds of both TEA and XTEA.

Proposition 4. *Over 15 rounds of TEA and XTEA, any linear approximation with input mask $(\varGamma_1^R|\varGamma_1^L) = (1|0)$ and output mask $(\varGamma_{15}^R|\varGamma_{15}^L) = (0|e_{1,\sim})$ has a correlation of exactly zero. Moreover, over 14 rounds of TEA and XTEA, any linear approximation with input mask $(\varGamma_1^R|\varGamma_1^L) = (1|0)$ and output mask $(\varGamma_{14}^R|\varGamma_{14}^L) = (e_{1,\sim}|\bar{e}_{5,\sim})$ has zero correlation.*

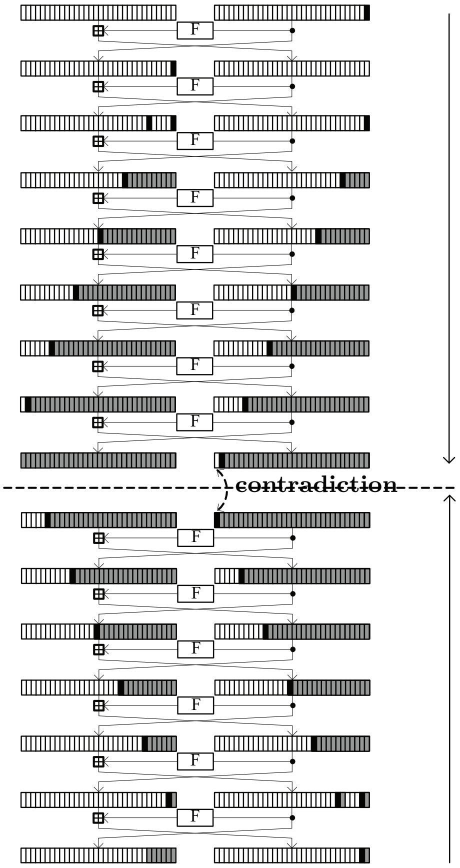


Fig. 3. Zero correlation linear approximation for 14-round TEA and XTEA (grey – undefined bits, black – bits set to 1)

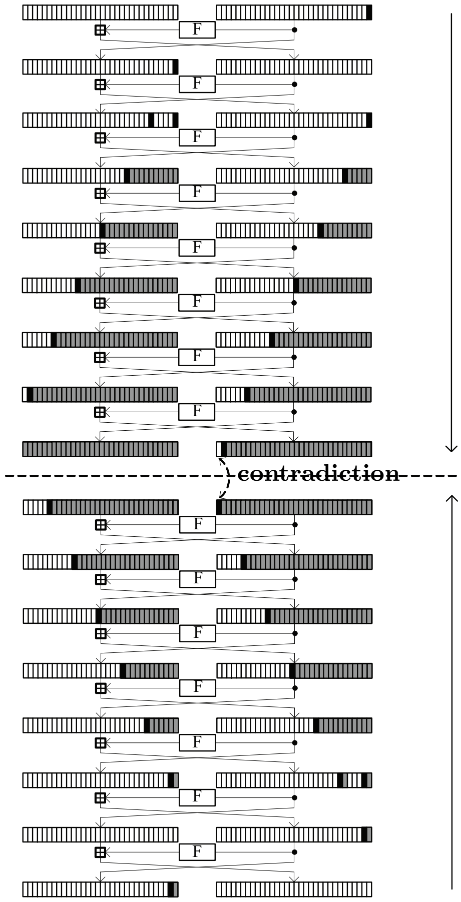


Fig. 4. Zero correlation linear approximation for 15-round TEA and XTEA (grey – undefined bits, black – bits set to 1)

Proof. First, we follow the linear approximation in the forward direction. From $\varGamma_1^L = 0$ and $\varGamma_1^R = 1$, it is obtained that $\varGamma_2^L = 0$ and $\varGamma_2^R = 1$, then we get $\varGamma_3^L = 1 \oplus (1 << 5)$ and $\varGamma_3^R = 1$. From Property 2, $\varGamma_3^L = 1 \oplus (1 << 5)$ and $\varGamma_3^R = 1$, then we have $\varGamma_4^R = e_{5,\sim}$ and $\varGamma_4^L =$

$e_{5,\sim} \oplus e_{5+5\sim 5} \oplus 1 = e_{10,\sim}$. Similarly, we get $(\Gamma_5^R|\Gamma_5^L) = (e_{10,\sim}|e_{15,\sim})$, $(\Gamma_6^R|\Gamma_6^L) = (e_{15,\sim}|e_{20,\sim})$, $(\Gamma_7^R|\Gamma_7^L) = (e_{20,\sim}|e_{25,\sim})$, $(\Gamma_8^R|\Gamma_8^L) = (e_{25,\sim}|e_{30,\sim})$ and $(\Gamma_9^R|\Gamma_9^L) = (e_{30,\sim}|?)$.

Second, we follow the 7-round linear approximation in the backward direction. From $\Gamma_{16}^L = e_{1,\sim}$ and $\Gamma_{16}^R = 0$, we can derive that $(\Gamma_{15}^R|\Gamma_{15}^L) = (e_{1,\sim}|0)$, $(\Gamma_{14}^R|\Gamma_{14}^L) = (e_{1,\sim} \oplus e_{6\sim 5}|e_{1,\sim})$, $(\Gamma_{13}^R|\Gamma_{13}^L) = (e_{11,\sim}|e_{6,\sim})$, $(\Gamma_{12}^R|\Gamma_{12}^L) = (e_{16,\sim}|e_{11,\sim})$, $(\Gamma_{11}^R|\Gamma_{11}^L) = (e_{21,\sim}|e_{16,\sim})$, $(\Gamma_{10}^R|\Gamma_{10}^L) = (e_{26,\sim}|e_{21,\sim})$ and $(\Gamma_9^R|\Gamma_9^L) = (e_{31,\sim}|e_{26,\sim})$.

From the forward direction, the most significant bit of Γ_9^R has to be zero, and from the backward direction, the most significant bit of Γ_9^R has to be one. This yields a contradiction and shows that there are no characteristics for this linear approximation. By the sufficient condition of [5] for constructing zero correlation linear approximations, this is enough for the approximation to have correlation zero. So the linear approximation for 15-round TEA and XTEA with the input mask $(1|0)$ and the output mask $(0|e_{1,\sim})$ has zero correlation. By restricting this linear approximation to 14 rounds and adding several undefined bits to the output mask, one gets all the claims of the proposition. □

There are only 2 zero correlation linear approximations of this form over 15 rounds. We note however that there are 2^7 different zero correlation linear approximations over 14 rounds of both TEA and XTEA. They can be generated by setting the undefined bits (depicted in gray in Figure 3 and Figure 4) to different values.

5 Zero correlation linear cryptanalysis of round-reduced (X)TEA

5.1 Key recovery for 21 rounds of TEA

For the cryptanalysis of 21-round TEA, we use the 14-round zero correlation approximations of the type depicted in Figure 3 of Subsection 4.5. The availability of many such approximations allows us to use the data complexity reduction technique of Section 3.

We place the 14-round zero correlation linear approximations in the middle of the 21-round TEA. It covers rounds 5 to 18. Following the procedure outlined in Subsection 2.3, up to the boundaries of the linear approximations, we partially encrypt over the 4 first rounds 1 to 4 and partially decrypt over the 3 last rounds 19 to 21. The attack is illustrated in Figure 5.

The linear approximations involve 9 state bits: R_5^0, $R_{19}^{1\sim 0}$, and $L_{19}^{5\sim 0}$. In the corresponding 9 bits of the input and output masks, only 7 can take on 0 and 1 values: ${\Gamma_{19}^R}^0$ and ${\Gamma_{19}^L}^{5\sim 0}$. For the evaluation of the linear approximations from a plaintext-ciphertext pair, we guess 54 key bits $K_0^{15\sim 0}$, $K_1^{15\sim 0}$, $K_2^{10\sim 0}$, and $K_3^{10\sim 0}$. The attack flow is as follows given N known plaintext-ciphertext pairs.

For each possible guess of the 54-bit subkey $\kappa = (K_0^{15\sim 0}|K_1^{15\sim 0}|K_2^{10\sim 0}|K_3^{10\sim 0})$:

1. Allocate a 128-bit counter W and set it to zero. W will contain the χ^2 statistic for the subkey guess κ.
2. Allocate a 64-bit counter $V[x]$ for each of 2^9 possible values of
$$x = (R_5^0|R_{19}^{1\sim 0}|L_{19}^{5\sim 0})$$
and set it to 0. $V[x]$ will contain the number of times the partial state value x occurs for N texts.
3. For each of N plaintext-ciphertext pairs: partially encrypt 4 rounds and partially decrypt 3 rounds, obtain the 9-bit value for $x = (R_5^0|R_{19}^{1\sim 0}|L_{19}^{5\sim 0})$ and add one to the counter $V[x]$.
4. For each of 2^7 zero correlation linear approximations:
   (a) Set the 64-bit counter U to zero.

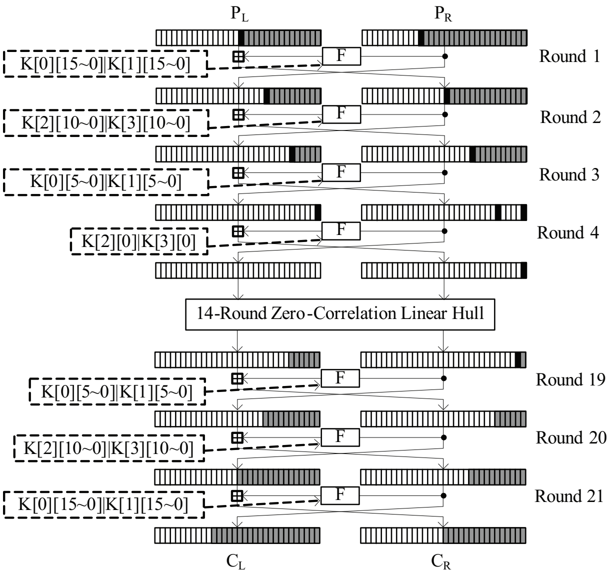


Fig. 5. Key recovery for 21 rounds of TEA. For the estimation of correlation, grey and black bits need to be computed and white bits are irrelevant. Uses the zero correlation approximation of Figure 3.

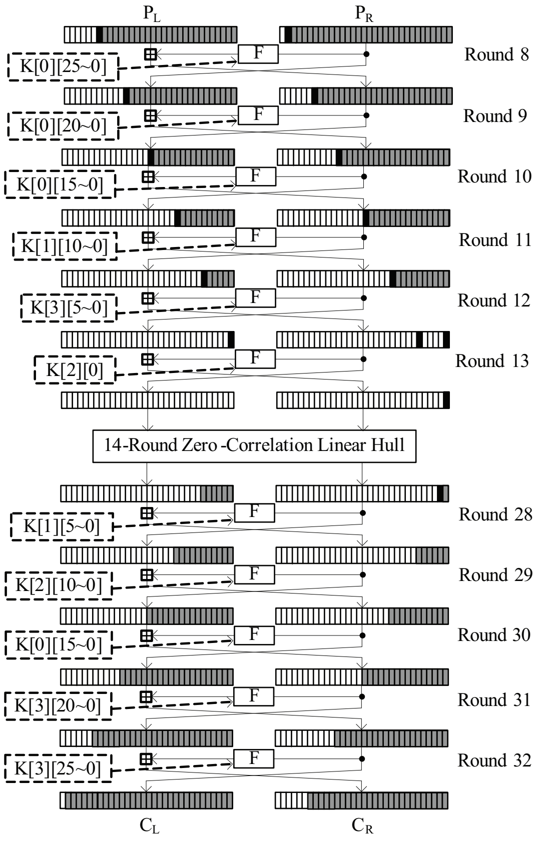


Fig. 6. Key recovery for 25 rounds of XTEA For the estimation of correlation, grey and black bits need to be computed and white bits are irrelevant. Uses the zero correlation approximation of Figure 3.

(b) For 2^9 values of x, verify if the linear approximation holds. If so, add $V[x]$ to U.
(c) $W = W + (2 \cdot U/N - 1)^2$.

5. If $W < t$, then κ is a possible subkey candidate and all cipher keys it is compatible with are tested exhaustively against a maximum of 3 plaintext-ciphertext pairs.

The correct 54-bit subkey κ is likely to be among the candidates with the χ^2 statistic W lower than the threshold $t = \sigma_0 \cdot z_{1-\beta_0} + \mu_0 = \frac{\sqrt{2l}}{N} \cdot z_{1-\beta_0} + \frac{l}{N} = \frac{\sqrt{2 \cdot 2^7}}{N} \cdot z_{1-\beta_0} + \frac{2^7}{N}$, see Subsection 3.1 with its Proposition 1 as well as Theorem 1.

In this attack, we set $\beta_0 = 2^{-2.7}$, $\beta_1 = 2^{-4.49}$ and get $z_{1-\beta_0} = 1$, $z_{1-\beta_1} = 1.7$. Note once again that $n = 64$ and $\ell = 2^7$. Theorem 1 suggests the data complexity of $N = 2^{62.62}$ known plaintext-ciphertexts with those parameters. The decision threshold is $t = 2^{-55.56}$.

The computational complexity is dominated by Steps 3 and 5. The computational complexity T_3 of Step 3 is 2^{54} times 7 half-round encryptions for each of N texts. This gives $T_3 = 2^{54} \cdot 2^{62.62} \cdot 7 \cdot 0.5/21 = 2^{114.03}$ 21-round TEA encryptions.

One in $1/\beta_1 = 2^{4.49}$ keys is expected to survive the test against zero correlation. The remaining key space is be covered by exhaustive search which is performed in Step 5. The computational complexity T_5 of Step 5 is about $T_5 = 2^{126-4.49} = 2^{121.51}$ 21-round encryptions using the equivalent key property. T_5 dominates the total computational complexity.

Summarizing the attack, its computational complexity is about $2^{121.51}$, data complexity is about $2^{62.62}$ known plaintext-ciphertext pairs, and the memory complexity is negligible. The success probability is about 0.846.

5.2 Key recovery for 25-round XTEA

Similarly to the attack on 21 rounds of TEA provided in the previous subsection, we use the 14-round zero correlation linear approximation depicted in Figure 3 to attack 25-round XTEA. Note that the attack covers rounds 8 to 32. It is illustrated in Figure 6. The linear approximations are placed in rounds 14 to 27. We partially encrypt 6 rounds (8 to 13) and partially decrypt 5 rounds (28 to 32) to evaluate the parity of approximations.

The linear approximations involve 9 bits and in the corresponding 9 bits of the input and output masks, again only 7 can take on 0 and 1 values: ${\Gamma_{28}^{R}}^{0}$ and ${\Gamma_{28}^{L}}^{5\sim 0}$. For the evaluation of the linear approximations from a plaintext-ciphertext pair, we guess altogether 74 key bits $K_0^{25\sim 0}$, $K_1^{10\sim 0}$, $K_2^{10\sim 0}$, and $K_3^{25\sim 0}$. The attack itself is similar to that on 21-round TEA.

For each possible 63-bit value of $(K_0^{25\sim 0}|K_1^{10\sim 0}|K_3^{25\sim 0})$:

1. Allocate and set to zero the 32-bit counter $V_1[x]$ for each of 2^{30} possible values of
$$x = (R_{13}^0|R_{13}^5|L_{13}^0|R_{30}^{10\sim 0}|L_{30}^{15\sim 0}).$$
2. For each of N plaintext-ciphertext pairs: partially encrypt 5 rounds and partially decrypt 3 rounds, obtain 30-bit $x = (R_{13}^0|R_{13}^5|L_{13}^0|R_{30}^{10\sim 0}|L_{30}^{15\sim 0})$, and add one to $V_1[x]$.
3. For each possible 11 bits value of $K_2^{10\sim 0}$:
   (a) Allocate and set to zero a 128-bit counter W.
   (b) Allocate and set to zero a 64-bit counter $V_2[y]$ for each of 2^9 possible values of
   $$y = (R_{14}^0|L_{28}^{5\sim 0}|R_{28}^{1\sim 0}).$$
   (c) Encrypt one round and decrypt two rounds for 2^{30} values for x to get 9 bits of y and add $V_1[x]$ to $V_2[y]$.
   (d) For each of 2^7 zero correlation linear approximations:
      i. Set the 64-bit counter U to zero.
      ii. For 2^9 values of y, verify if the linear approximation holds. If so, add $V_2[y]$ to the counter U.
      iii. $W = W + (2 \cdot U/N - 1)^2$.
   (e) If $W < t$, then κ is a possible subkey candidate and all cipher keys it is compatible with are tested exhaustively against a maximum of 3 plaintext-ciphertext pairs.

The correct 74-bit subkey is likely to be among the candidates with the χ^2 statistic W lower than the threshold t. As we again set $\beta_0 = 2^{-2.7}$ and $\beta_1 = 2^{-4.49}$, we obtain $N = 2^{62.62}$ and $t = 2^{-55.56}$.

The computational complexity is dominated by Step 2 and checking for false positives in Step 3(e). T_2 of Step 2 is constituted by $2^{63}N$ computations of 5 rounds of 25-round XTEA and by $2^{63}N$ increments in the memory of 2^{30} 32-bit counters. Assuming that one increment of a memory cell costs one XTEA round, we obtain $T_2 = 2^{63} \cdot 2^{62.62} \cdot (5/25 + 1/25) = 2^{123.56}$. In Step 3(e), the remaining $T_{3(e)} = 2^{128-4.49} = 2^{123.51}$ keys can checked exhaustively by the same number of 25-round XTEA encryptions. Thus, the overall computational complexity is about $T_2 + T_{3(e)} = 2^{123.56} + 2^{123.51} = 2^{124.53}$ 25-round XTEA encryptions. The memory complexity is 2^{30} 32-bit words. Again, the data complexity is about $2^{62.62}$ known plaintext-ciphertext pairs, and the success probability is about 0.846.

5.3 Attacking more rounds with the full codebook

The attacks in the previous subsections use 14-round zero correlation linear approximations to enable data complexity reduction. As we only identified 2 15-round approximations, we cannot use this longer property to attack more rounds and still get a non-negligible decrease in the number of texts required. By taking advantage of the full codebook, we are however able to perform key recovery for up to 23 rounds of TEA and up to 27 rounds of XTEA, see Appendix D.

6 Conclusions

In this paper, we have demonstrated a technique for data complexity reduction for the promising recent zero correlation linear cryptanalysis which is based on linear approximations holding with a probability of exactly 1/2. This attack vector can be seen as the counterpart of the successful impossible differential cryptanalysis in the domain of linear cryptanalysis. Using ℓ linear approximations, we are able to reduce the data complexity to $\mathcal{O}(2^n/\sqrt{\ell})$, where n is the block size of the cipher.

As an application, we show 14- and 15-round linear approximations with correlation zero for round-reduced TEA and XTEA. Based on those, we propose key recovery attacks on 21-round TEA and 25-round XTEA with data complexity $2^{62.62}$ as well as on 23-round TEA and 27-round XTEA by taking advantage of all 2^{64} texts. All four attacks are the best key recoveries for both TEA and XTEA published to date in the single secret key setting. For these ciphers, our zero correlation linear attacks outperform their differential counterpart (impossible differential attacks), among other techniques.

These two contributions make the zero correlation linear cryptanalysis one of the major cryptanalytic techniques available today for attacking and evaluating symmetric-key ciphers.

Acknowledgements. We would like to thank Vincent Rijmen and Gregor Leander for insightful discussions. Andrey Bogdanov is postdoctoral fellow of the Fund for Scientific Research - Flanders (FWO). This work has been supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State, by the European Commission under contract number ICT-2007-216676 ECRYPT NoE phase II, by KU Leuven-BOF (OT/08/027), by the Research Council KU Leuven (GOA TENSE), by NSFC Projects (No.61133013, No.61070244 and No.60931160442) as well as Outstanding Young Scientists Foundation Grant of Shandong Province (No.BS2009DX030).

References

1. E. Biham, A. Biryukov, A. Shamir: Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In: EUROCRYPT'99, LNCS, pp. 12–23, Springer-Verlag, 1999.
2. E. Biham, O. Dunkelman, N. Keller: Related-Key Impossible Differential Attacks on 8-Round AES-192. In: CT-RSA'06, LNCS, pp. 21–33, Springer-Verlag, 2006.
3. E. Biham, A. Shamir: Differential Cryptanalysis of DES-like Cryptosystems. In: CRYPTO'90, LNCS, pp. 2–21, Springer-Verlag, 1990.
4. A. Bogdanov, D. Khovratovich, C. Rechberger: Biclique Cryptanalysis of the Full AES. In: ASIACRYPT'11, LNCS, pp. 344–371, Springer-Verlag, 2011.
5. A. Bogdanov, V. Rijmen: Zero Correlation Linear Cryptanalysis of Block Ciphers, IACR Eprint Archive Report 2011/123, March 2011.
6. J. Borst, L. R. Knudsen, V. Rijmen: Two Attacks on Reduced IDEA. In: EUROCRYPT'97, LNCS, pp. 1–13, Springer-Verlag, 1997.
7. C. Bouillaguet, O. Dunkelman, G. Leurent, P.-A. Fouque: Another Look at Complementation Properties. In: FSE 2010, LNCS, vol. 6147, pp. 347–364, 2010.
8. J. Chen, M. Wang, B. Preneel: Impossible Differential Cryptanalysis of Lightweight Block Ciphers TEA, XTEA and HIGHT. IACR Eprint Archive Report 2011/616, 2011.
9. B. Collard and F.-X. Standaert: Experimenting Linear Cryptanalysis. In P. Junod, A. Canteaut (eds.) *Advanced Linear Cryptanalysis of Block and Stream Ciphers*, vol, 7 of *Cryptology and Information Security Series*. IOS Press, 2011.
10. B. Collard, F.-X. Standaert, J.-J. Quisquater: Improving the Time Complexity of Matsui's Linear Cryptanalysis. In: ICISC'07, LNCS, vol. 4817, pp 77–88, Springer-Verlag, 2007.
11. J. Daemen, R. Govaerts, J. Vandewalle: Correlation Matrices. In: FSE 1994, LNCS, vol. 1008, pp. 275–285, Springer-Verlag, 1995.
12. J. Daemen, V. Rijmen: Probability distributions of correlations and differentials in block ciphers. Journal on Mathematical Cryptology 1(3), pp. 221-242, 2007.
13. J. Daemen, V. Rijmen. *The Design of Rijndael: AES – The Advanced Encryption Standard*. Springer-Verlag, 2002.

14. H. Demirci and A.A. Selçuk: A Meet-in-the-Middle Attack on 8-Round AES. In: FSE'08, LNCS, vol. 5086, pp. 116–126, Springer-Verlag, 2008.
15. O. Dunkelman, N. Keller: An Improved Impossible Differential Attack on MISTY1. In: ASIACRYPT'08, LNCS, vol. 5350, pp. 441–454, Springer-Verlag, 2008.
16. J. Etrog, M. J. B. Robshaw: On Unbiased Linear Approximations. In ACISP'10, LNCS, vol. 6168, pp. 74–86. Springer-Verlag, 2010.
17. W. Feller: *An Introduction to Probability Theory and Its Applications*, vol. 1, Wiley & Sons, 1968.
18. O. Dunkelman, N. Keller, A. Shamir: Improved Single-Key Attacks on 8-Round AES-192 and AES-256. In: ASIACRYPT'10, LNCS, vol. 6477, pp. 158–176, Springer-Verlag, 2010.
19. P. Hoel, S. Port, C. Stone: *Introduction to Probability Theory*, Brooks Cole, 1972.
20. S. Hong, D. Hong, Y. Ko, D. Chang, W. Lee, S. Lee: Differential Cryptanalysis of TEA and XTEA. In: ICISC'03, LNCS, vol. 2971, pp. 402–417, Springer-Verlag, 2004.
21. P. Junod: On the Complexity of Matsui's Attack. In: SAC'01, LNCS, vol. 2259, pp. 199–211, Springer-Verlag, 2001.
22. J.-P. Kaps: Chai-Tea, Cryptographic Hardware Implementations of xTEA. In: INDOCRYPT 2008, LNCS, vol. 5365, pp. 363–375, Springer-Verlag, 2008.
23. J. Kelsey, B. Schneier, D. Wagner: Key-Schedule Cryptoanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In: CRYPTO 1996, LNCS, vol. 1109, pp. 237–251, Springer-Verlag, 1996.
24. J. Kelsey, B. Schneier, D. Wagner: Related-key Cryptanalysis of 3-WAY, Biham-DES,CAST, DES-X, NewDES, RC2, and TEA. In: ICICS'97, LNCS, vol. 1334, pp. 233–246, Springer-Verlag, 1997.
25. E. Lee, D. Hong, D. Chang, S. Hong, J. Lim: A Weak Key Class of XTEA for a Related-Key Rectangle Attack. In: VIETCRYPT 2006, LNCS, vol. 4341, pp. 286–297, Springer-Verlag, 2006.
26. J. Lu: Related-key rectangle attack on 36 rounds of the XTEA block cipher. International Journal of Information Security 8(1), 1-11 (2009)
27. J. Lu, J. Kim, N. Keller, O. Dunkelman: Improving the Efficiency of Impossible Differential Cryptanalysis of Reduced Camellia and MISTY1. In: CT-RSA'08, LNCS, vol. 4964, pp. 370–386, Springer-Verlag, 2008.
28. J. Lu, O. Dunkelman, N. Keller, J. Kim: New Impossible Differential Attacks on AES. In: INDOCRYPT'08, LNCS, vol. 5365, pp. 279–293, Springer-Verlag, 2008.
29. H. Mala, M. Dakhilalian, V. Rijmen, M. Modarres-Hashemi: Improved Impossible Differential Cryptanalysis of 7-Round AES-128. In: INDOCRYPT'10, LNCS, vol. 6498, pp. 282–291, Springer-Verlag, 2010.
30. M. Matsui: Linear cryptanalysis method for DES cipher. In: EUROCRYPT'93, LNCS, vol. 765, pp. 386–397, Springer-Verlag, 1993.
31. M. Matsui: The First Experimental Cryptanalysis of the Data Encryption Standard. In: CRYPTO'94, LNCS, vol. 839, pp. 1–11, Springer-Verlag, 1994.
32. D. Moon, K. Hwang, W. Lee, S. Lee, J. Lim,: Impossible Differential Cryptanalysis of Reduced Round XTEA and TEA. In: FSE 2002, LNCS, vol. 2365, pp. 49–60, Springer-Verlag, 2002.
33. R.M. Needham, D.J. Wheeler: Tea extensions. Technical report, Computer Laboratory, University of Cambridge, October 1997, http://www.cix.co.uk/∼ klockstone/xtea.pdf
34. K. Nyberg: Correlation theorems in cryptanalysis. Discrete Applied Mathematics, 111(1-2):177–188, 2001.
35. L. O'Connor: Properties of Linear Approximation Tables. In: FSE 1994, LNCS, vol. 1008, pp. 131–136, Springer-Verlag, 1995.
36. A. Röck, K. Nyberg: Exploiting Linear Hull in Matsui's Algorithm 1. WCC'11, 2011.
37. G. Sekar, N. Mouha, V. Velichkov, B. Preneel: Meet-in-the-Middle Attacks on Reduced-Round XTEA. In: CT-RSA 2011, LNCS, vol. 6558, pp. 250-267, Springer-Verlag, 2011.
38. A.A. Selçuk: On Probability of Success in Linear and Differential Cryptanalysis. Journal of Cryptology, Volume 21(1), pp. 131–147, Springer-Verlag, 2008.
39. T. Shirai, K. Shibutani, T. Akishita, S. Moriai, T. Iwata: The 128-Bit Blockcipher CLEFIA (Extended Abstract). In: FSE'07, LNCS, vol.4593, pp. 181–195. Springer-Verlag, 2007.
40. M. Steil: 17 Mistakes Microsoft Made in the Xbox Security System. Chaos Communication Congress 2005, 2005. http://events.ccc.de/congress/2005/fahrplan/events/559.en.html
41. Y. Tsunoo, E. Tsujihara, M. Shigeri, T. Saito, T. Suzaki, H. Kubo. Impossible Differential Cryptanalysis of CLEFIA. In: FSE'08, LNCS, vol. 5086, pp. 398–411, Springer-Verlag, 2008.
42. S. Vaudenay. *Decorrelation: A Theory for Block Cipher Security.* J. Cryptology, 16(4):249–286, Springer-Verlag, 2003.
43. D.J. Wheeler, R.M. Needham: TEA, a Tiny Encryption Algorithm. In: FSE'94, LNCS, vol. 1008, pp. 363–366, Springer-Verlag, 1995.
44. Y. Zheng, T. Matsumoto, H. Imai: On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses. In: CRYPTO'89, LNCS, vol. 435, pp. 461–480, Springer-Verlag, 1989.

A Proof of Proposition 1 (distinguishing distributions)

Proof. Again, first assume that $\mu_0 < \mu_1$. The error probabilities β_0 and β_1 can be derived from the value of threshold t and the CDFs of the two normal distributions:

$$\begin{aligned} \beta_0 &= 1 - \Phi_{\mu_0,\sigma_0}(t), \\ \beta_1 &= \Phi_{\mu_1,\sigma_1}(t), \end{aligned} \tag{7}$$

where Φ_{μ_i,σ_i} is the CDF of the respective normal distribution. (7) can be rewritten as follows using the CDF of the standard normal distribution:

$$\begin{aligned} \beta_0 &= 1 - \Phi_{0,1}\left(\tfrac{t-\mu_0}{\sigma_0}\right), \\ \beta_1 &= \Phi_{0,1}\left(\tfrac{t-\mu_1}{\sigma_1}\right). \end{aligned} \tag{8}$$

By going to quantiles in (8), one obtains

$$\begin{aligned} z_{1-\beta_0} &= \tfrac{t-\mu_0}{\sigma_0}, \\ z_{\beta_1} &= \tfrac{t-\mu_1}{\sigma_1}. \end{aligned}$$

Expressing and equating t in the two cases yields:

$$\mu_0 + \sigma_0 z_{1-\beta_0} = \mu_1 + \sigma_1 z_{\beta_1}$$

and, eventually, recalling that $z_{\beta_1} = -z_{1-\beta_1}$ gives the relation

$$\frac{\sigma_0 z_{1-\beta_0} + \sigma_1 z_{1-\beta_1}}{\mu_1 - \mu_0} = 1. \tag{9}$$

Considering $\mu_0 > \mu_1$ yields a change of denominator in (9) to $\mu_0 - \mu_1$. The claim of the theorem follows. □

B Proof of Property 1 (modular addition)

Proof. We denote the i-th bit for x, y and z as x_i, y_i and z_i, $0 \leq i \leq n-1$, respectively. From the modular addition, we have

$$\begin{aligned} & z_0 = x_0 \oplus y_0, c_0 = 0, \\ & z_1 = x_1 \oplus y_1 \oplus c_1, c_1 = f_1(x_0, y_0), \\ & \cdots, \\ & z_i = x_i \oplus y_i \oplus c_i, c_i = f_i(x_0, x_1, \ldots, x_{i-1}, y_0, y_1, \ldots, y_{i-2}). \cdots, \\ & z_{n-1} = x_{n-1} \oplus y_{n-1} \oplus c_{n-1}, c_{n-1} = f_{n-1}(x_0, x_1, \ldots, x_{n-2}, y_0, y_1, \ldots, y_{n-2}), \end{aligned}$$

where c_i is the carrying bit of the i-th bit and f_i is the non-linear carrying function of the i-th bit. From the above equations, the linear approximations with non-zero bias have the following form:

$$\begin{aligned} & z_0 = x_0 \oplus y_0, \\ & z_1 = x_1 \oplus y_1[\oplus x_0 \oplus y_0], \\ & z_2 = x_2 \oplus y_2[\oplus x_1 \oplus y_1 \oplus x_0 \oplus y_0], \\ & \cdots, \\ & z_i = x_i \oplus y_i[\oplus x_{i-1} \oplus y_{i-1} \oplus \ldots \oplus x_0 \oplus y_0], \\ & \cdots, \\ & z_{n-1} = x_{n-1} \oplus y_{n-1}[\oplus x_{n-2} \oplus y_{n-2} \oplus \ldots \oplus x_0 \oplus y_0], \end{aligned}$$

where the terms in the square brackets are optional. So any linear approximation with non-zero bias will be produced from any one or the combination from the above linear relations which can be denoted as follows,

$$z_i[\oplus z_{i-1} \oplus \cdots \oplus z_0] = x_i \oplus y_i[\oplus x_{i-1} \oplus y_{i-1} \oplus \ldots \oplus x_0 \oplus y_0].$$

If there is a linear approximation with the following form,

$$z_j \oplus z_i[\oplus z_{i-1} \oplus \cdots \oplus z_0] = x_i \oplus y_i[\oplus x_{i-1} \oplus y_{i-1} \oplus \ldots \oplus x_0 \oplus y_0], i < j < n. \tag{10}$$

We substitute the equation $z_j = x_j \oplus y_j \oplus c_j, c_j = f_i(x_0, x_1, \ldots, x_{j-1}, y_0, y_1, \ldots, y_{j-1})$ into Equation 10, we get

$$\begin{aligned}&x_j \oplus y_j \oplus f_i(x_0, x_1, \ldots, x_{j-1}, y_0, y_1, \ldots, y_{j-1})\\ &\oplus z_i[\oplus z_{i-1} \oplus \cdots \oplus z_0] = x_i \oplus y_i[\oplus x_{i-1} \oplus y_{i-1} \oplus \ldots \oplus x_0 \oplus y_0], i < j < n.\end{aligned}$$

In the above equation, $x[j], x[j-1], \ldots x[i+1], y[j], y[j-1], \ldots y[i+1]$ are not related with $z_i, z_{i-1}, \ldots, z_0$, so they are independent random variables. The involved independent random variables will make the linear approximation Equation (10)be random, so the bias for Equation (10)will be zero. Similarly, the linear approximation with the following forms will also have zero bias,

$$\begin{aligned}z_i[\oplus z_{i-1} \oplus \cdots \oplus z_0] = x[j] \oplus x_i \oplus y_i[\oplus x_{i-1} \oplus y_{i-1} \oplus \ldots \oplus x_0 \oplus y_0], i < j < n.\\ z_i[\oplus z_{i-1} \oplus \cdots \oplus z_0] = y[j] \oplus x_i \oplus y_i[\oplus x_{i-1} \oplus y_{i-1} \oplus \ldots \oplus x_0 \oplus y_0], i < j < n.\end{aligned} \tag{11}$$

This means that the most non-zero significant bits for $\Gamma x, \Gamma y$ and Γz must be same, otherwise, the linear approximation will have zero bias. □

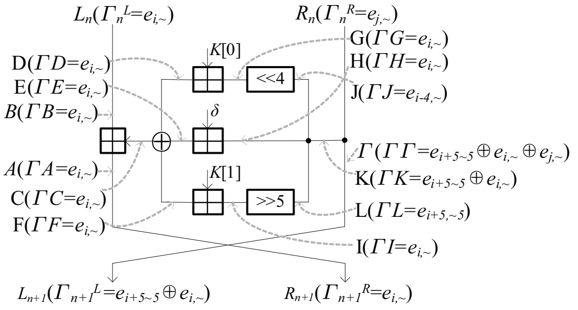

$L_n(\Gamma_n{}^L=e_{i,\sim})$
$R_n(\Gamma_n{}^R=e_{j,\sim})$
$G(\Gamma G=e_{i,\sim})$
$H(\Gamma H=e_{i,\sim})$
$J(\Gamma J=e_{i-4,\sim})$
$D(\Gamma D=e_{i,\sim})$
$E(\Gamma E=e_{i,\sim})$
$B(\Gamma B=e_{i,\sim})$
$K[0]$
<<4
δ
$\Gamma(\Gamma\Gamma=e_{i+5\sim5}\oplus e_{i,\sim}\oplus e_{j,\sim})$
$A(\Gamma A=e_{i,\sim})$
$K(\Gamma K=e_{i+5\sim5}\oplus e_{i,\sim})$
$K[1]$
>>5
$C(\Gamma C=e_{i,\sim})$
$L(\Gamma L=e_{i+5,\sim5})$
$F(\Gamma F=e_{i,\sim})$
$I(\Gamma I=e_{i,\sim})$
$L_{n+1}(\Gamma_{n+1}{}^L=e_{i+5\sim5}\oplus e_{i,\sim})$
$R_{n+1}(\Gamma_{n+1}{}^R=e_{i,\sim})$

Fig. 7. Linear approximation of one TEA round

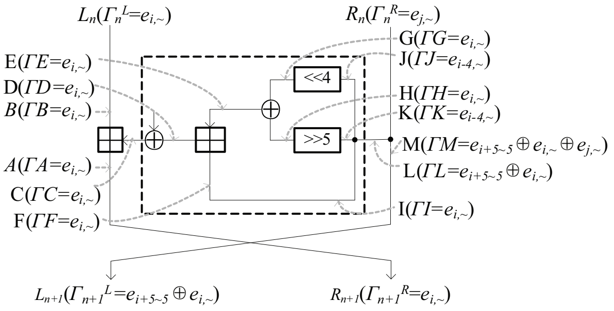

$L_n(\Gamma_n{}^L=e_{i,\sim})$
$R_n(\Gamma_n{}^R=e_{j,\sim})$
$G(\Gamma G=e_{i,\sim})$
$J(\Gamma J=e_{i-4,\sim})$
$E(\Gamma E=e_{i,\sim})$
$D(\Gamma D=e_{i,\sim})$
<<4
$H(\Gamma H=e_{i,\sim})$
$B(\Gamma B=e_{i,\sim})$
$K(\Gamma K=e_{i-4,\sim})$
>>5
$M(\Gamma M=e_{i+5\sim5}\oplus e_{i,\sim}\oplus e_{j,\sim})$
$A(\Gamma A=e_{i,\sim})$
$L(\Gamma L=e_{i+5\sim5}\oplus e_{i,\sim})$
$C(\Gamma C=e_{i,\sim})$
$I(\Gamma I=e_{i,\sim})$
$F(\Gamma F=e_{i,\sim})$
$L_{n+1}(\Gamma_{n+1}{}^L=e_{i+5\sim5}\oplus e_{i,\sim})$
$R_{n+1}(\Gamma_{n+1}{}^R=e_{i,\sim})$

Fig. 8. Linear approximation of one XTEA round

C Proof of Property 2 (one round)

Proof. The linear approximation for the encryption round function of TEA and XTEA have been shown in Fig.7 and Fig. 8, we use the notations $A, B, C, D, E, F, G, H, I, J, K, L, M$ to denote the intermediate variables and the notation ΓX to denote the respective mask value for $X \in \{A, B, C, D, E, F, G, H, I, J, K, L, M\}$. Next, we will only give the proof for the linear approximation of TEA. The proof for XTEA is similar as that for TEA and we will not describe it due to the limited space.

From Theorem 1, $C = A + B$, $MA = e_{i,\sim}$, so we have $\Gamma B = e_{i,\sim}$ and $\Gamma C = e_{i,\sim}$. Then from Lemma 1 in [5] and $C = D \oplus E \oplus F$, we can get $\Gamma D = \Gamma E = \Gamma F = \Gamma C = e_{i,\sim}$. From

Theorem 1, $D = G + K[0]$ and $\Gamma D = e_{i,\sim}$, so $\Gamma G = e_{i,\sim}$; $E = H + \delta$ and $\Gamma E = e_{i,\sim}$, so $\Gamma H = e_{i,\sim}$; $F = I + K[1]$ and $\Gamma F = e_{i,\sim}$, so $\Gamma I = e_{i,\sim}$. As $G = J \ll 4$ and $I = L \gg 5$, then $\Gamma J = \Gamma G \gg 4 = e_{i-4,\sim}$ and $\Gamma L = \Gamma I \ll 5 = e_{i+5\sim5}$. From Lemma 2 in [5], we have $\Gamma K = \Gamma G \oplus \Gamma H \oplus \Gamma I = \Gamma H \oplus \Gamma J \oplus \Gamma L = e_{i,\sim} \oplus e_{i-4,\sim} \oplus e_{i+5\sim5} = e_{i,\sim} \oplus e_{i+5\sim5}$. As $j < i$, $\Gamma M = \Gamma K \oplus \Gamma_n^R = e_{i,\sim} \oplus e_{i+5\sim5} \oplus e_{j,\sim} = e_{i,\sim} \oplus e_{i+5\sim5}$.

The proof for the linear approximation of the decryption round function can be proved in the same way. □

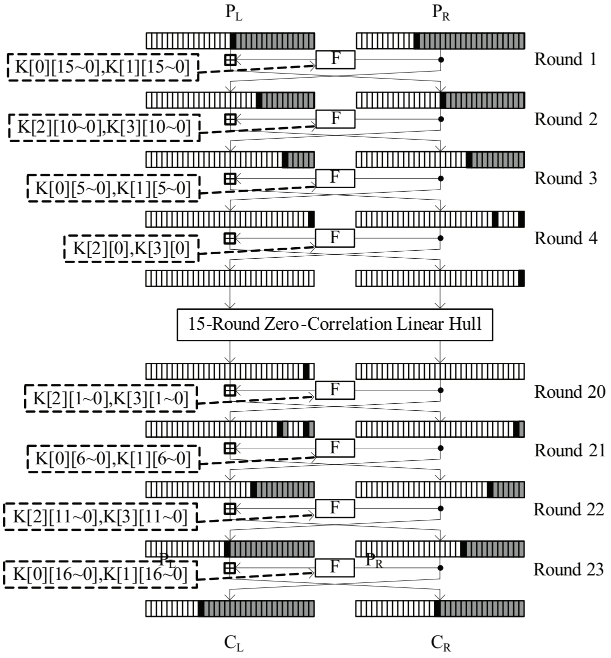


Fig. 9. Key recovery for 23 rounds of TEA For the estimation of correlation, grey and black bits need to be computed and white bits are irrelevant. Uses the zero correlation approximation of Figure 4.

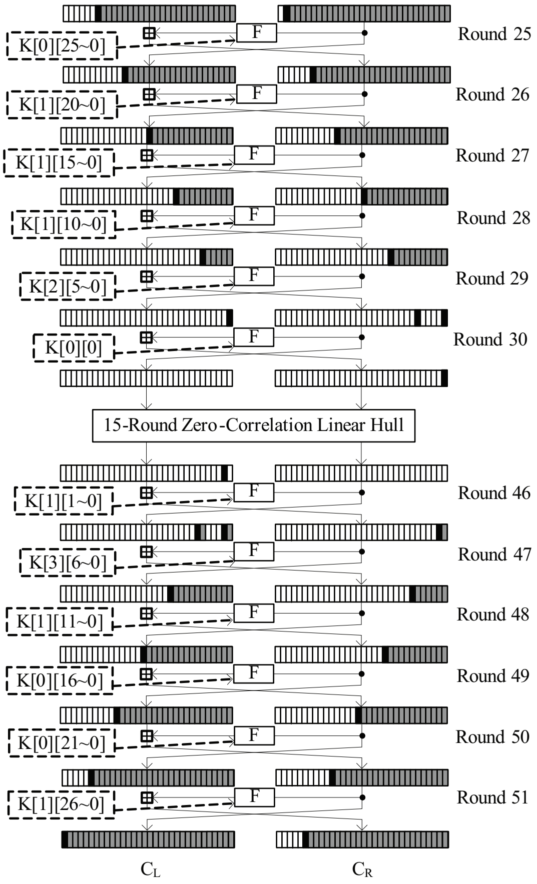


Fig. 10. Key recovery for 27 rounds of XTEA. For the estimation of correlation, grey and black bits need to be computed and white bits are irrelevant. Uses the zero correlation approximation of Figure 4.

D Key Recovery for More Rounds of (X)TEA with the Full Codebook

D.1 Key Recovery for 23 Rounds of TEA

We use the 15-round zero correlation linear approximations of Figure 4 to attack 23-round TEA, see Figure 9 for an illustration.

Now we use the basic zero correlation linear cryptanalysis with the full code book. To compute the parity of the approximation, we need to guess 58 bits: $(K_0^{16\sim0}|K_1^{16\sim0}|K_2^{11\sim0}|K_3^{11\sim0})$. For each guess, we partially encrypt 4 rounds and decrypt 4 rounds for the whole code book to get $R_5^0|L_{20}^1$ and verify if the equation holds. The computational complexity is dominated by those computations: $2^{58+64} \cdot (1 + 0.5 \cdot 3 + 0.5 \cdot 4)/23 \simeq 2^{119.64}$. Memory complexity is negligible. Data complexity is 2^{64}. Success probability is 1.

D.2 Key Recovery for 27 Rounds of XTEA

We use the same 15-round zero correlation linear approximation to attack 27-round XTEA, see Figure 10 for the attack. Again, using the full codebook, we rely on the basic zero correlation linear cryptanalysis procedure of [5].

The attack is proceeded as follows:

For each possible 59 bits value of $(K_0^{25\sim0}|K_1^{26\sim0}|K_2^{5\sim0})$:

- Allocate and set to zero the 64-bit counter $V[x]$ for each of 2^{22} possible values of
$$x = (R_{30}^5|R_{30}^0||L_{30}^0|R_{48}^{6\sim0}|L_{48}^{11\sim0}).$$
- Partially encrypt 5 rounds from round 25 and partially decrypt 4 rounds from round 51 for the whole code book, get 22-bit $(R_{30}^5|R_{30}^0||L_{30}^0|R_{48}^{6\sim0}|L_{48}^{11\sim0})$ and add one to $V[x]$.
- For each possible 7 bits value of $K_3^{6\sim0}$:
    - Partially encrypt 1 round from round 30 and decrypt 2 rounds from 47 for 2^{22} possible values for x to get 2 bits of $(R_{31}^0|L_{46}^1)$ and verify if the linear approximation holds.
    - If the counter equals to zero, it means that the guessed value for key bits is right with high probability.

The computational complexity of the attack is dominated by the partial encryption and decryption: $2^{59} \cdot 2^{64} \cdot (2 + 0.5 \cdot 3 + 2 + 0.5 \cdot 2)/27 = 2^{120.71}$ 27-round XTEA encryptions. The memory complexity is 2^{23} 32-bit words. Data complexity is 2^{64}. Success probability is 1.

A Model for Structure Attacks, with Applications to PRESENT and Serpent

Meiqin Wang[1,2,3,⋆], Yue Sun[4], Elmar Tischhauser[2,3,⋆⋆], and Bart Preneel[2,3]

[1] Key Laboratory of Cryptologic Technology and Information Security,
Ministry of Education, Shandong University, Jinan 250100, China.
[2] Department of Electrical Engineering ESAT/SCD-COSIC, Katholieke Universiteit Leuven,
Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium.
[3] Interdisciplinary Institute for BroadBand Technology (IBBT), Belgium.
[4] Institute for Advanced Study, Tsinghua University, Beijing 100084, China.
mqwang@sdu.edu.cn

Abstract. Differential cryptanalysis is a classic cryptanalytic method for block ciphers, hash functions and stream ciphers. Many extensions and refinements of differential cryptanalysis have been developed. In this paper, we focus on the use of so-called structures in differential attacks, *i.e.* the use of multiple input and one output difference. We give a general model and complexity analysis for structure attacks and show how to choose the set of differentials to minimize the time and data complexities. Being a subclass of multiple differential attacks in general, structure attacks can also be analyzed in the model of Blondeau *et al.* from FSE 2011. In this very general model, a restrictive condition on the set of input differences is required for the complexity analysis. We demonstrate that in our dedicated model for structure attacks, this condition can be relaxed, which allows us to consider a wider range of differentials. Finally, we point out an inconsistency in the FSE 2011 attack on 18 rounds of the block cipher PRESENT and use our model for structure attacks to attack 18-round PRESENT and improve the previous structure attacks on 7-round and 8-round Serpent. To the best of our knowledge, those attacks are the best known differential attacks on these two block ciphers.

Keywords: Structure Attack, Block Cipher, Differential, PRESENT, Serpent

1 Introduction

Differential cryptanalysis [2] is a classic cryptanalytic method that has been successfully applied to block ciphers, hash functions and stream ciphers. The key step for a differential attack is to identify a differential characteristic with high probability as a distinguisher, then use it to recover (part of) the key. Lai *et al.* propose the notion of *differential* which encompasses the collection of all possible differential characteristics [13] for one fixed input and output difference. A lower bound for the probability of a differential (and thus, an upper bound for the complexity of the attack) can be obtained by combining the probabilities of a number of differential characteristics belonging to the differential. Therefore, differentials give a better estimation of the actual attack complexity than characteristics, since the distinguisher can exploit any characteristic belonging to the differential. In order to further improve differential attacks, multiple differentials with a single output difference but multiple input differences can be used. This can reduce the data complexity provided that the set of input differences for the differentials can be combined in a so-called *structure*. Therefore, we call this type of differential attacks *structure attacks*. The structure technique in differential cryptanalysis was originally introduced in a more restrictive

[⋆] This author is supported by 973 Project (No.2007CB807902), National Natural Science Foundation of China (Grant No.61133013, No.61070244 and No.60931160442), Outstanding Young Scientists Foundation Grant of Shandong Province (No.BS2009DX030), Shandong University Initiative Scientific Research Program (2009TS087)

[⋆⋆] Elmar Tischhauser is a research assistant of the F.W.O., Fund for Scientific Research — Flanders.

way as *quartets* to attack DES [2], and multiple differential characteristics with multiple input differences and a single output difference have been used to attack DES. In addition, Biham *et al.* use the structure technique to attack reduced-round versions of the Serpent block cipher [3].

At FSE 2011, Blondeau *et al.* proposed multiple differential cryptanalysis with multiple input differences and multiple output differences [4] and gave an explicit formula to compute the success probability of multiple differential cryptanalysis. Traditionally, a normal approximation to the binomial distribution was used to evaluate the success probability of a differential attack [18, 19]. The approach of [4] provides a more accurate estimation of the success probability. Since structure attacks are a special case of multiple differential cryptanalysis, those results also apply to our structure attacks.

However, in order to ensure that one pair of ciphertexts can be only counted once, the model of [4] requires a certain condition to be met (see Definition 1), which severely restricts the set of input difference values that can be used in an attack. In this paper, we demonstrate that this condition on the set of the input difference values is so strong that many valuable differentials may be excluded. We show that in the structure technique, this condition can be relaxed without counting ciphertexts more than once. This enables us to choose our differentials more freely, leading to improved attack complexities. Moreover, in our model, the analysis of the data and time complexities and the success rate is carried out independently of the general framework of [4], and most importantly does not rely on condition (4) to be fulfilled. This is an important difference to the setting of [4], in which the filtering step also deals with the restriction of condition (4) (although this is not made explicit), but where this condition is still required for the analysis of the data complexity and success rate of the attack.

We stress that this condition and the general model of [4] are still necessary for the analysis of the general case where one has multiple input and multiple output differences. What we propose in this paper, is a tailored model for structure attacks, which are an important and often particularly efficient subclass of multiple differential cryptanalysis.

Furthermore, the multiple differential attack on 18-round PRESENT [4] uses 561 differentials with 17 input differences and 33 output differences [5]. It turns out that the sum of the probabilities of those 561 differentials is not correct in [4]. When calculated correctly, however, the obtained probability is lower than the random probability, implying that this set of 561 differentials cannot be used in an attack. Even if we modify the attack of [4] to use only the best possible subset of those differentials, the resulting probability will be so close to the random probability that this attack on PRESENT will have much lower success probability than described in [4]. Finally, we compare our attack to the corrected version [6] of the attack of [4].

In order to evaluate the resistance of a block cipher to differential cryptanalysis, it is crucial to take into account the effect of combining multiple differentials. However, it is often not clear a priori which choice of differentials can actually lead to an improvement. Compared to classic differential cryptanalysis with one differential, a structure attack can obviously reduce the data complexity. In order to reduce the overall time complexity, however, the differentials have to be chosen carefully.

In this paper, we first present a general model for structure attacks, providing guidance on how to choose the differentials to minimize the time complexity. Secondly, we demonstrate structure attacks for 18-round PRESENT-80 with a data complexity of 2^{64} chosen plaintexts and time complexity of 2^{76} 18-round encryptions. We find that the properties of differentials in PRESENT cause structure attacks to be more efficient than the multiple differential cryptanalysis proposed in [4]. Thirdly, we improve the differential cryptanalytic result for the block cipher Serpent. In [3], Biham *et al.* describe a differential attack for 7-round Serpent with a data complexity of 2^{84} chosen plaintexts and a time complexity of 2^{85} memory accesses. Biham *et al.* also give a differential attack on 8-round Serpent-256 with 2^{213} memory accesses and 2^{84} chosen plaintexts. In our attack for 7-round Serpent, the data complexity is reduced to 2^{71} chosen plaintexts and the time complexity is $2^{74.99}$ encryptions. The attack can be further extended to 8-round Serpent-256. The time complexity is then increased to $2^{203.81}$ encryptions, with the data complexity remaining at 2^{71} chosen plaintexts.

For PRESENT-80, the best known attack is the linear hull cryptanalysis of 26-round PRESENT [8]. For Serpent-128, the best known cryptanalytic result is the differential-linear cryptanalysis on 12 rounds [12]. Although our attacks do not improve on those results for PRESENT and Serpent, to the best of our knowledge, they are the best *differential* attacks for PRESENT and Serpent. Moreover, our proposed attack model can be used to improve differential cryptanalytic results on other block ciphers as well.

This paper is organized as follows. Section 2 briefly describes the method for computing the success probability with multiple differentials. Section 3 introduces the structure attack model and the probability distribution of the key under multiple differentials. In Sect. 4, we demonstrate the attack for 18-round PRESENT. In Sect. 5, the improved attacks on 7-round and 8-round Serpent are presented. Section 6 concludes the paper.

2 Brief Description of Blondeau *et al.'s* Multiple Differential Cryptanalysis

In [4], Blondeau *et al.* propose multiple differential cryptanalysis using multiple differentials with different input differences and different output differences and give a precise analytical model to compute the success probability. In [18], Selçuk uses a Gaussian approximation of the binomial distribution to derive a formula for the success probability for differential cryptanalysis. Since then, his formula has been used in many papers on differential cryptanalysis. Blondeau *et al.* demonstrate that Selçuk's method cannot be applied to multiple differential cryptanalysis and express the distribution of key counters instead in terms of a hybrid distribution including the Kullback-Leibler divergence and a Poisson distribution [4]. Blondeau *et al.* obtain the following formula for the success probability P_S:

$$P_S \approx 1 - G_*[G^{-1}(1 - \tfrac{l-1}{2^{n_k}-2}) - 1], \tag{1}$$

where n_k is the number of key candidates, l is the size of the list to keep, G is defined by $G^{-1}(y) = \min\{x | G(x) \geq y\}$. The functions G and G_* are defined as follows:

$$G_*(\tau) \overset{\text{def}}{=} G(\tau, p_*) \quad \text{and} \quad G(\tau) \overset{\text{def}}{=} G(\tau, p),$$

where $p_* = \frac{\sum_{i,j} p_*^{(i,j)}}{|\Delta_0|}$ and $p = \frac{|\Delta|}{2^m |\Delta_0|}$. $p_*^{(i,j)}$ is the probability for the differential with the i-th input difference value and the j-th output difference value, m is the block size, $|\Delta_0|$ is the number of input difference values and $|\Delta|$ is the number of differentials. $G(\tau, p_*)$ and $G(\tau, p)$ can be calculated with the following equations:

$$G(\tau, q) \overset{\text{def}}{=} \begin{cases} G_-(\tau, q) & \text{if} \quad \tau < q - 3 \cdot \sqrt{q/N_s}, \\ 1 - G_+(\tau, q) & \text{if} \quad \tau > q + 3 \cdot \sqrt{q/N_s}, \\ G_{\mathcal{P}}(\tau, q) & \text{otherwise}, \end{cases} \tag{2}$$

where $G_{\mathcal{P}}(\tau, q)$ is the cumulative distribution function of the Poisson distribution with parameter qN_s, and N_s is the number of samples. $G_-(\tau, q)$ and $G_+(\tau, q)$ are defined as follows:

$$\begin{aligned} G_-(\tau, q) &\overset{\text{def}}{=} e^{-N_s D(\tau \| q)} \cdot [\tfrac{q\sqrt{1-\tau}}{(q-\tau)\sqrt{2\pi\tau N_s}} + \tfrac{1}{\sqrt{8\pi\tau N_s}}], \\ G_+(\tau, q) &\overset{\text{def}}{=} e^{-N_s D(\tau \| q)} \cdot [\tfrac{(1-q)\sqrt{\tau}}{(\tau-q)\sqrt{2\pi N_s(1-\tau)}} + \tfrac{1}{\sqrt{8\pi\tau N_s}}], \end{aligned} \tag{3}$$

where $D(\tau\|q)$ is the Kullback-Leibler divergence and is defined by:

$$D(\tau\|q) \overset{\text{def}}{=} \tau \ln\left(\frac{\tau}{q}\right) + (1-\tau)\ln\left(\frac{1-\tau}{1-q}\right).$$

On the assumptions for this analysis. In order to guarantee that each pair is counted only once, Blondeau *et al.* give Definition 1 as a necessary condition for the set of the input differences Δ_0.

Definition 1. *The set of input differences Δ_0 is admissible if there exists a set χ of $N/2$ plaintexts that fulfils the condition:*

$$\forall \delta_0^{(i)} \in \Delta_0, \forall x \in \chi, x \oplus \delta_0^{(i)} \notin \chi, \tag{4}$$

where N is the number of chosen plaintexts. However, this condition is so strong that many differentials will be excluded. For example, independent of the algorithm under consideration, the set of input differences $\Delta_0 = \{1_x, 2_x, 3_x\}$ is never admissible in any substitution-permutation network (SPN) because of this condition, since the overlapping bits of $3_x = 1_x \oplus 2_x$ will always result in double-counting.

By contrast, in the structure technique, we can use a hash table to exclude the duplicate pair arising from the violation of Definition 1. In fact, making use of hash tables, structure attacks can use more differentials while still ensuring that each pair is counted only once. Since we only have one possible output difference, this also enables the use of the complexity analysis of [4] for sets of plaintexts not satisfying Def. 1: This condition is only necessary to avoid counting both x and $x \oplus \delta_0^{(i)}$ for any $\delta_0^{(i)} \in \Delta_0$, i.e. guarantee $N_s = N|\Delta_0|/2$. This is satisfied in our approach, since each hash table will produce $N/2$ plaintext pairs with one input difference from N plaintexts, in total therefore $N_s = |\Delta_0|N/2$ plaintext pairs with $|\Delta_0|$ input diffference values. For structure attacks, the complexity analysis of [4] is therefore applicable independent of Def. 1.

This has additionally been verified by experiments on SmallPresent with block length of 24 bits, 12 rounds, and a set of 11 differentials with input differences violating Definition 1 and a single output difference.

On previous attacks on 18-round PRESENT. There are two previously published differential attacks on 18-round PRESENT [4, 6]. In this section, we point out two inconsistencies in both attacks, and demonstrate that our attack compares favourably to them.

In [4], a multiple differential attack for 18-round PRESENT is presented. They identify 561 differentials[5] including 17 input differences and 33 output differences using a branch-and-bound algorithm. In [4], the probabilities p_* and p are calculated as $p_* = \frac{\sum_{i,j} p_*^{(i,j)}}{|\Delta_0|} = 2^{-58.50}$ and $p = \frac{|\Delta|}{2^m|\Delta_0|} = 2^{-64} \cdot 33 = 2^{-58.96}$. However, the value of p_* is not correct; it should be $p_* = 2^{-60.39}$, which is less than the random probability for 33 output differences $p = 2^{-58.96}$. A possible remedy for this is to only choose some subset of the 561 differentials. We found that the best case is to choose four differentials with one input difference value 1001_x and the following four output difference values:

$$\{04040404_x||00000000_x, 00000404_x||00000000_x, 04000404_x||00000000_x, 00040404_x||00000000_x\}.$$

For this case, the success probability will take the maximum 68.08% for $n_k = 42, l = 2^{41}$ and $N = 2^{64}$. In this way, the attack for 18-round PRESENT-80 presented in [4] will have time

[5] These differentials have been obtained through private communication with Blondeau *et al.*

complexity 2^{79} and data complexity 2^{64} with a success probability of only 68.08%. The memory requirements are 2^{42} bytes for counters.

In [6], another multiple differential attack on 18-round PRESENT is presented. It can be seen from Table 4 of [6], that $|\Delta_0| = 17$ (and not 16 as assumed in the paper). This results in $p_* = 2^{-62.6765}$ (instead of $2^{-62.59}$) and $p = 2^{-63.56}$ (instead of p=$2^{63.47}$). Based on these values, we compare this attack to our attack from Sect. 4 for different values of the number ℓ of remaining key candidates (see Table 1). One can see that for the same data and time complexities, the structure attack performs consistently better than multiple differential cryptanalysis with multiple input differences and multiple output differences. This implies that PRESENT is not a good example to show the efficiency of multiple differential cryptanalysis with different input differences and different output differences.

Table 1. Comparison of our attacks on PRESENT with the multiple differential cryptanalysis of [6].

Attack of [6]		Attack of Sect. 4			
ℓ	P_S	ℓ	P_S	N	time complexity
2^{38}	65.27%	2^{36}	85.94%	2^{64}	2^{76}
2^{39}	79.68%	2^{37}	92.30%	2^{64}	2^{77}
2^{41}	94.62%	2^{39}	98.36%	2^{64}	2^{79}

3 Structure Attack

3.1 Principle of the attack

The structure attack is a form of differential cryptanalysis which uses multiple input differences and a single output difference. Structure attacks are a special case of multiple differential cryptanalysis, but their form allows for a dedicated attack procedure, which we describe in this section.

A structure attack is performed in three phases:

1. **Data Collection Phase:** Collect a large number of ciphertext pairs with the differences produced from the output difference of the differentials and the corresponding plaintext differences belong to the set of the input differences.
2. **Data Analysis Phase:** Derive the list of the best candidates for some key bits from the collected ciphertext pairs.
3. **Key Search Phase:** Search the list of candidates and all the corresponding master keys (*i.e.*, the unexpanded key from which the round subkeys are derived).

The idea of the structure attack is to use more differentials with multiple input differences and a single output difference to reduce the data complexity. However, the set of the input differences must be controlled in order to reduce the time complexity. This is done by organizing the plaintext in so-called *structures*:

Definition 2. *Let $\{\Delta_0^1, \ldots, \Delta_0^t\}$ be a set of t input differences. A collection of plaintexts of the form*

$$\bigcup_x \{x \oplus \Delta \mid \Delta \in \mathrm{span}\{\Delta_0^1, \ldots, \Delta_0^t\}\}, \tag{5}$$

with span *denoting the linear span operator, is called a* structure*.*

In this way, we can construct structures to produce the expected number of right pairs with lower data complexity compared with a single differential. Now we will give a model to choose the differentials to reduce the complexity. For clarity of exposition, we describe the model for the case of a substitution-permutation network (SPN); however, the concept can analogously be applied to other block cipher constructions, most importantly Feistel ciphers.

If we attack an R-round block cipher with $|\Delta_0|$ r-round differentials with a single output difference and multiple input differences, we denote these differentials as follows:

$$\Delta_0^i \xrightarrow{r} \Delta_r,\ Probability = p_i,\ (1 \leq i \leq |\Delta_0|),$$

where Δ_0^i and Δ_r are the i-th input difference and the output difference, respectively. The following notations are related with the attack:

- m: the block size of the block cipher.
- k: the key size of the block cipher.
- $|\Delta_0|$: the number of differentials.
- p_i: the probability of the differential with input difference Δ_0^i.
- N_{st}: the number of structures is $2^{N_{st}}$.
- N_p: the number of plaintexts bits involved in the active S-boxes in the first round for all differentials.
- N_c: the number of ciphertexts bits involved in the non-active S-boxes in the last round deriving from Δ_r.
- β: the filtering probability for the ciphertext pairs.
- p_f: the filtering probability for the ciphertext pairs according to active S-boxes, $p_f = \beta \cdot 2^{N_C}$.
- l: the size of the candidate list.
- n_k: the number of guessed subkey bits in the last $R - r$ rounds.

In the attack, $2^{N_{st}}$ structures are constructed. In each structure, all the input bits to non-active S-boxes in the first round are fixed to some random value, while N_p input bits of all active S-boxes take all 2^{N_p} possible values. There are $2^{N_{st}} \cdot 2^{N_p - 1} = 2^{N_{st}+N_p-1}$ pairs for each differential. We expect that about $2^{N_{st}+N_p-1} \cdot \sum_{i=1}^{|\Delta_0|} p_i$ pairs produce the output difference Δ_r. These pairs are right pairs.

The attack is described as follows.

1. For each structure:
   (a) Insert all the ciphertexts into a hash table indexed by N_c bits of the non-active S-boxes in the last round.
   (b) For each entry with the same N_c bits value, check whether the input difference is any one of the total $|\Delta_0|$ possible input differences. If a pair satisfies one input difference, then go to the next step.
   (c) For the pairs in each entry, check whether the output differences of active S-boxes in the last round can be caused by the input differences according to the differential distribution table. If the pair passes the test, then go to the next step.
   (d) Guess n_k bits subkeys to decrypt the ciphertext pairs to round r and check whether the obtained output difference at round r is equal to Δ_r. If so, add one to the corresponding counter.
2. Choose the list of the l best key candidates from the counters.
3. Search the list of candidates and all the corresponding master key.

Obviously the time complexity in step 2 is negligible, so we denote T_a, T_b, T_c, T_d and T_3 as the time complexity in step (a), (b), (c), (d) and 3, respectively, which are listed in following:

$$\begin{cases} T_a : 2^{N_{st}+N_p} \text{ memory accesses;} \\ T_b : 2^{N_{st}+2N_p-N_c} \text{ memory accesses;} \\ T_c : |\varDelta_0| \cdot 2^{N_{st}+N_p-N_c} \text{ memory accesses;} \\ T_d : |\varDelta_0| \cdot 2^{N_{st}+N_p-N_c} \cdot p_f \cdot 2^{n_k} \text{ partial decryptions;} \\ T_3 : l \cdot 2^{k-n_k}. \end{cases}$$

This assumes that there are n_k independent subkey bits from the key schedule. In general, T_d can be approximated by $|\varDelta_0| \cdot 2^{N_{st}+N_p-N_c} = T_c$. Since $|\varDelta_0| < 2^{N_p}$, we have $T_c < T_b$. Then the whole time complexity can be expressed as follows:

$$T_a + T_b + T_c + T_d + T_3 \simeq \begin{cases} T_a + T_3 & \text{if } N_p < N_c, \\ T_b + T_3 & \text{if } N_p > N_c, \\ 2T_a + T_3 = 2T_b + T_3 & \text{if } N_p = N_c. \end{cases}$$

If the time complexity in the key searching process T_3 is much smaller than the time complexity of the data collection process and the data analysis process, we will take $N_p = N_c$ to minimise the whole time complexity as the minimum value $2T_a$. Otherwise, we can try to take a larger value for N_p to increase the sum of the probabilities for differentials to further reduce the data complexity.

It is worth noting that in the structure attack, any pair of plaintexts with the given input difference is only counted once. In this way, the number of input differences can be increased compared with the condition in Definition 1, improving the efficiency of the attacks. This is especially applicable in an attack scenario where the probability of many differentials are close to 2^{-m}, implying a low success rate P_S. Therefore, a large value for l has to be chosen, which causes the complexity T_3 of step 3 to increase. In this case, increasing the number of input differences can help improving the attack, whereas increasing the number of output differences does not have this effect in the case of multiple differential cryptanalysis.

In the case of reduced-round PRESENT, we have the above-mentioned scenario (many differentials with probability close to 2^{-64}), so that when choosing our set of differentials, we only include a limited number of high-probability differentials to maintain a good success probability P_S. For reduced-round Serpent, the probabilities of the differentials are much larger than 2^{-128} (the inverse of the block size), so that we can choose more differentials here without affecting the success probability. In order to minimize the time complexity, we choose $N_p = N_c$ according to our model.

3.2 Ratio of Weak Keys for Multiple Differentials

In general, the differential probability is related to the value of the key. As we use multiple differentials in the structure attack, we need to consider the ratio of keys which can produce the expected number of right pairs. We call those keys *weak keys* since the attacks are only expected to work for those.

A cipher is called *key-alternating* if it consists of an alternating sequence of unkeyed rounds and simple bitwise key additions. Note that most block cipher proposals, including PRESENT and Serpent, are key-alternating ciphers. The fixed-key cardinality of a differential $N[K](a, b)$ is the number of pairs with input difference a and output difference b where the key K is fixed to a specific value. In [11, 10], Daemen and Rijmen give the following theorem.

Theorem 1. *Assuming that the set of pairs following a characteristic for a given key can be modeled by a sampling process, the fixed-key cardinality of a differential in a key-alternating cipher is a stochastic variable with the following distribution:*

$$\Pr(N[K](a,b)=i) \approx \text{Poisson}\left(i, 2^{m-1}EDP(a,b)\right),$$

where m is the block size, $EDP(a,b)$ denotes the expected differential probability of the differential (a,b), and the distribution function measures the probability over all possible values of the key and all possible choices of the key schedule.

For multiple differentials with multiple input differences and a single output difference, we have $p_j = EDP(a_j,b), 1 \leq j \leq |\Delta_0|$. We denote the fixed-key cardinality of multiple differentials (a_j,b) with a single output difference b by $N[K]\{(a_j,b)\}_j$. Based on Theorem 1, we can now derive Theorem 2.

Theorem 2. *Under the assumptions of Theorem 1, in a key-alternating cipher, the fixed-key cardinality of multiple differentials is a stochastic variable with the following distribution:*

$$\Pr\left(N[K]\{(a_j,b)\}_j = i\right) \approx \text{Poisson}\left(i, 2^{m-1}\sum_j EDP(a_j,b)\right),$$

where the distribution function measures the probability over all possible values of the key and all possible choices of the key schedule.

Proof. The cardinality of multiple differentials equals the sum of the cardinalities of each differential (a_j,b) for the iterative cipher, so we have

$$N[K]\{(a_j,b)\}_j = \sum_j N[K](a_j,b).$$

From Theorem 1, the cardinality for each differential (a_j,b) has Poisson distribution. Making the standard assumption that the cardinalities of the differentials are independent random variables, the sum still is Poisson distributed with as λ-parameter the sum of the λ-parameters of the terms:

$$\lambda = \sum_j 2^{m-1}EDP(a_j,b).$$

□

From Theorem 2, in the structure attack based on the differentials

$$\Delta_0^i \xrightarrow{r} \Delta_r,\ Probability = p_i,\ (1 \leq i \leq |\Delta_0|),$$

the ratio of the weak keys r_w that can produce more than or equal to μ right pairs can be computed as follows:

$$r_w = 1 - \sum_{x=0}^{\mu-1} \text{Poisson}\left(x, 2^{m-1}\sum_{j=1}^{|\Delta_0|} p_i\right).$$

Note that when evaluating the ratio of weak keys, we have a different setting than when dealing with the distribution of the counters in a (multiple) differential attack. While approximating the distribution of the counters with either normal or Poisson distributions was shown to be problematic for accurately estimating the tails [19, 4], the distribution of the weak keys instead depends on the *cardinality* of the multiple differentials. In this setting, using the Poisson distribution as in Theorem 2 also yields a good approximation for the tails. This was also experimentally verified with small-scale variants of the block cipher PRESENT [14], with block lengths ranging from 8 to 24 bits.

Additionally, the accuracy of the weak key ratio r_w based on Theorem 2 has been verified by experiments on SMALLPRESENT with a block length of 24 bits, 12 rounds and an master key with 8 bit entropy. 7 differentials with 7 different input and a single output difference were used. The λ-parameter of the Poisson distribution was $2^{23} \cdot \left(5 \cdot 2^{-23} + 2 \cdot 2^{-22}\right) = 2^{3.17}$. The distribution of the ratio of weak keys for different values of μ is listed in Table 2. The experimental results very closely follow the theoretical estimate.

Table 2. Theoretical and experimental weak key ratio for SMALLPRESENT-24.

μ	2	4	6	8	16
theoretical r_w	0.9988	0.9788	0.8843	0.6762	0.0220
experimental r_w	1	0.98	0.89	0.68	0.02

4 Attack on 18-Round PRESENT

The block cipher PRESENT is designed as a very lightweight cipher. It has a 31-round SPN structure in which the S-box layer has 16 parallel 4-bit S-boxes and the diffusion layer is a bit permutation [7]. The block size is 64 bits and the key size can be 80 bits or 128 bits. One round of PRESENT is illustrated in Fig. 1.

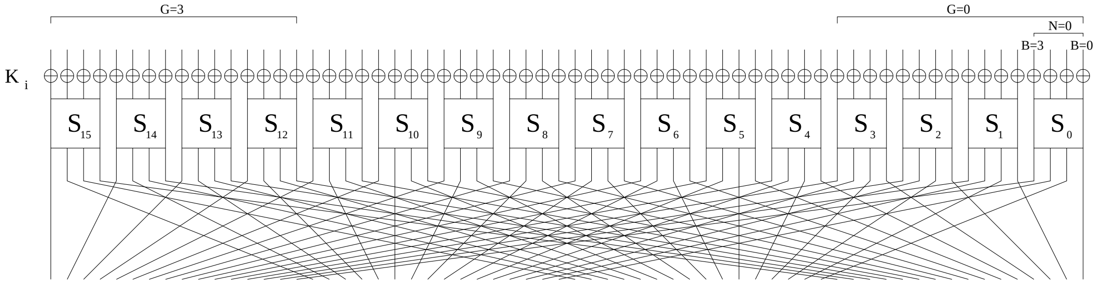


Fig. 1. One round of the PRESENT block cipher.

PRESENT has been extensively analyzed. Wang presents a differential attack on 16-round PRESENT [20]. Collard *et al.* give a statistical saturation attack for 24-round PRESENT [9]. There are three papers about attacks based on linear hulls for PRESENT [8, 17, 16], leading to linear attacks for up to 26 rounds. Since the S-box of PRESENT admits linear approximations with single-bit linear masks, the attacker can exploit linear hulls containing many single-bit linear trails over an arbitrary number of rounds. However, for differential attacks, we have to use paths in which two active S-boxes appear per round. Hence, a linear attack will typically be more efficient than differential attacks. As explained in Sect. 2, Blondeau *et al.* use multiple differentials to attack 18 rounds of the PRESENT block cipher. However, as outlined in Sect. 2, this attack does not work as described. By analyzing the properties of the differentials of PRESENT, we have found that the structure attack for PRESENT is more efficient than the multiple differential attack.

In order to identify a differential with high probability, we must collect more differential paths with high probability for a differential. The differential paths with two active S-boxes in every round have a much bigger contribution to the differential, so we will focus on differential paths with only two active S-boxes in each round. Using those differential paths, a high-probability differential can be identified. Then we can choose more differentials to improve the attack according to the formulas for the overall time complexity described in Sect. 3, .

4.1 Searching Differential Paths for PRESENT

We now give a method to search all differential characteristics with two active S-boxes in each round which have higher probability compared with other differential paths.

First, we introduce some notation. The block size of PRESENT is 64 bits and we can divide 16 nibbles into four groups, in each of which there are four nibbles. We define G as the index of a group, so the four least significant nibbles belong to the group $G = 0$ and the four most significant nibbles belong to the group $G = 3$. We denote the index of a nibble as N, and in each group the least significant nibble is $N = 0$ and the most significant nibble is the nibble with $N = 3$. In each nibble, we denote B as the B-th bit, the least significant bit is $B = 0$ and the most significant bit is $B = 3$. In this way, the position of any bit can be denoted by a triple (G, N, B), as also illustrated in Fig. 1. The permutation layer P is computed as follows,

$$P(16 \cdot G + 4 \cdot N + B) = 16 \cdot B + 4 \cdot G + N, 0 \leq G, N, B \leq 3.$$

After the permutation layer P, the bit (G, N, B) will be transferred to the bit (B, G, N). Here we also give another triple (G, N, V) where G and N are the group index and nibble index, respectively, while V is the difference of the nibble. We will use the following notation:

- $(G_{r,k}, N_{r,k}, B_{r,k})$: The position of the k-th ($k = 1, 2, 3, 4$) output bit for S-box in round r.
- $(G_{r,k}, N_{r,k}, V_{r,k})$: The output difference value of the k-th ($k = 1, 2$) active S-box for nibble $(G_{r,k}, N_{r,k})$ in round r.

We focus on finding differential characteristics with two active S-boxes in each round. The foundation for this search is formulated in Theorem 3. Next an efficient searching algorithm for the differential characteristics with two active S-boxes per round will be presented.

Theorem 3. *For the PRESENT block cipher, differential characteristics with only two active S-boxes per round must have the following pattern:*

1. *If two active S-boxes are in the same group in round r, their output difference will be equal and must have two non-zero bits to ensure that only two active S-boxes appear in the $(r+2)$-nd round, and two active S-boxes in round $r + 1$ will be in the different groups;*
2. *If two active S-boxes are in different groups in round r, their output difference will be equal and must have only one non-zero bit to ensure that only two active S-boxes appear in the $(r + 1)$-st round, and two active S-boxes in round $r + 1$ will be in the same group.*

For the proof of Theorem 3, see Appendix A. With Theorem 3, we give the searching algorithm for the differential paths in Fig. 3 in the appendix.

Using Algorithm in Fig. 3, we search for 16-round differential paths (characteristics) with two active S-boxes in each round having a probability greater than 2^{-92}. In total, we find 139 *differentials* with probability greater than 2^{-64}, among which 91 differentials have output difference $\Delta_{16} = 00000500_x||00000500_x$ and 18 differentials have output difference $\Delta_{16} = 00000900_x||00000900_x$. We list them in Table 5 and Table 6, respectively. The differentials have been ordered according to their probabilities in these two tables. In both Table 5 and Table 6, the first column i contains the number of the differential, Δ_0^i is the input difference and p_i is the probability for each differential. At the same time, we list the remaining 30 differentials with different output difference values in Table 7. Moreover, we present the number of differential paths with different probability for Table 5 in Table 8 and Table 9. In Table 8, the first column denotes the index number in the first column of Table 5. For example, the differentials with number 19 and 20 consist of differential trails with the same probabilities. Columns $2, 3, \ldots, 12$ denote the number of differential paths with probability $2^{-71}, 2^{-73}, \ldots, 2^{-91}$, respectively. In Table 9, the first column denotes the index number in the first column of Table 5. Column $2, 3, \ldots, 13$ denote the number of differential paths with probability $2^{-70}, 2^{-72}, \ldots, 2^{-92}$, respectively. There is no differential path with probability greater than 2^{-70} or less than 2^{-92} for the 91 differentials.

4.2 Key Recovery Attack on 18-Round PRESENT-80

In this section, we show how to use the 16-round differentials listed in Table 5 to attack 18-round PRESENT-80. The first step is to choose the set of differentials. From the output difference $00000500_x||00000500_x$ at round 16, we can derive that the number of recovered subkey bits in round 17 and round 18 is $8 + 32 = 40$. Those 40 subkey bits are independent according to the key schedule. In this attack, we will use the whole codebook and set the size of the candidates of subkey counters l to 2^{36}. In our structure attack, we will use Blondeau *et al.*'s method (see Sect. 2) to compute the success rate. With Equation (1), we have $n_k = 40$, $l = 2^{36}$ and $N = 2^{64}$. We gradually increase the number of differentials with higher probability from Table 5 to compute the success probability for every case. As a result, we found that the success rate will increases as $|\Delta_0| = i$ increases if $1 \leq i \leq 36$. The success probability is 85.95% as $|\Delta_0| = 36$. If we add the i-th ($37 \leq i \leq 91$) differential to the set, the success probability will be reduced. This implies that the i-th ($37 \leq i \leq 91$) differential has no contribution to reduce the data complexity since its probability is too low. Therefore, in our attack, we will only use the first 36 differentials in Table 5.

If we use multiple differentials cryptanalysis for PRESENT following Blondeau *et al.*, we can choose more output difference values. We can add the 18 differentials in Table 6 to the set of 36 differentials. The input difference values for the 18 differentials belong to the set of the input difference values for the 36 differentials, so we have $|\Delta_0| = 36$ and $|\Delta_{16}| = 2$. Then we get $p_* = 2^{-62.74}$ and $p = 2^{-63}$. As τ ($p < \tau < p_*$) increases, $G(\tau, p)$ will decrease. Even if we take $\tau = p_*$, $G(\tau, p)$ is still larger than $(1 - \frac{l-1}{2^{n_k}-2})$, so the attack will not work for $l = 2^{36}$. Therefore, our structure attack works better for PRESENT than the multiple differential cryptanalysis presented in [4].

Moreover, we have identified the differential trails with two active S-boxes per round but more than two active S-boxes in the last round. As a result, those differentials have no advantage compared with the differentials in Table 5. Therefore, these differentials do not contribute to improving multiple differential cryptanalysis for PRESENT.

We will use the structure attack for 18-round PRESENT-80 with the first 36 differentials with $p_* = 2^{-63.14}$ and $p = 2^{-64}$. For the 36 input differences, there are 10 active S-boxes in the first round which are nibbles 0, 1, 2, 3, 4, 8, 12, 13, 14 and 15, so the S-boxes for the nibbles 5, 6, 7, 9, 10 and 11 are all non-active.

We construct 2^{24} structures of 2^{40} chosen plaintexts each. In each structure, all the inputs to the 6 non-active S-boxes in the first round take a fixed random value, while 40 bits of input to 10 active S-boxes take 2^{40} possible values. In all structures, there are $2^{24} \cdot 2^{39} = 2^{63}$ pairs for each possible differential. The sum of the probabilities for all 36 differentials is $2^{-57.97}$, so the number of right pairs is $2^{63} \cdot 2^{-57.97} = 2^{5.03}$.

According to the output difference of 16-round differentials, there are two active S-boxes in round 17 in nibble 2 and 10 whose input difference is 5 and the possible output differences will be 1, 4, 9, 10, 11, 12 or 13. After the bit permutation, 8 output bits from the two active S-boxes in round 17 will be one input bit to 8 different S-boxes in round 18 respectively. As the number of non-zero bits among the 8 output bits is at most 6, the maximum number of active S-boxes for round 18 is 6 and the minimum number of active S-boxes for round 18 is 2. We denote the number of active S-boxes in round 18 as N_a ($2 \le N_a \le 6$), the output difference for the j-th S-box in round i as $Y_{i,j}$, the filter probability with N_a active S-boxes in round 18 as $p_f^{(a)}$. We present the filter probability for different values of N_a in Table 3. The filter probability for the ciphertext pairs β according to active S-boxes can be computed with the sum of column 3 in Table 3, and we get $\beta = 2^{-12.55}$.

Table 3. Filter probability for the structure attack on 18-round PRESENT.

N_a	$(Y_{17,2}, Y_{17,10})$	$p_f^{(a)}$
2	$\{(1,1),(1,4),(4,1),(4,4)\}$	$2^{-24} \cdot (\frac{7}{16})^2 \cdot 4 = 2^{-24.83}$
3	$\{(1,9),(1,10),(1,12),(4,9),$ $(4,10),(4,12),(9,1),(9,4),$ $(10,1),(10,4),(12,1),(12,4))\}$	$2^{-20} \cdot (\frac{7}{16})^3 \cdot 12 = 2^{-19.99}$
4	$\{(9,9),(9,10),(9,12),(10,9),$ $(10,10),(10,12),(12,9),(12,10),$ $(12,12),(1,11),(1,13),(4,11),$ $(4,13),(11,1),(11,4),(13,1),(13,4)\}$	$2^{-16} \cdot (\frac{7}{16})^4 \cdot 17 = 2^{-16.68}$
5	$\{(9,11),(9,13),(10,11),(10,13),$ $(12,11),(12,13),(11,9),(11,10),$ $(11,12),(13,9),(13,10),(13,12)\}$	$2^{-12} \cdot (\frac{7}{16})^5 \cdot 12 = 2^{-14.38}$
6	$\{(11,11),(11,13),(13,11),(13,13)\}$	$2^{-8} \cdot (\frac{7}{16})^6 \cdot 4 = 2^{-13.16}$

We now describe in detail the attack procedure of Sect. 3 for 18-round PRESENT-80. We have $|\Delta_0| = 36$, $\sum_{i=1}^{|\Delta_0|} p_i = 2^{-57.97}$, $N_{st} = 24$, $N_p = 40$, $N_c = 32$, $\beta = 2^{-12.55}$, $p_f = 2^{-44.55}$, $n_k = 40$ and $l = 2^{36}$.

We denote T_a, T_b, T_c, T_d and T_3 as the time complexity in step (a), (b), (c) (d) and 3, respectively, which are as follows:

$T_a : 2^{64}$ memory accesses;
$T_b : 2^{72}$ memory accesses;
$T_c : 36 \cdot 2^{32}$ memory accesses;
$T_d : 36 \cdot 2^{31} \cdot 2^{-12.55} \cdot 2^{40} \cdot (\frac{1}{2} + \frac{1}{8}) \cdot 2 = 2^{65.20}$ 1-round encryptions.
$T_3 : 2^{36} \cdot 2^{40} = 2^{76}$ 18-round encryptions.

Therefore, the total time complexity will be 2^{76} 18-round encryptions. The data complexity is 2^{64} chosen plaintexts and the memory requirements are 2^{40} 128-bit cells for the hash table, which can be reused for the 2^{40} counters. The success probability is 85.95%.

The ratio of weak key satisfying the sum of the probabilities of the 36 differentials is computed as follows:

$$r_w = 1 - \sum_{x=0}^{\mu-1} \text{Poisson}\left(x, 2^{n-1} \sum_{j=1}^{N_d} p_i\right) = 1 - \sum_{x=0}^{2^{5.03}-1} \text{Poisson}\left(x, 2^{63} \cdot 2^{-57.97}\right) = 0.57.$$

This means that the number of weak keys for which our attack can succeed is $2^{80} \cdot 0.57 = 2^{79.19}$ for PRESENT-80. A comparison with the attack of [6] can be found in Table 1.

5 Attack on Reduced-Round Serpent

Serpent was one of the five AES candidates in the final round; it is an SPN block cipher with 32 rounds [1]. Fig. 2 depicts Serpent reduced to 8 rounds, from round 4 to 11.

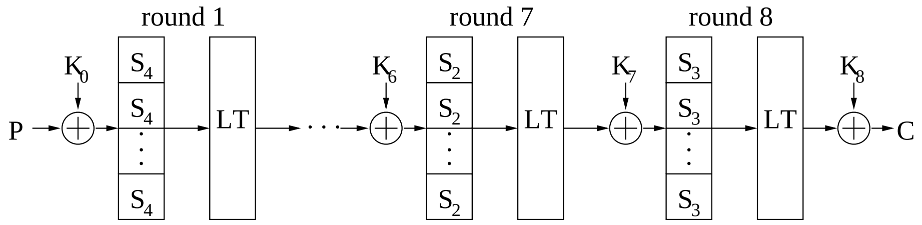


Fig. 2. The block cipher Serpent reduced to 8 rounds.

In the previous differential cryptanalysis of Serpent in [3], Biham *et al.* used the structure attack for Serpent. They identify a differential characteristic for $\frac{1}{2}+5$ rounds staring from the linear transformation with fewer active S-boxes (13 active S-boxes) in the first half round, then extend it backwards to 6 rounds. Moreover, there is only one differential characteristic in each differential due to the strong avalanche characteristics of Serpent. Biham *et al.* claim that 2^{14} differential characteristics with probability 2^{-93} have been found. However, it can be shown that there are only 2^{13} differential characteristics with probability 2^{-93}. The proof has been omitted due to space constraints.

For the differential characteristics, the output difference of S-boxes in the first round is $\{0906b010_x||00000080_x||13000226_x||06040030_x\}$. As the first round uses S_4 S-boxes, the partial differential distribution table for S_4 is listed in Table 4. We will use all the possible non-zero input differences according to the output differences for the S-boxes in the first round. So we have $|\Delta_0| = 2^{35.32}$ and $\sum_{i=1}^{|\Delta_0|} p_i = 2^{-65}$ which is equal to the probability of the differential characteristic from round 2 to round 6.

Table 4. Partial differential distribution table for S_4

out \in	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
1_x	0	0	0	2	0	2	0	0	0	2	0	4	2	2	2	0
2_x	0	0	0	2	0	0	0	2	0	4	2	0	0	4	2	0
3_x	0	0	2	0	4	2	2	2	2	0	0	0	0	0	2	0
4_x	0	0	0	2	0	0	4	2	0	2	0	0	0	2	0	4
6_x	0	0	4	0	0	0	0	0	0	2	2	0	0	2	2	4
8_x	0	0	0	2	0	2	2	2	0	0	2	0	2	0	2	2
9_x	0	0	2	2	0	2	2	0	2	2	0	0	0	2	2	0
B_x	0	2	0	2	4	0	0	0	2	0	2	4	0	0	0	0

We now apply the structure attack described in Sect. 3. We construct 2^{19} structures of 2^{52} chosen plaintexts each. In each structure, all the inputs to non-active S-boxes in the first round are fixed to some random value, while the 52 bits of input to all the active S-boxes take all the 2^{52} possible values. There are $2^{19} \cdot 2^{51} = 2^{70}$ pairs for each differential characteristic. We expect that about $2^{70} \cdot 2^{-65} = 2^5$ pairs produce the output difference Δ_6. In order to reduce the time complexity and ensure a higher success probability, 52 bits subkey are guessed after the data collection process. After retrieving 52 bits of the subkey, we can use the right pairs to recover the remaining 24 bits of the subkey.

The success probability P_S can be computed with Equation (1). Here $N = 2^{71}$, $|\Delta_0| = 2^{35.32}$, $p_* = 2^{-65} \cdot 2^{-35.32} \cdot 2^{52} = 2^{-48.32}$, $N_s = 2^{70} \cdot 2^{35.32} \cdot 2^{-52} = 2^{53.32}$, $p = 2^{-52}$, $n_k = 52$, $l = 2$, $\beta = 2^{-26.22}$, hence we get $P_S = 89.87\%$.

The time complexity is $2^{27.10} \cdot 2^{52} \cdot 13/32 = 2^{77.81}$ one-round encryptions which is equivalent to $2^{74.99}$ 7-round encryptions, the data complexity is 2^{71} chosen plaintexts and the memory requirements are 2^{52} hash cells of 256 bits and 2^{52} 32-bit counters storing 2^5 pairs each, hence using about 2^{57} 256-bit words. This attack consequently applies to Serpent with all key sizes of 128,192 and 256 bits.

The attack can be further extended to 8-round Serpent-256. By exhaustively searching the 128-bit subkey in the last round to decrypt to round 7, the above attack for 7 rounds can be applied. The time complexity is $2^{203.81}$ 8-round encryptions, the data complexity is 2^{71} chosen plaintexts and the memory requirements are the same as for the 7-round attack. This attack therefore applies only to Serpent with a 256-bit key.

In comparison, the previous differential attack for 7-round Serpent described in [3] has a time complexity of 2^{85} memory accesses and a data complexity of 2^{84} chosen plaintexts. For the previous differential attack on 8-round Serpent, the time complexity is 2^{213} memory accesses and the data complexity is 2^{84} chosen plaintexts. This implies that our attacks require much less chosen plaintexts. Under the assumption that in this case, a seven-round encryption is roughly equivalent to $243/2 \cdot 7/32 = 2^{4.7}$ memory accesses [15], our attacks also slightly reduce the time complexity.

It is possible to further reduce the data requirements at the expense of the time complexity. We have identified another set of differentials for 5.5 rounds which have 16 instead of 13 active S-boxes in the first round (the sequence of active S-Boxes is 16–10–6–2–1–5, and there are $2^{41.49}$ input differences). The combined probability of these differentials is $2^{-62.85}$, leading to a total time complexity greater than the previously described attack.

The ratio of weak keys satisfying the probability of the multiple differentials is computed as follows:

$$r_w = 1 - \sum_{x=0}^{\mu-1} \text{Poisson}\left(x, 2^{n-1}\sum_{j=1}^{N_d} p_i\right) = 1 - \sum_{x=0}^{2^5-1} \text{Poisson}\left(x, 2^{70} \cdot 2^{-65}\right) = 0.52.$$

This means that this attack is expected to work with about half of all possible keys, independent of the key size.

6 Conclusion

Modern block ciphers are designed to withstand differential cryptanalysis, as it is one of the most important cryptanalytic methods. Usually, the resistance to this attack is evaluated based on bounding the probabilities of differential characteristics; sometimes this is extended to the case of differentials. However, bounding the probability of a single differential path or a single differential is not sufficient to demonstrate resistance to differential cryptanalysis. The case of multiple differentials has to be considered as well. In this paper, we give a general model for the structure attack, providing guidance on how to choose the set of differentials to minimize the time complexity. As concrete applications of our model, we present structure attacks on 18-round PRESENT and improve the previous differential cryptanalytic results for the Serpent block cipher. To the best of our knowledge, those attacks are the best known differential attacks on these two block ciphers.

Comparing our model for structure attacks against the general model for multiple differential cryptanalysis proposed in [4], we conclude that the limitation for the set of input differences imposed by the model of [4] excludes many valuable differentials. We show that in structure attacks, a very important – and often particularly efficient – subclass of multiple differential attacks, this restriction can be relaxed. In our model presented in Sect. 3, the analysis of an attack can be carried out without this assumption.

The relevance of the limitation imposed by the condition of Definition 1 is additionally supported by our concrete application of the structure attack to PRESENT, which is more efficient than the multiple differential cryptanalysis with different output differences described in [4] and [6] where this condition was necessary. By removing this limitation, we have identified new sets of differentials that improve on the previous analysis.

It remains an interesting open question to find a block cipher other than PRESENT for which multiple differential cryptanalysis with multiple output differences produces superior results to the structure attack.

Furthermore, our attack model can be used as a guidance to improve differential attacks for other algorithms. Applying it to other block ciphers than PRESENT or Serpent will be subject of future work.

Acknowledgements. We would like to thank Vincent Rijmen for valuable advice and Kerem Varici for his support for the work on PRESENT. The work in this paper was supported in part by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II and in part by the Concerted Research Action (GOA) TENSE 2011 of the Flemish Government, the IAP Program P6/26 BCRYPT of the Belgian State (Belgian Science Policy) and by the Research Fund KU Leuven (project OT/08/027).

References

1. R. Anderson, E. Biham, and L.R. Knudsen. A Proposal for the Advanced Encryption Standard. NIST AES proposal, 1998.
2. E. Biham, and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. Journal of Cryptology, 4(1), pp. 3–72, 1991.
3. E. Biham, O. Dunkelman, and N. Keller. The Rectangle Attack — Rectangling the Serpent. EUROCRYPT 2001, LNCS 2045, pp. 340–357, Springer-Verlag, 2001.
4. C. Blondeau, and B. Gérard. Multiple Differential Cryptanalysis: Theory and Practice. FSE 2011, LNCS 6733, pp. 35–54, Springer-Verlag, 2011.
5. C. Blondeau, and B. Gérard. Private communication: The 561 Differentials. 2011.
6. C. Blondeau and B.Gérard. Multiple Differential Cryptanalysis: Theory and Practice (Corrected). Cryptology ePrint Archive: Report 2011/115.
7. A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: an Ultra-Lightweight Block Cipher. CHES 2007, LNCS 4727, pp. 450–466 Springer-Verlag, 2007.
8. J. Cho. Linear Cryptanalysis of Reduced-Round PRESENT. CT-RSA 2010, LNCS 5985, pp. 302–317, Springer-Verlag, 2010.
9. B. Collard, and F.-X. Standaert. A Statistical Saturation Attack against the Block Cipher PRESENT. CT-RSA 2009, LNCS 5473, pp. 195–210, Springer-Verlag, 2009.
10. J. Daemen and V. Rijmen. Probability distributions of correlations and differentials in block ciphers. Journal of Mathematical Cryptology 1(3), pp. 221-242, 2007.
11. J. Daemen, and V. Rijmen. Probability distributions of Correlation and Differentials in Block Ciphers. http://eprint.iacr.org/2005/212, 2005.
12. O. Dunkelman, S. Indesteege, and N. Keller. A Differential-Linear Attack on 12-Round Serpent. INDOCRYPT 2008, LNCS 5365, pp 308–321, Springer-Verlag, 2008.
13. X. Lai, J.L. Massey, and S. Murphy. Markov Ciphers and Differential Cryptanalysis. Advances in Cryptology-Eurocrypt'91, LNCS 547, pp. 17–38, Springer-Verlag, 1991.
14. G. Leander. Small scale variants of the block cipher PRESENT. Cryptology ePrint Archive, Report 2010/143, 2010.
15. M. Matsui and J. Nakajima. On the Power of Bitslice Implementation on Intel Core2 Processor. CHES 2007, LNCS 4727, pp. 121–134, Springer-Verlag, 2007.
16. J. Nakahara, P. Sepehrdad, B. Zhang, and M. Wang. Linear (Hull) and Algebraic Cryptanalysis of the Block Cipher PRESENT. CANS 2009, LNCS 5888, pp. 58–75, Springer-Verlag, 2009.
17. K. Ohkuma. Weak keys of Reduced-Round PRESENT for Linear Cryptanalysis. SAC 2009, LNCS 5867, pp. 249-265, Springer-Verlag, 2009.
18. A.A. Selçuk, and A. Bicak. On Probability of Success in Linear and Differential Cryptanalysis. SCN 2002, LNCS 2576, pp. 174–185, Springer-Verlag, 2003.
19. A.A. Selçuk. On Probability of Success in Linear and Differential Cryptanalysis. Journal of Cryptology, Volume 21(1), pp. 131–147, Springer-Verlag, 2008.
20. M. Wang. Differential Cryptanalysis of reduced-round PRESENT. AfricaCrypt 2008, LNCS 5023, pp. 40–49, Springer-Verlag, 2008.

A Proof of Theorem 3

Proof. The output differences for the two active S-boxes are $(G_{r,1}, N_{r,1}, V_{r,1})$ and $(G_{r,2}, N_{r,2}, V_{r,2})$. First, we will prove the case for two active S-boxes in the same group in round r. We have $G_{r,1} = G_{r,2}$ and $N_{r,1} \neq N_{r,2}$.

1. $V_{r,1} \in \{1, 2, 4, 8\}$**:** If $V_{r,2} \in \{1, 2, 4, 8\}$, we denote their two non-zero bits as

   $$\{(G_{r,1}, N_{r,1}, B_{r,1}), (G_{r,1}, N_{r,2}, B_{r,2})\}.$$

   We have

   $$\begin{aligned}&\{(G_{r,1}, N_{r,1}, B_{r,1}), (G_{r,1}, N_{r,2}, B_{r,2})\} \xrightarrow{P} \{(B_{r,1}, G_{r,1}, N_{r,1}), (B_{r,2}, G_{r,1}, N_{r,2})\} \xrightarrow{S}\\ &\{(B_{r,1}, G_{r,1}, N_{r+1,1}), (B_{r,1}, G_{r,1}, N_{r+1,2}), (B_{r,2}, G_{r,1}, N_{r+1,3}), (B_{r,2}, G_{r,1}, N_{r+1,4})\}\\ &\xrightarrow{P} \{(N_{r+1,1}, B_{r,1}, G_{r,1}), (N_{r+1,2}, B_{r,1}, G_{r,1}), (N_{r+1,3}, B_{r,2}, G_{r,1}), (N_{r+1,4}, B_{r,2}, G_{r,1},)\}.\end{aligned}$$

   As there are two active S-boxes in round $r{+}1$, we have $B_{r,1} \neq B_{r,2}$. Because bit $N_{r+1,1}$ and bit $N_{r+1,2}$ are from the same S-box, we have $N_{r+1,1} \neq N_{r+1,2}$. Similarly, we have $N_{r+1,3} \neq N_{r+1,4}$. There will be four active S-boxes in the $(r{+}2)$-nd round. If $V_{r,2} \in \{3, 5, 6, 9, 10, 12\}$, we denote the three non-zero bits as

   $$\{(G_{r,1}, N_{r,1}, B_{r,1}), (G_{r,1}, N_{r,2}, B_{r,2}), (G_{r,1}, N_{r,2}, B_{r,3})\}.$$

   We have

   $$\begin{aligned}&\{(G_{r,1}, N_{r,1}, B_{r,1}), (G_{r,1}, N_{r,2}, B_{r,2}), (G_{r,1}, N_{r,2}, B_{r,3})\}\\ &\xrightarrow{P} \{(B_{r,1}, G_{r,1}, N_{r,1}), (B_{r,2}, G_{r,1}, N_{r,2}), (B_{r,3}, G_{r,1}, N_{r,2}) | B_{r,1} = B_{r,2} \neq B_{r,3}\}\\ &\xrightarrow{S} \{(B_{r,1}, G_{r,1}, N_{r+1,1}), (B_{r,3}, G_{r,1}, N_{r+1,2}), (B_{r,3}, G_{r,1}, N_{r+1,3}) | N_{r+1,2} \neq N_{r+1,3}\}\\ &\xrightarrow{P} \{(N_{r+1,1}, B_{r,1}, G_{r,1}), (N_{r+1,2}, B_{r,3}, G_{r,1}), (N_{r+1,3}, B_{r,3}, G_{r,1})\}.\end{aligned}$$

   There will be three active S-boxes in round $r + 2$.
2. $V_{r,1} \in \{7, 11, 13, 14, 15\}$ **or** $V_{r,2} \in \{7, 11, 13, 14, 15\}$**:** There will be at least three active S-boxes in round $r + 1$.
3. $V_{r,1}, V_{r,2} \in \{3, 5, 6, 9, 10, 12\}$**:** We denote the four non-zero bits as $\{(G_{r,1}, N_{r,1}, B_{r,1}), (G_{r,1}, N_{r,1}, B_{r,2}), (G_{r,1}, N_{r,2}, B_{r,3}), (G_{r,1}, N_{r,2}, B_{r,4})\}$.
   We have

   $$\begin{aligned}&\{(G_{r,1}, N_{r,1}, B_{r,1}), (G_{r,1}, N_{r,1}, B_{r,2}), (G_{r,1}, N_{r,2}, B_{r,3}), (G_{r,1}, N_{r,2}, B_{r,4})\}\\ &\xrightarrow{P} \{(B_{r,1}, G_{r,1}, N_{r,1},)(B_{r,2}, G_{r,1}, N_{r,1}), (B_{r,3}, G_{r,1}, N_{r,2}), (B_{r,4}, G_{r,1}, N_{r,2})\}.\end{aligned}$$

   Only if $B_{r,1} = B_{r,3}$ and $B_{r,2} = B_{r,4}$, there will be 2 active S-boxes in round $r + 1$, so we have $V_{r,1} = V_{r,2}$. For $B_{r,1} \neq B_{r,2}$, the two active S-boxes in round $r + 1$ will be in different groups.

   Next, we will prove the case for two active S-boxes in different groups in round r. We have $G_{r,1} \neq G_{r,2}$.

1. $V_{r,1} \in \{7, 11, 13, 14, 15\}$ **or** $V_{r,2} \in \{7, 11, 13, 14, 15\}$**:** There will be at least three active S-boxes in round $r + 1$.
2. $V_{r,1} \in \{3, 5, 6, 9, 10, 12\}$**:** There are at least three non-zero bits, namely $(G_{r,1}, N_{r,1}, B_{r,1}), (G_{r,1}, N_{r,1}, B_{r,2})$ and $(G_{r,2}, N_{r,2}, B_{r,3})$.
   We have

   $$\begin{aligned}&\{(G_{r,1}, N_{r,1}, B_{r,1}), (G_{r,1}, N_{r,1}, B_{r,2}), (G_{r,2}, N_{r,2}, B_{r,3})\}\\ &\xrightarrow{P} \{(B_{r,1}, G_{r,1}, N_{r,1}), (B_{r,2}, G_{r,1}, N_{r,1}), (B_{r,3}, G_{r,2}, N_{r,2})\}.\end{aligned}$$

   For $B_{r,1} \neq B_{r,2}$ and $G_{r,1} \neq G_{r,2}$, there are three active S-boxes in round $r + 1$.

3. **$V_{r,1} \in \{1,2,4,8\}$:** From the above proof, we have $V_{r,2} \in \{1,2,4,8\}$. There are two non-zero bits $\{(G_{r,1}, N_{r,1}, B_{r,1}), (G_{r,2}, N_{r,2}, B_{r,2})\}$. We have

$$\begin{aligned}
&\{(G_{r,1}, N_{r,1}, B_{r,1}), (G_{r,2}, N_{r,2}, B_{r,2})\} \xrightarrow{P} \{(B_{r,1}, G_{r,1}, N_{r,1}), (B_{r,2}, G_{r,2}, N_{r,2})\} \\
&\xrightarrow{S} \{(B_{r,1}, G_{r,1}, N_{r+1,1}), (B_{r,1}, G_{r,1}, N_{r+1,2}), (B_{r,2}, G_{r,2}, N_{r+1,3}), (B_{r,2}, G_{r,2}, N_{r+1,4})\} \\
&\xrightarrow{P} \{(N_{r+1,1}, B_{r,1}, G_{r,1}), (N_{r+1,2}, B_{r,1}, G_{r,1}), (N_{r+1,3}, B_{r,2}, G_{r,2}), (N_{r+1,4}, B_{r,2}, G_{r,2})\}.
\end{aligned}$$

In order to ensure that there are two active S-boxes in round $r+2$, $N_{r+1,1} = N_{r+1,3}$, $N_{r+1,2} = N_{r+1,4}$ and $B_{r,1} = B_{r,2}$. So we have $V_{r,1} = V_{r,2}$ and the two active S-boxes in round $r+1$ are in the same group.

□

Table 5. Differentials for 16-round PRESENT with output difference $00000500_x||00000500_x$

i	$\varDelta_0^i$	$log_2^{p_i}$	i	$\varDelta_0^i$	$log_2^{p_i}$
1	$000f0000_x\|\|0000000f_x$	-62.13	47	$000f0000_x\|\|00000f00_x$	-63.79
2	$00070000_x\|\|00000007_x$	-62.57	48	$0f000000_x\|\|0000000f_x$	-63.79
3	$0f000000_x\|\|00000f00_x$	-62.79	49	$0f000000_x\|\|00000d00_x$	-63.79
4	$000f0000_x\|\|00000007_x$	-62.84	50	$0f000000_x\|\|00000b00_x$	-63.79
5	$00070000_x\|\|0000000f_x$	-62.84	51	$0f000000_x\|\|00000300_x$	-63.79
6	$000d0000_x\|\|0000000d_x$	-62.88	52	$0f000000_x\|\|00000500_x$	-63.79
7	$00f00000_x\|\|000000f0_x$	-62.95	53	$03000000_x\|\|00000f00_x$	-63.79
8	$00090000_x\|\|00000009_x$	-63.10	54	$05000000_x\|\|00000f00_x$	-63.79
9	$000f0000_x\|\|00000003_x$	-63.13	55	$0d000000_x\|\|00000f00_x$	-63.79
10	$000f0000_x\|\|00000005_x$	-63.13	56	$0b000000_x\|\|00000f00_x$	-63.79
11	$000f0000_x\|\|0000000b_x$	-63.13	57	$00070000_x\|\|00000003_x$	-63.84
12	$000f0000_x\|\|0000000d_x$	-63.13	58	$00070000_x\|\|00000005_x$	-63.84
13	$00030000_x\|\|0000000f_x$	-63.13	59	$00030000_x\|\|00000007_x$	-63.84
14	$00050000_x\|\|0000000f_x$	-63.13	60	$00050000_x\|\|00000007_x$	-63.84
15	$000b0000_x\|\|0000000f_x$	-63.13	61	$f0000000_x\|\|00000007_x$	-63.84
16	$000d0000_x\|\|0000000f_x$	-63.13	62	$70000000_x\|\|0000000f_x$	-63.84
17	$f0000000_x\|\|0000000f_x$	-63.13	63	$000f0000_x\|\|00007000_x$	-63.84
18	$000f0000_x\|\|0000f000_x$	-63.13	64	$00070000_x\|\|0000f000_x$	-63.84
19	$000d0000_x\|\|00000007_x$	-63.19	65	$0d000000_x\|\|00000700_x$	-63.85
20	$00070000_x\|\|0000000d_x$	-63.19	66	$07000000_x\|\|00000d00_x$	-63.85
21	$0f000000_x\|\|000000f0_x$	-63.21	67	$00000f00_x\|\|00000f00_x$	-63.87
22	$00f00000_x\|\|00000f00_x$	-63.21	68	$00000000_x\|\|0f000f00_x$	-63.87
23	$00000000_x\|\|000f000f_x$	-63.21	69	$d0000000_x\|\|0000000d_x$	-63.88
24	$0000000f_x\|\|0000000f_x$	-63.21	70	$000d0000_x\|\|0000d000_x$	-63.88
25	$07000000_x\|\|00000700_x$	-63.23	71	$00000000_x\|\|000f0007_x$	-63.91
26	$00700000_x\|\|00000070_x$	-63.39	72	$00000000_x\|\|0007000f_x$	-63.91
27	$000b0000_x\|\|0000000b_x$	-63.44	73	$0000000f_x\|\|00000007_x$	-63.91
28	$000f0000_x\|\|00000009_x$	-63.50	74	$00000007_x\|\|0000000f_x$	-63.91
29	$00090000_x\|\|0000000f_x$	-63.50	75	$00900000_x\|\|00000090_x$	-63.92
30	$0f000000_x\|\|00000700_x$	-63.50	76	$0f000000_x\|\|00000070_x$	-63.92
31	$07000000_x\|\|00000f00_x$	-63.50	77	$07000000_x\|\|000000f0_x$	-63.92
32	$000b0000_x\|\|00000007_x$	-63.52	78	$00f00000_x\|\|00000700_x$	-63.92
33	$00070000_x\|\|0000000b_x$	-63.52	79	$00700000_x\|\|00000f00_x$	-63.92
34	$0d000000_x\|\|00000d00_x$	-63.54	80	$00f00000_x\|\|00000030_x$	-63.95
35	$70000000_x\|\|00000007_x$	-63.57	81	$00f00000_x\|\|00000050_x$	-63.95
36	$00070000_x\|\|00007000_x$	-63.57	82	$00f00000_x\|\|000000b0_x$	-63.95
37	$000d0000_x\|\|00000009_x$	-63.58	83	$00f00000_x\|\|000000d0_x$	-63.95
38	$00090000_x\|\|0000000d_x$	-63.58	84	$00300000_x\|\|000000f0_x$	-63.95
39	$00000000_x\|\|00070007_x$	-63.64	85	$00500000_x\|\|000000f0_x$	-63.95
40	$00000007_x\|\|00000007_x$	-63.64	86	$00b00000_x\|\|000000f0_x$	-63.95
41	$07000000_x\|\|00000070_x$	-63.65	87	$00d00000_x\|\|000000f0_x$	-63.95
42	$00700000_x\|\|00000700_x$	-63.65	88	$0d000000_x\|\|000000d0_x$	-63.95
43	$00700000_x\|\|000000f0_x$	-63.66	89	$00d00000_x\|\|00000d00_x$	-63.95
44	$00f00000_x\|\|00000070_x$	-63.66	90	$00000000_x\|\|000d000d_x$	-63.95
45	$00d00000_x\|\|000000d0_x$	-63.70	91	$0000000d_x\|\|0000000d_x$	-63.95
46	$09000000_x\|\|00000900_x$	-63.76			

Table 6. Differentials for 16-round PRESENT with output difference $00000900_x\Vert 00000900_x$

i	Δ_0^i	$log_2^{p_i}$	i	Δ_0^i	$log_2^{p_i}$
1	$000f0000_x\Vert 0000000f_x$	-62.98	10	$000f0000_x\Vert 00000005_x$	-63.98
2	$00070000_x\Vert 00000007_x$	-63.42	11	$000f0000_x\Vert 0000000b_x$	-63.98
3	$0f000000_x\Vert 00000f00_x$	-63.68	12	$000f0000_x\Vert 0000000d_x$	-63.98
4	$000f0000_x\Vert 00000007_x$	-63.69	13	$00030000_x\Vert 0000000f_x$	-63.98
5	$00070000_x\Vert 0000000f_x$	-63.69	14	$00050000_x\Vert 0000000f_x$	-63.98
6	$000d0000_x\Vert 0000000d_x$	-63.72	15	$000b0000_x\Vert 0000000f_x$	-63.98
7	$00f00000_x\Vert 000000f0_x$	-63.92	16	$000d0000_x\Vert 0000000f_x$	-63.98
8	$00090000_x\Vert 00000009_x$	-63.94	17	$f0000000_x\Vert 0000000f_x$	-63.98
9	$000f0000_x\Vert 00000003_x$	-63.98	18	$000f0000_x\Vert 0000f000_x$	-63.98

Table 7. Other differentials for 16-round PRESENT

i	Δ_0^i	Δ_{16}	$log_2^{p_i}$
1	$00000000_x\Vert 00001001_x$	$00000404_x\Vert 00000000_x$	-62.96
2	$00001001_x\Vert 00000000_x$	$00000404_x\Vert 00000000_x$	-63.62
3	$00000000_x\Vert 00004004_x$	$00000404_x\Vert 00000000_x$	-63.66
4	$00000000_x\Vert 10010000_x$	$00000404_x\Vert 00000000_x$	-63.78
5	$00000000_x\Vert 0000c004_x$	$00000404_x\Vert 00000000_x$	-63.87
6	$00000000_x\Vert 0000400c_x$	$00000404_x\Vert 00000000_x$	-63.87
7	$00000000_x\Vert 0000c00c_x$	$00000404_x\Vert 00000000_x$	-63.87
8	$00000000_x\Vert 00002002_x$	$00000404_x\Vert 00000000_x$	-63.88
9	$00000000_x\Vert 00001008_x$	$00000404_x\Vert 00000000_x$	-63.96
10	$00000000_x\Vert 0000100e_x$	$00000404_x\Vert 00000000_x$	-63.96
11	$00000000_x\Vert 00008001_x$	$00000404_x\Vert 00000000_x$	-63.96
12	$00000000_x\Vert 0000e001_x$	$00000404_x\Vert 00000000_x$	-63.96
1	$000f0000_x\Vert 0000000f_x$	$05000000_x\Vert 00000500_x$	-63.00
2	$00070000_x\Vert 00000007_x$	$05000000_x\Vert 00000500_x$	-63.43
3	$0f000000_x\Vert 00000f00_x$	$05000000_x\Vert 00000500_x$	-63.66
4	$000f0000_x\Vert 00000007_x$	$05000000_x\Vert 00000500_x$	-63.69
5	$00070000_x\Vert 0000000f_x$	$05000000_x\Vert 00000500_x$	-63.69
6	$00f00000_x\Vert 000000f0_x$	$05000000_x\Vert 00000500_x$	-63.82
7	$000d0000_x\Vert 0000000d_x$	$05000000_x\Vert 00000500_x$	-63.80
1	$000f0000_x\Vert 0000000f_x$	$00000000_x\Vert 05000500_x$	-63.33
2	$00070000_x\Vert 00000007_x$	$00000000_x\Vert 05000500_x$	-63.75
3	$0f000000_x\Vert 00000f00_x$	$00000000_x\Vert 05000500_x$	-63.99
1	$000f0000_x\Vert 0000000f_x$	$00000300_x\Vert 00000300_x$	-63.29
2	$00070000_x\Vert 00000007_x$	$00000000_x\Vert 05000500_x$	-63.73
1	$000f0000_x\Vert 0000000f_x$	$00000005_x\Vert 00000005_x$	-63.51
2	$00070000_x\Vert 00000007_x$	$00000005_x\Vert 00000005_x$	-63.95
1	$00000000_x\Vert 00001001_x$	$00004004_x\Vert 00000000_x$	-63.81
1	$000f0000_x\Vert 0000000f_x$	$00005000_x\Vert 00005000_x$	-63.67
1	$000f0000_x\Vert 0000000f_x$	$00000050_x\Vert 00000050_x$	-63.67
1	$000f0000_x\Vert 0000000f_x$	$09000000_x\Vert 00000900_x$	-63.84

Table 8. Number of differential paths with different probability for differentials in Table 5 (first part)

i	2^{-71}	2^{-73}	2^{-75}	2^{-77}	2^{-79}	2^{-81}	2^{-83}	2^{-85}	2^{-87}	2^{-89}	2^{-91}
9,10,11,12,13,14,15,16,17,18	12	160	986	3744	9654	17440	21988	18536	9280	1920	0
19,20	12	157	952	3567	9092	16264	20348	17068	8520	1760	0
32,33	9	123	769	2913	7350	12692	14780	10980	4600	800	0
35,36	9	117	707	2669	7056	13858	20936	24568	21248	11520	2560
37,38	6	89	628	2795	8562	18504	27976	28004	16200	3680	0
47,48	8	104	628	2348	5976	10676	13340	11160	5568	1152	0
49,50,51,52,53,54,55,56	4	64	486	2336	7838	19064	33976	43600	38368	20736	4608
57,58,59,60,61,62,63,64	9	114	655	2258	5092	7600	7180	3800	800	0	0
65,66	4	63	472	2243	7448	17942	31704	40376	35344	19040	4224
69,70	3	55	457	2295	7744	18318	30608	35268	26256	11040	1920
80,81,82,83,84,85,86,87	4	60	438	2066	6886	16766	30064	38908	34584	18880	4224

Table 9. Number of differential paths with different probability for differentials in Table 5 (second part)

i	2^{-70}	2^{-72}	2^{-74}	2^{-76}	2^{-78}	2^{-80}	2^{-82}	2^{-84}	2^{-86}	2^{-88}	2^{-90}	2^{-92}
1	12	160	986	3744	9654	17440	21988	18536	9280	1920	0	0
2	9	117	707	2669	7056	13858	20936	24568	21248	11520	2560	0
3	4	64	486	2336	7838	19064	33976	43600	38368	20736	4608	0
4,5	9	114	655	2258	5092	7600	7180	3800	800	0	0	0
6	3	55	457	2295	7744	18318	30608	35256	26256	11040	1920	0
7	4	60	438	2066	6886	16766	30064	38908	34584	18880	4224	0
8	3	49	383	1897	6526	16098	28564	35504	28928	13440	2560	0
21,22	4	56	382	1708	5490	13088	23300	30260	27208	15168	3456	0
23,24	0	48	472	2112	5724	10404	13104	11336	6400	1920	0	0
25	3	47	351	1673	5650	14212	27472	41472	48928	43520	25600	6144
26	3	44	316	1480	4971	12516	24286	36824	43656	39168	23296	5632
27	0	21	274	1641	6002	14746	25040	29168	22336	10080	1920	0
28,29,30,31	3	46	331	1486	4562	9840	14808	14736	8480	1920	0	0
34	1	21	205	1243	5222	15940	35960	59616	70464	55488	24960	4608
39,40	0	36	342	1496	4090	8128	12572	14936	12928	7680	2560	0
41,42	3	41	275	1223	3976	9836	18950	28680	34008	30720	18688	4608
43,44	3	43	297	1309	4000	8664	13168	13268	7720	1760	0	0
45	1	20	188	1112	4609	14004	31658	52832	63048	50160	22752	4224
46	1	19	175	1037	4364	13596	31832	55600	70336	60416	30208	6144
67,68	0	16	200	1184	4420	11276	20280	26080	23392	13824	4608	0
71,72,73,74	0	36	330	1330	3072	4480	4280	2600	800	0	0	0
75	1	18	160	928	3857	11954	28014	49196	62800	54528	27520	5632
76,77,78,79	3	40	257	1070	3152	6706	10188	10412	6200	1440	0	0
88,89	1	19	169	949	3768	11100	24650	40920	49128	39696	18336	3456
90,91	0	12	178	1160	4430	10944	18260	20952	16416	8160	1920	0

(s_0, s_1): the nibble positions for the two active S-boxes.
(v_0, v_1): the difference values for the two active S-boxes.
$cnt[i][j]$: the entity with input difference i and output difference j in differential distribution table of S-box.
xor: a list of structures and each structure is a triple.
$xor[i][0]$: the information of the first active S-box in round $i + 1$.
$xor[i][1]$: the information of the second active S-box in round $i + 1$.
$xor[i][j].sbox$: the nibble position of the active S-box.
$xor[i][j].in$: the input difference of the active S-box.
$xor[i][j].out$: the output difference of the active S-box.

```
Procedure($s_0$,$s_1$,$v_0$,$v_1$):
struct {
        sbox
        in
        out
}$xor[ROUND][2] = 0$

if $\frac{s_0}{4} = \frac{s_1}{4}$:                    #(same group)
for $xor[0][0].out \in (3, 5, 6, 9, 10, 12)$:
    $xor[0][1].out = xor[0][0].out$
    if $cnt[v_0][xor[0][0].out]! = 0$ and $cnt[v_1][xor[0][1].out]! = 0$:
        $(xor[1][0].sbox, xor[1][1].sbox, xor[1][0].in, xor[1][1].in) =$
        $P(s_0, s_1, xor[0][0].out, xor[0][1].out)$
        for $xor[1][0].out \in (1, 2, 4, 8)$:
            if $cnt[xor[1][0].in][xor[1][0].out]! = 0$:
                $xor[1][1].out = xor[1][0].out$
                $(xor[2][0].sbox, xor[2][1].sbox, xor[2][0].in, xor[2][1].in) =$
                $P(xor[1][0].sbox, xor[1][1].sbox, xor[1][0].out, xor[1][1].out)$
                ...
                Compute probability and display differential path.
else                              #(different groups)
for $xor[0][0].out \in (1, 2, 4, 8)$:
    $xor[0][1].out = xor[0][0].out$
    if $cnt[v_0][xor[0][0].out]! = 0$ and $cnt[v_1][xor[0][1].out]! = 0$:
        $(xor[1][0].sbox, xor[1][1].sbox, xor[1][0].in, xor[1][1].in) =$
        $P(s_0, s_1, xor[0][0].out, xor[0][1].out)$
        for $xor[1][0].out \in (3, 5, 6, 9, 10, 12)$:
            if $cnt[xor[1][0].in][xor[1][0].out]! = 0$:
                $xor[1][1].out = xor[1][0].out$
                $(xor[2][0].sbox, xor[2][1].sbox, xor[2][0].in, xor[2][1].in) =$
                $P(xor[1][0].sbox, xor[1][1].sbox, xor[1][0].out, xor[1][1].out)$
                ...
                Compute probability and display differential path.
```

Fig. 3. Search algorithm for the differential paths of PRESENT with two active S-boxes per round.

A Methodology for Differential-Linear Cryptanalysis and Its Applications[⋆]

(Extended Abstract)

Jiqiang Lu

Institute for Infocomm Research,
Agency for Science, Technology and Research
1 Fusionopolis Way, #19-01 Connexis, Singapore 138632
lvjiqiang@hotmail.com

Abstract. In 1994 Langford and Hellman introduced a combination of differential and linear cryptanalysis under two default independence assumptions, known as differential-linear cryptanalysis, which is based on the use of a differential-linear distinguisher constructed by concatenating a linear approximation with a (truncated) differential with probability 1. In 2002, by using an additional assumption, Biham, Dunkelman and Keller gave an enhanced version that can be applicable to the case when a differential with a probability of smaller than 1 is used to construct a differential-linear distinguisher. In this paper, we present a new methodology for differential-linear cryptanalysis under the original two assumptions implicitly used by Langford and Hellman, without using the additional assumption of Biham et al. The new methodology is more reasonable and more general than Biham et al.'s methodology, and apart from this advantage it can lead to some better differential-linear cryptanalytic results than Biham et al.'s and Langford and Hellman's methodologies. As examples, we apply it to attack 10 rounds of the CTC2 block cipher with a 255-bit block size and key, 13 rounds of the DES block cipher, and 12 rounds of the Serpent block cipher. The new methodology can be used to cryptanalyse other block ciphers, and block cipher designers should pay attention to this new methodology when designing a block cipher.

Key words: Block cipher, CTC2, DES, Serpent, Differential cryptanalysis, Linear cryptanalysis, Differential-linear cryptanalysis.

1 Introduction

Differential cryptanalysis was introduced in 1990 by Biham and Shamir [10]. Linear cryptanalysis was introduced in 1992 by Matsui and Yamagishi [36]. A differential cryptanalysis attack is based on the use of one or more so-called differentials, and a linear cryptanalysis attack is based on the use of one or more so-called linear approximations. Both the cryptanalytic methods were used to attack the full Data Encryption Standard (DES) [37] algorithm faster than exhaustive key search [12, 34].

In 1994 Langford and Hellman [31] introduced a combination of differential and linear cryptanalysis under two default independence assumptions, known as differential-linear cryptanalysis, and they applied it to break 8-round DES. Such an attack is constructed on a so-called differential-linear distinguisher; a differential-linear distinguisher treats a block cipher as a cascade of two sub-ciphers, and it uses a linear approximation for a sub-cipher and, for the other sub-cipher it uses a differential (or a truncated differential [27]) with a one probability that does not affect the bit(s) concerned by the input mask of the linear approximation. In 2002, by using an additional assumption Biham, Dunkelman and Keller [6] introduced an enhanced version of differential-linear cryptanalysis, which is applicable to the case when a differential with a smaller probability is used to construct a differential-linear distinguisher; and they applied

[⋆] An earlier version of this work appeared in 2010 as part of Cryptology ePrint Archive Report 2010/025 [33], which was done when the author was with Eindhoven University of Technology (The Netherlands) under the support of the Dutch Sentinels project PINPASJC (No. TIF.6687).

Table 1. Our and previous main cryptanalytic results on CTC2, DES and Serpent

Cipher	Attack Technique	Rounds	Data	Time	Success Rate	Source
CTC2 (255-bit version)	Algebraic	6	4CP	2^{253}Enc.	not specified	[14]
	Differential-linear	$8^{\dagger}$	2^{37}CP	2^{37}Enc.	61.8%	[19]
		10	2^{142}CP	2^{207}Enc.	99.9%	Sect. 5.4
DES	Differential	full	$2^{47.2}$CP	2^{37}Enc.	not specified	[12]
	Linear	full	2^{43}KP	2^{47}Enc.	85%	[34]
	Davis's attack [17]	full	2^{50}KP	2^{50}Enc.	51%	[3]
	Differential-linear	8	768CP	2^{40}Enc.	95%	[31]
		9	$2^{15.75}$CP	2^{38}Enc.	88.8%	[18]
		10	$2^{29.66}$CP	2^{44}Enc.	97%	Sect. 4.2
		13	$2^{52.1}$CP	$2^{54.2}$Enc.	99%	Sect. 4.2
Serpent	Differential	8	2^{84}CP	$2^{206.7}$Enc.	not specified	[4]
	Amplified boomerang [24]	9	2^{110}CP	2^{252}Enc.	not specified	[24]
	Boomerang [40]	10	$2^{126.3}$ACPC	2^{165}Enc.	not specified	[7]
	Rectangle	10	$2^{126.3}$CP	2^{165}Enc.	not specified	[7]
	Linear	11	2^{118}KP	2^{178}Enc.	not specified	[13]
	Differential-linear	12	$2^{123.5}$CP	$2^{249.4}$Enc.	84%	[20]
			$2^{124.5}$CP	$2^{244.9}$Enc.	98.8%	Sect. 6.3

†: There is a flaw; see Section 5.2 for detail.

the enhanced version to break 9-round DES. Differential-linear cryptanalysis has been used to yield the best currently published cryptanalytic results for a number of state-of-the-art block ciphers [6, 8, 19, 20].

In this paper, we present a new methodology for differential-linear cryptanalysis under the two default assumptions implicitly used by Langford and Hellman, without using the additional assumption due to Biham et al. The new methodology is more reasonable and more general than Biham et al.'s methodology, and it can lead to some better differential-linear cryptanalytic results than Biham et al.'s and Langford and Hellman's methodologies. As examples, we apply the new methodology to mount differential-linear attacks on 10 rounds of the CTC2 [14] block cipher with a 255-bit block size and key, 13 rounds of DES, and 12 rounds of the Serpent [1] block cipher. In terms of the numbers of attacked rounds: The 10-round CTC2 attack is the first published cryptanalytic attack on the version of CTC2; the 13-round DES attack is much better than any previously published differential-linear cryptanalytic results for DES, though it is inferior to the best previously published cryptanalytic results for DES; and the 12-round Serpent attack matches the best previously published cryptanalytic result for Serpent, that was obtained under Biham et al.'s methodology. Table 1 summarises both our and previous main cryptanalytic results on CTC2, DES and Serpent, where CP, KP and ACPC refer respectively to the required numbers of chosen plaintexts, known plaintexts and adaptively chosen plaintexts and ciphertexts, and Enc. refers to the required number of encryption operations of the relevant version of CTC2, DES and Serpent.

The remainder of the paper is organised as follows. In the next section we give the notation used throughout the paper and briefly describe differential and linear cryptanalysis. In Section 3 we give the new methodology for differential-linear cryptanalysis. In Sections 4–6 we present our cryptanalytic results on DES, CTC2 and Serpent, respectively. We discuss a few possible extensions to our methodology in Section 7. Section 8 concludes this paper.

2 Preliminaries

In this section we describe the notation, differential and linear cryptanalysis.

2.1 Notation

In the following descriptions, we assume that a number without a prefix is in decimal notation, and a number with prefix $0x$ is in hexadecimal notation, unless otherwise stated. The bits of a value are numbered from right to left, the leftmost bit is the most significant bit, and the rightmost bit is the least significant bit, except in the case of DES, where we use the same numbering notation as in FIPS-46 [37]. We use the following notation.

$\oplus$	bitwise logical exclusive OR (XOR) of two bit strings of the same length
$\odot$	dot product of two bit strings of the same length
$\vert\vert$	string concatenation
$\ll$	left shift of a bit string
$\lll$	left rotation of a bit string
$\circ$	functional composition. When composing functions X and Y, X $\circ$ Y denotes the function obtained by first applying X and then applying Y
e_j	a 255-bit value with zeros everywhere except for bit position j, $(0 \leq j \leq 254)$
$e_{i_0,\cdots,i_j}$	the 255-bit value equal to $e_{i_0} \oplus \cdots \oplus e_{i_j}$, $(0 \leq i_0, \cdots, i_j \leq 254)$
$\mathbb{E}$	an n-bit block cipher when used with a specific user key

2.2 Differential Cryptanalysis

Differential cryptanalysis [10] takes advantage of how a specific difference in a pair of inputs of a cipher can affect a difference in the pair of outputs of the cipher, where the pair of outputs are obtained by encrypting the pair of inputs using the same key. The notion of difference can be defined in several ways; the most widely discussed is with respect to the XOR operation. The difference between the inputs is called the input difference, and the difference between the outputs of a function is called the output difference. The combination of the input difference and the output difference is called a differential. The probability of a differential is defined as follows.

Definition 1 (from [32]). *If α and β are n-bit blocks, then the probability of the differential (α, β) for $\mathbb{E}$, written $\Delta\alpha \to \Delta\beta$, is defined to be*

$$\Pr_{\mathbb{E}}(\Delta\alpha \to \Delta\beta) = \Pr_{P \in \{0,1\}^n}(\mathbb{E}(P) \oplus \mathbb{E}(P \oplus \alpha) = \beta).$$

The following result follows trivially from Definition 1:

Proposition 1 (from [32]). *If α and β are n-bit blocks, then*

$$\Pr_{\mathbb{E}}(\Delta\alpha \to \Delta\beta) = \frac{|\{x | \mathbb{E}(x) \oplus \mathbb{E}(x \oplus \alpha) = \beta, x \in \{0,1\}^n\}|}{2^n}.$$

For a random function, the expected probability of a differential for any pair (α, β) is 2^{-n}. Therefore, if $\Pr_{\mathbb{E}}(\Delta\alpha \to \Delta\beta)$ is larger than 2^{-n}, we can use the differential to distinguish $\mathbb{E}$ from a random function, given a sufficient number of chosen plaintext pairs.

Sometimes, we simply write $\Delta\alpha \overset{\mathbb{E}}{\to} \Delta\beta$ to denote the differential $\Delta\alpha \to \Delta\beta$ for $\mathbb{E}$ in this paper.

2.3 Linear Cryptanalysis

Linear cryptanalysis [34, 36] exploits correlations between a particular linear function of the input blocks and a second linear function of the output blocks. The combination of the two linear functions is called a linear approximation. The most widely used linear function involves computing the bitwise dot product operation of the block with a specific binary vector (the specific value combined with the input blocks may be different from the value applied to the output blocks). The value combined with the input blocks is called the input mask, and the value applied to the output blocks is called the output mask. The probability of a linear approximation is defined as follows.

Definition 2 (from [32]). *If α and β are n-bit blocks, then the probability of the linear approximation (α, β) for $\mathbb{E}$, written $\Gamma\alpha \to \Gamma\beta$, is defined to be*

$$\Pr_{\mathbb{E}}(\Gamma\alpha \to \Gamma\beta) = \Pr_{P \in \{0,1\}^n}(P \odot \alpha = \mathbb{E}(P) \odot \beta).$$

We refer to below the dot product $P \odot \alpha$ as the input parity, and the dot product $\mathbb{E}(P) \odot \beta$ as the output parity. The following result follows trivially from Definition 2:

Proposition 2 (from [32]). *If α and β are n-bit blocks, then*

$$\Pr_{\mathbb{E}}(\Gamma\alpha \to \Gamma\beta) = \frac{|\{x | x \odot \alpha = \mathbb{E}(x) \odot \beta, x \in \{0,1\}^n\}|}{2^n}.$$

For a random function, the expected probability of a linear approximation for any pair (α, β) is $\frac{1}{2}$. The bias of a linear approximation $\Gamma\alpha \to \Gamma\beta$, denoted by ϵ, is defined to be $\epsilon = |\Pr_{\mathbb{E}}(\Gamma\alpha \to \Gamma\beta) - \frac{1}{2}|$. Thus, if the bias ϵ is sufficiently large, we can use the linear approximation to distinguish $\mathbb{E}$ from a random function, given a sufficient number of matching plaintext-ciphertext pairs.

2.4 General Assumptions Used in Practice

Propositions 1 and 2 give the accurate probability values of a differential and a linear approximation from a theoretical point of view. However, it is usually hard to apply them to a block cipher with a large block size, for example, $n = 64$ or 128 which is currently being widely used in reality, and even harder when the differential or linear approximation operates on many rounds of the cipher. In practice, for a Markov block cipher [29], a multi-round differential (or linear approximation) is usually obtained by concatenating a few one-round differential characteristics (respectively, linear approximations), and the probability of the multi-round differential (or linear approximation) is regarded as the product (respectively, the piling-up function [34]) of the probabilities of the one-round differential characteristics (respectively, linear approximations) under the following Assumption 1.

Assumption 1 *The involved round functions behave independently.*

We note that one may argue the correctness of Assumption 1 and may use a different assumption, for example, many people would like to use the assumption that the round keys are independent and uniformly distributed; however, it is not accurate, either, for generally the round keys are actually dependent, being generated from a global user key under the key schedule algorithm of the cipher. Anyway, all such assumptions require us to treat the involved rounds as independent. As mentioned in [22], this is "most often not exactly the case, but as often it is a good approximation".

Differential and linear cryptanalyses generally treat a basic unit of input (i.e. a chosen-plaintext pair for differential cryptanalysis; a known-plaintext for linear cryptanalysis) as a random variable, and assume that given a set of inputs of the basic unit, the inputs that satisfy the required property can be approximated by an independent distribution, as followed in [11,34].

3 Differential-Linear Cryptanalysis: Previous Work and Our Methodology

In this section we first review previous work on differential-linear cryptanalysis, and then give our new methodology, followed by a few implications. First observe that for simplicity we assume that the probability for a linear approximation with bias ϵ is $\frac{1}{2}+\epsilon$ in all the following descriptions; but the same results can be obtained when the probability is $\frac{1}{2}-\epsilon$.

3.1 Previous Work

Langford and Hellman's Methodology. In 1994 Langford and Hellman [31] introduced differential-linear cryptanalysis as a combination of differential and linear cryptanalysis, which is based on the use of a differential-linear distinguisher. To define a differential-linear distinguisher, they treated $\mathbb{E}$ as a cascade of two sub-ciphers $\mathbb{E}_0$ and $\mathbb{E}_1$, where $\mathbb{E} = \mathbb{E}_0 \circ \mathbb{E}_1$. A differential-linear distinguisher is then defined to be the combination of a (truncated) differential and a linear approximation $(\Delta\alpha \to \Delta\beta, \Gamma\gamma \to \Gamma\delta)$, where $\Gamma\gamma \to \Gamma\delta$ is a linear approximation with bias ϵ for $\mathbb{E}_1$, and $\Delta\alpha \to \Delta\beta$ is a (truncated) differential with probability 1 for $\mathbb{E}_0$ which has a zero output difference in the bit positions concerned by the input mask of the linear approximation (thus $\beta \odot \gamma = 0$ holds). Let P be a plaintext chosen uniformly at random from $\{0,1\}^n$. Thus, we have $\mathbb{E}_0(P) \odot \gamma = \mathbb{E}_0(P \oplus \alpha) \odot \gamma$ with probability 1. The differential-linear distinguisher is concerned with the event $\delta \odot \mathbb{E}(P) = \delta \odot \mathbb{E}(P \oplus \alpha)$; and under Assumption 1 and the following Assumption 2 it has a probability of $\Pr(\delta \odot \mathbb{E}(P) = \delta \odot \mathbb{E}(P \oplus \alpha)) = (\frac{1}{2}+\epsilon) \times (\frac{1}{2}+\epsilon) + (\frac{1}{2}-\epsilon) \times (\frac{1}{2}-\epsilon) = \frac{1}{2}+2\epsilon^2$.

Assumption 2 *The two inputs $\mathbb{E}_0(P)$ and $\mathbb{E}_0(P \oplus \alpha)$ of the linear approximation for $\mathbb{E}_1$ behave as independent inputs with respect to the linear approximation.*

Note that $\mathbb{E}(P) = \mathbb{E}_1(\mathbb{E}_0(P))$ and $\mathbb{E}(P \oplus \alpha) = \mathbb{E}_1(\mathbb{E}_0(P \oplus \alpha))$ in the above descriptions. Assumption 2 is somewhat like assuming an independent distribution for plaintext pairs generated from a particular structure of data with certain property in differential cryptanalysis.

By contrast, for a random function, the expected probability of a differential-linear distinguisher is $\frac{1}{2}$. Therefore, if the bias $|\Pr(\delta \odot \mathbb{E}(P) = \delta \odot \mathbb{E}(P \oplus \alpha)) - \frac{1}{2}| = 2\epsilon^2$ is sufficiently large, we can distinguish $\mathbb{E}$ from a random function.

Biham et al.'s Methodology. A differential-linear distinguisher plays a fundamental role in a differential-linear cryptanalysis attack. In 2002 Biham, Dunkelman and Keller [6] presented an enhanced version to make a differential-linear distinguisher cover more rounds of a block cipher, so that an attacker can potentially break more rounds of the cipher. Biham et al.'s enhanced version includes the case when the (truncated) differential $\Delta\alpha \to \Delta\beta$ has a smaller probability than 1, p say, with β meeting the condition $\beta \odot \gamma = 0$.[1] A slightly revised version was given in [18]. They applied Langford and Hellman's analysis described above when $\mathbb{E}_0(P) \oplus \mathbb{E}_0(P \oplus \alpha) = \beta$, and used the following Assumption 3 for the cases where $\mathbb{E}_0(P) \oplus \mathbb{E}_0(P \oplus \alpha) \neq \beta$:[2]

Assumption 3 *The output parities $\delta \odot \mathbb{E}(P)$ and $\delta \odot \mathbb{E}(P \oplus \alpha)$ have a uniform and independent distribution in $\{0,1\}$ for the cases where $\mathbb{E}_0(P) \oplus \mathbb{E}_0(P \oplus \alpha) \neq \beta$.*

[1] A more general condition is $\beta \odot \gamma = c$, where $c \in \{0,1\}$ is a constant. Without loss of generality, we consider the case with $c = 0$ throughout this paper.

[2] We note that Biham et al. used a different assumption when reviewing the enhanced version in a few other papers, [9] say, where they assumed that $\mathbb{E}_0(P) \odot \gamma = \mathbb{E}_0(P \oplus \alpha) \odot \gamma$ holds with half a chance for the cases where $\mathbb{E}_0(P) \oplus \mathbb{E}_0(P \oplus \alpha) \neq \beta$, yielding the same probability value $\frac{1}{2} + 2p\epsilon^2$ as Assumption 3. We treat this assumption as Assumption 3, though they are different.

Finally, under Assumptions 1, 2 and 3, Biham et al. got $\Pr(\delta \odot \mathbb{E}(P) = \delta \odot \mathbb{E}(P \oplus \alpha)) = p \times (\frac{1}{2} + 2\epsilon^2) + (1-p) \times \frac{1}{2} = \frac{1}{2} + 2p\epsilon^2$.

As a result, they concluded that if the bias $2p\epsilon^2$ is sufficiently large, the distinguisher can be used as the basis of a differential-linear attack to distinguish $\mathbb{E}$ from a random function. Roughly, the attack has a data complexity of about $O(p^{-2}\epsilon^{-4})$.

Note. We learnt from the comments of an anonymous reviewer that the same methodology appeared earlier in Langford's PhD thesis [30], (which seems to be not publicly accessible). For simplicity, in this paper we use the phrase "Biham et al.'s methodology" to express this methodology, but hope the reader to keep in mind that Langford proposed the same methodology a few years earlier.

3.2 Our Methodology

In summary, the differential-linear distinguishers described above are concerned with the correlation between a pair of output parities, where the pair of output parities are obtained by applying a linear function (e.g. bitwise dot product with δ) to the outputs of a pair of input blocks with difference α (under the same key). The combination of the input difference and the linear function is called a differential-linear distinguisher. More formally, we define the probability of the differential-linear distinguisher as follows.

Definition 3. *If α and δ are n-bit blocks, then the probability of the differential-linear distinguisher (α, δ) for $\mathbb{E}$, written $\Delta\alpha \to \Gamma\delta$, is defined to be*

$$\Pr_{\mathbb{E}}(\Delta\alpha \to \Gamma\delta) = \Pr_{P \in \{0,1\}^n}(\mathbb{E}(P) \odot \delta = \mathbb{E}(P \oplus \alpha) \odot \delta).$$

The following result follows trivially from Definition 3:

Proposition 3. *If α and δ are n-bit blocks, then*

$$\Pr_{\mathbb{E}}(\Delta\alpha \to \Gamma\delta) = \frac{|\{x | \mathbb{E}(x) \odot \delta = \mathbb{E}(x \oplus \alpha) \odot \delta, x \in \{0,1\}^n\}|}{2^n}.$$

For a random function, the expected probability of a differential-linear distinguisher for any combination (α, δ) is $\frac{1}{2}$. Similarly, the bias of the differential-linear distinguisher $\Delta\alpha \to \Gamma\delta$ is defined to be $|\Pr_{\mathbb{E}}(\Delta\alpha \to \Gamma\delta) - \frac{1}{2}|$. Thus, if the bias is sufficiently large, we can use the differential-linear distinguisher to distinguish $\mathbb{E}$ from a random function, given a sufficient number of chosen plaintext pairs.

In practice, it is usually infeasible to compute the accurate probability of a differential-linear distinguisher $\Delta\alpha \to \Gamma\delta$ by Proposition 3, and we have to make use of some assumptions to approximate it, like Biham et al.'s methodology described in Section 3.1. However, Biham et al.'s methodology uses the three assumptions as hypotheses and works only when Assumption 3 holds; otherwise it may give probability values that are highly inaccurate in some situations; for example, let's intuitively consider the naive situation where the differential $\Delta\alpha \to \Delta\beta$ has probability $\frac{1}{2}$ and meets $\beta \odot \gamma = 0$, and all the other possible differentials $\{\Delta\alpha \to \Delta\widehat{\beta}\}$ meet $\widehat{\beta} \odot \gamma = 1$. Such an example can be easily built for a practical block cipher, DES say. The differential $\Delta\alpha \to \Delta\beta$ contributes $\frac{1}{2}[(\frac{1}{2} + \epsilon) \times (\frac{1}{2} + \epsilon) + (\frac{1}{2} - \epsilon) \times (\frac{1}{2} - \epsilon)] = \frac{1}{4} + \epsilon^2$ to the probability of the distinguisher, and the other differentials $\{\Delta\alpha \to \Delta\widehat{\beta}\}$ contribute $\frac{1}{2}[(\frac{1}{2} + \epsilon) \times (\frac{1}{2} - \epsilon) + (\frac{1}{2} - \epsilon) \times (\frac{1}{2} + \epsilon)] = \frac{1}{4} - \epsilon^2$, which also cause a bias, but in a negative way, canceling the bias due to $\Delta\alpha \to \Delta\beta$. So the real bias of the distinguisher is 0, that is, the distinguisher has no cryptanalytic significance. But if we applied Biham et al.'s methodology in this situation, the distinguisher would have a bias of $2 \times \frac{1}{2} \times \epsilon^2 = \epsilon^2$, and thus the distinguisher would be useful (if ϵ^2 is large enough); but nevertheless it is useless in fact. Notice that this case is not truly a counterexample to Biham et al.'s methodology, for it is clear that Assumption 3 does not hold for it, but it suggests that we should be cautious about using Assumption 3 and actually, we should be careful with using any assumption, and it is preferable to use as few assumptions as possible.

Biham, Dunkelman and Keller used a heuristic way to approximate the probability of a differential-linear distinguisher. We make an analysis for the probability of a differential-linear distinguisher from a mathematical point, and obtain a new methodology under only Assumptions 1 and 2. Our result is given as Theorem 1, followed by a proof.

Theorem 1. *An n-bit block cipher $\mathbb{E}$ is represented as a cascade of two sub-ciphers $\mathbb{E}_0$ and $\mathbb{E}_1$, where $\mathbb{E} = \mathbb{E}_0 \circ \mathbb{E}_1$. If α $(\neq 0)$ is an input difference for $\mathbb{E}_0$, $\varGamma\gamma \to \varGamma\delta$ is a linear approximation with bias ϵ for $\mathbb{E}_1$, and the sum of the probabilities for the differentials $\{\varDelta\alpha \to \varDelta\beta | \Pr_{\mathbb{E}_0}(\varDelta\alpha \to \varDelta\beta) > 0, \gamma \odot \beta = 0, \beta \in \{0,1\}^n\}$ is $\widehat{p}$ $(= \sum_{\gamma\odot\beta=0} \Pr_{\mathbb{E}_0}(\varDelta\alpha \to \varDelta\beta))$, then under Assumptions 1 and 2 the probability of the differential-linear distinguisher $\varDelta\alpha \to \varGamma\delta$ is*

$$\Pr_{P\in\{0,1\}^n}(\mathbb{E}(P) \odot \delta = \mathbb{E}(P \oplus \alpha) \odot \delta) = \frac{1}{2} + 2(2\widehat{p} - 1)\epsilon^2.$$

Proof. Given the input difference α for $\mathbb{E}_0$, there are one or more possible output differences $\{\beta | \Pr_{\mathbb{E}_0}(\varDelta\ \alpha \to \varDelta\beta) > 0, \beta \in \{0,1\}^n\}$; these output differences can be classified into two sets: one is $\{\beta|\gamma\odot\beta = 0, \Pr_{\mathbb{E}_0}(\varDelta\alpha \to \varDelta\beta) > 0, \beta \in \{0,1\}^n\}$, and the other is $\{\beta|\gamma\odot\beta = 1, \Pr_{\mathbb{E}_0}(\varDelta\alpha \to \varDelta\beta) > 0, \beta \in \{0,1\}^n\}$.

Let P be a plaintext chosen uniformly at random from $\{0,1\}^n$. Then, under Assumptions 1 and 2 we have

$$\begin{aligned}
&\Pr(\mathbb{E}(P) \odot \delta = \mathbb{E}(P \oplus \alpha) \odot \delta | \mathbb{E}_0(P) \oplus \mathbb{E}_0(P \oplus \alpha) = \beta, \gamma \odot \beta = 0)\\
&= \Pr(\mathbb{E}_0(P) \odot \gamma = \mathbb{E}(P) \odot \delta, \mathbb{E}_0(P \oplus \alpha) \odot \gamma = \mathbb{E}(P \oplus \alpha) \odot \delta |\\
&\quad \mathbb{E}_0(P) \oplus \mathbb{E}_0(P \oplus \alpha) = \beta, \gamma \odot \beta = 0) +\\
&\quad \Pr(\mathbb{E}_0(P) \odot \gamma \neq \mathbb{E}(P) \odot \delta, \mathbb{E}_0(P \oplus \alpha) \odot \gamma \neq \mathbb{E}(P \oplus \alpha) \odot \delta |\\
&\quad \mathbb{E}_0(P) \oplus \mathbb{E}_0(P \oplus \alpha) = \beta, \gamma \odot \beta = 0)\\
&= (\frac{1}{2} + \epsilon) \times (\frac{1}{2} + \epsilon) + [1 - (\frac{1}{2} + \epsilon)] \times [1 - (\frac{1}{2} + \epsilon)]\\
&= \frac{1}{2} + 2\epsilon^2,
\end{aligned}$$

and

$$\begin{aligned}
&\Pr(\mathbb{E}(P) \odot \delta = \mathbb{E}(P \oplus \alpha) \odot \delta | \mathbb{E}_0(P) \oplus \mathbb{E}_0(P \oplus \alpha) = \beta, \gamma \odot \beta = 1)\\
&= \Pr(\mathbb{E}_0(P) \odot \gamma = \mathbb{E}(P) \odot \delta, \mathbb{E}_0(P \oplus \alpha) \odot \gamma \neq \mathbb{E}(P \oplus \alpha) \odot \delta |\\
&\quad \mathbb{E}_0(P) \oplus \mathbb{E}_0(P \oplus \alpha) = \beta, \gamma \odot \beta = 1) +\\
&\quad \Pr(\mathbb{E}_0(P) \odot \gamma \neq \mathbb{E}(P) \odot \delta, \mathbb{E}_0(P \oplus \alpha) \odot \gamma = \mathbb{E}(P \oplus \alpha) \odot \delta |\\
&\quad \mathbb{E}_0(P) \oplus \mathbb{E}_0(P \oplus \alpha) = \beta, \gamma \odot \beta = 1)\\
&= (\frac{1}{2} + \epsilon) \times [1 - (\frac{1}{2} + \epsilon)] + [1 - (\frac{1}{2} + \epsilon)] \times (\frac{1}{2} + \epsilon)\\
&= \frac{1}{2} - 2\epsilon^2.
\end{aligned}$$

Next, under Assumptions 1 and 2 we can compute the probability of the differential-linear distinguisher as follows.

$$\Pr(\mathbb{E}(P) \odot \delta = \mathbb{E}(P \oplus \alpha) \odot \delta)$$

$$
\begin{aligned}
&= \sum_{\beta\in\{0,1\}^n, Y\in\{0,1\}} \Pr(\mathbb{E}(P)\odot\delta=\mathbb{E}(P\oplus\alpha)\odot\delta, \mathbb{E}_0(P)\odot\gamma\oplus\mathbb{E}_0(P\oplus\alpha)\odot\gamma=Y,\\
&\quad \mathbb{E}_0(P)\oplus\mathbb{E}_0(P\oplus\alpha)=\beta)\\
&= \sum_{\beta\in\{0,1\}^n, Y\in\{0,1\}} \Pr(\mathbb{E}(P)\odot\delta=\mathbb{E}(P\oplus\alpha)\odot\delta | \mathbb{E}_0(P)\odot\gamma\oplus\mathbb{E}_0(P\oplus\alpha)\odot\gamma=Y,\\
&\quad \mathbb{E}_0(P)\oplus\mathbb{E}_0(P\oplus\alpha)=\beta)\times\\
&\quad \Pr(\mathbb{E}_0(P)\odot\gamma\oplus\mathbb{E}_0(P\oplus\alpha)\odot\gamma=Y, \mathbb{E}_0(P)\oplus\mathbb{E}_0(P\oplus\alpha)=\beta)\\
&= \sum_{\beta\in\{0,1\}^n} \Pr(\mathbb{E}(P)\odot\delta=\mathbb{E}(P\oplus\alpha)\odot\delta | \mathbb{E}_0(P)\odot\gamma\oplus\mathbb{E}_0(P\oplus\alpha)\odot\gamma=0,\\
&\quad \mathbb{E}_0(P)\oplus\mathbb{E}_0(P\oplus\alpha)=\beta)\times\Pr(\mathbb{E}_0(P)\odot\gamma\oplus\mathbb{E}_0(P\oplus\alpha)\odot\gamma=0,\\
&\quad \mathbb{E}_0(P)\oplus\mathbb{E}_0(P\oplus\alpha)=\beta)+\\
&\quad \sum_{\beta\in\{0,1\}^n} \Pr(\mathbb{E}(P)\odot\delta=\mathbb{E}(P\oplus\alpha)\odot\delta | \mathbb{E}_0(P)\odot\gamma\oplus\mathbb{E}_0(P\oplus\alpha)\odot\gamma=1,\\
&\quad \mathbb{E}_0(P)\oplus\mathbb{E}_0(P\oplus\alpha)=\beta)\times\Pr(\mathbb{E}_0(P)\odot\gamma\oplus\mathbb{E}_0(P\oplus\alpha)\odot\gamma=1,\\
&\quad \mathbb{E}_0(P)\oplus\mathbb{E}_0(P\oplus\alpha)=\beta) \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (1)\\
&= (\frac{1}{2}+2\epsilon^2)\times \sum_{\beta\in\{0,1\}^n, \gamma\odot\beta=0} \Pr(\mathbb{E}_0(P)\oplus\mathbb{E}_0(P\oplus\alpha)=\beta)+\\
&\quad (\frac{1}{2}-2\epsilon^2)\times \sum_{\beta\in\{0,1\}^n, \gamma\odot\beta=1} \Pr(\mathbb{E}_0(P)\oplus\mathbb{E}_0(P\oplus\alpha)=\beta)\\
&= \frac{1}{2}+2(2\widehat{p}-1)\epsilon^2. \ \square
\end{aligned}
$$

Consequently, the bias of the differential-linear distinguisher $\mathit{\Delta}\alpha \rightarrow \mathit{\Gamma}\delta$ is

$$|\Pr_{P\in\{0,1\}^n}(\mathbb{E}(P)\odot\delta=\mathbb{E}(P\oplus\alpha)\odot\delta)-\frac{1}{2}| = 2|2\widehat{p}-1|\epsilon^2.$$

3.3 Implications

Biham et al.'s methodology requires Assumptions 1, 2 and 3, while our methodology requires only Assumptions 1 and 2. Thus, our methodology is more reasonable than Biham et al.'s methodology.

Biham et al.'s methodology holds only when Assumption 3 holds, and under the situation we have $\widehat{p} = p + (1-p)\frac{1}{2} = \frac{1}{2} + \frac{p}{2}$, meaning that the probability value obtained using Biham et al.'s methodology equals that obtained using our methodology. Thus, when Biham et al.'s methodology holds, our methodology always holds. However, our methodology holds under some situations where Biham et al.'s methodology does not hold, for example, it works for the naive situation discussed in Section 3.2 where $\widehat{p} = p = \frac{1}{2}$. Therefore, our methodology is more general than Biham et al.'s methodology. (When Langford and Hellman's methodology holds, our methodology always holds as well.)

Our methodology still requires Assumptions 1 and 2. Assumption 1 is extensively used in and is commonly regarded as necessary for differential and linear cryptanalysis in practice. Assumption 2 seems irremovable to get such a simple and practical probability formula; otherwise, the formula could not be so simple, but a more accurate version can be easily obtained from our above reasonings, for instance, from Eq. (1), though it is complicated and appears to be hardly applicable in practice. The assumptions mean that, in some cases, the probability of a differential-linear distinguisher may be overestimated or underestimated, and so is the success probability of the attack; however, computer experiments [8,20,28,31,34,35] have shown that the assumptions work well in practice for some block ciphers. Anyway, it seems reasonable to take the worst case assumption from the point of the user of a cipher. We suggest that if possible an attacker should check the validity of these assumptions when applying them to a specific cipher.

Our result shows that using only one (truncated) differential satisfying $\beta \odot \gamma = 0$ is not sufficient in most situations, and it is likely to be not sufficient in the general situation; we should use all the differentials satisfying $\beta \odot \gamma = 0$ instead. This makes the distinguisher harder and even impossible to construct in practice, due to a large number of possible output differences. Anyway, we should use at least those differentials with a significant contribution to reduce the deviation if we are able to do so. Biham et al.'s methodology suggests that if the bias of the linear approximation keeps constant, the larger p is, the bigger is the bias of the distinguisher. Now, we know that may be not true in the general situation: A differential with a bigger probability will not necessarily result in a distinguisher with a bigger bias.

When constructing a differential-linear distinguisher, in Biham et al.'s methodology the attacker first chooses a (truncated) differential that meets the condition (as followed in [6,8,19,20], in practice the output difference of the differential has zeros in the bit positions concerned by the input mask of the linear approximation), then calculates the probability of the differential, and finally takes this probability as the value of p. Our new methodology suggests a different format, that is, computing $\widehat{p}$. Once the linear approximation and the input difference of the differentials are chosen, that how many rounds can be constructed for a distinguisher depends to some extent on the computational power available for the attacker.

Our new methodology can lead to some better differential-linear cryptanalytic results than Biham et al.'s and Langford and Hellman's methodologies, as to be demonstrated by its applications to the block ciphers DES, CTC2 and Serpent in the following sections. Before further proceeding, observe that DES is a Markov cipher under the XOR difference notion [29], and similarly we can learn that both CTC2 and Serpent are Markov ciphers under the XOR difference notion.

At last, to be conservative, we would like to suggest that one should pay attention to all these methodologies, for a real situation is usually hard to predict, and it may make the Assumption 3 for Biham et al.'s methodology hold.

4 Application to the DES Block Cipher

The DES block cipher is well known to both academia and industry, which has a 64-bit block size, a 56-bit user key, and a total of 16 rounds. We refer the reader to [37] for the specifications of DES.

In 1994, under the two default Assumptions 1 and 2 Langford and Hellman [31] used their methodology to obtain a 6-round differential-linear distinguisher of DES, and finally applied it to break 8-round DES; the attack recovers 16 key bits with a time complexity of $2^{14.6}$ 8-round DES encryptions, so it would take 2^{40} encryptions to recover the remaining 40 key bits with an exhaustive search, meaning that a total of approximately 2^{40} 8-round DES encryptions are required to recover the whole 56 key bits (Note that there might exist an efficient way to obtain the remaining key bits). In 2002, under Assumptions 1, 2 and 3, Biham, Dunkelman and Keller [6] described a 7-round differential-linear distinguisher of DES using their enhanced methodology, and finally gave differential-linear attacks on 8 and 9-round DES; and an improved version of the 9-round attack appeared in pages 108–111 of [18]. Their attack recovers 18 key bits with a time complexity of $2^{29.17}$ 9-round DES encryptions, the remaining 38 key bits would take 2^{38} encryptions to recover with a key exhaustion, and thus it has a total of approximately 2^{38} 9-round DES encryptions to recover the whole 56 key bits.

Nevertheless, we find that our new methodology enables us to construct 7 and 8-round differential-linear distinguishers of DES based on the same 3-round linear approximation as used in the previous differential-linear cryptanalysis of DES [6, 31]; the 8-round distinguisher can allow us to break 10-round DES. More importantly, we are able to construct a 11-round differential-linear distinguisher of DES, and finally use it as the basis of a differential-linear attack on 13-round DES. Below we describe the 11-round differential-linear distinguisher and our attack on 13-round DES. We write the subkey used in the S_l S-box of Round m as $K_{m,l}$, where $1 \leq m \leq 16, 1 \leq l \leq 8$.

4.1 A 11-Round Differential-Linear Distinguisher with Bias $2^{-24.05}$

The 11-round differential-linear distinguisher is made up of a 6-round linear approximation $\varGamma\gamma \to \varGamma\delta$ with bias $1.95 \times 2^{-9} \approx 2^{-8.04}$ and all the 5-round differentials $\{\varDelta\alpha \to \varDelta\beta\}$ with $\varDelta\alpha = 0x4000000000000000$. The 6-round linear approximation $\varGamma\gamma \to \varGamma\delta$ is $0x0000000001040080 \to 0x2104008000008000$, (which is the best 6-round linear approximation given in [34]). Let's compute the probability of the 11-round differential-linear distinguisher using our new methodology.

We first consider the 5-round differentials $\{\varDelta\alpha \to \varDelta\beta\}$. There is a one probability in the first round, meaning that the first round is bypassed by the differential characteristic with probability 1. After the $\mathbf{E}$ expansion operation of the second round, $0x4$ in $\varDelta\alpha$ becomes $0x8$, which enters the S_1 S-box of the second round and generates 11 differences after the S-box: $\{\omega|\omega = 0x3, 0x5, 0x6, 0x7, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF\}$; the probabilities for these output differences are given in the second column of Table 2. We represent ω as a concatenation of four one-bit variables $a||b||c||d$, where $a, b, c, d \in \{0, 1\}$. Thus, the right half of the third round has the input difference $00000000a0000000b00000c0000000d0$ in binary notation, and this input difference can make at most 6 S-boxes of the third round active: $\mathrm{S}_2, \mathrm{S}_3, \mathrm{S}_4, \mathrm{S}_5, \mathrm{S}_6, \mathrm{S}_8$.

In the third round, the S_2 S-box has an input difference $00000a$ in binary notation, the S_3 S-box has an input difference $0a0000$ in binary notation, the S_4 S-box has an input difference $00000b$ in binary notation, the S_5 S-box has an input difference $0b0000$ in binary notation, the S_6 S-box has an input difference $000c00$ in binary notation, and the S_8 S-box has an input difference $000d00$ in binary notation. We denote respectively by x_0, x_1, x_2 the most significant bit, the second most significant bit and the second least significant bit of the output difference of the S_2 S-box, by $x_3||x_4||x_5||x_6$ the output difference of the S_3 S-box, by x_7, x_8, x_9 the second most significant bit, the second least significant bit and the least significant bit of the output difference of the S_4 S-box, by $x_{10}||x_{11}||x_{12}||x_{13}$ the output difference of the S_5 S-box, by x_{14}, x_{15}, x_{16} the most significant bit, the second most significant bit and the second least significant bit of the output difference of the S_6 S-box, and by x_{17}, x_{18}, x_{19} the most significant bit, the second least significant bit and the least significant bit of the output difference of the S_8 S-box.

In the fourth round, the S_1 S-box has the input difference $0||x_9||(x_2 \oplus 1)||x_{13}||\ x_{14}||x_{17}$, and we denote by y_0 the second most significant bit of its output difference; the S_2 S-box has the input difference $x_{14}||x_{17}||x_6||0||x_{10}||0$, and we denote by y_1 the least significant bit of its output difference; the S_3 S-box has the input difference $x_{10}||0||x_8||x_{16}||0||x_0$, and we denote by y_2 the second most significant bit of its output difference; the S_4 S-box has the input difference $0||x_0||x_{11}||x_{18}||x_4||0$, and we denote by y_3 the second most significant bit of its output difference; the S_6 S-box has the input difference $x_7||x_{19}||0||0||x_3||x_{12}$, and we denote by y_4 the least significant bit of its output difference; the S_8 S-box has the input difference $x_1||x_{15}||x_5||0||0||x_9$, and we denote by y_5 the least significant bit of its output difference. Thus we have that the input difference of the S_5 S-box of the fifth round is $y_2||(y_0 \oplus b)||y_1||y_4||y_3||y_5$.

A simple analysis reveals that the three bits concerned by the input mask $\varGamma\gamma$ depend on: (1) x_{10}, x_{11} and x_{12}; and (2) The three most significant bits of the output difference of the S_5 S-box of the fifth round; and we denote the XOR of the three bits by z.

Table 2. Probabilities for the eleven output differences in $\{\omega\}$

ω	$\Pr_{\mathrm{S}_1}(\varDelta 0x8 \to \varDelta\omega)$	$\Pr(\varDelta\beta_\omega \odot \varGamma\gamma = 0 \vert \varDelta 0x8 \to \varDelta\omega)$
$0x3$	$\frac{12}{64}$	0.49779944866895676
$0x5$	$\frac{8}{64}$	0.49595199525356293
$0x6$	$\frac{8}{64}$	0.50433863041689619
$0x7$	$\frac{4}{64}$	0.50256029706542904
$0x9$	$\frac{6}{64}$	0.50855094581311278
$0xA$	$\frac{2}{64}$	0.50591027818154544
$0xB$	$\frac{8}{64}$	0.50239421910760029
$0xC$	$\frac{8}{64}$	0.49929085310759547
$0xD$	$\frac{2}{64}$	0.49968796220765910
$0xE$	$\frac{2}{64}$	0.50061782109781916
$0xF$	$\frac{4}{64}$	0.50005227406592345

For each difference ω, we denote by β_ω the output difference(s) of the 5-round DES. Now, by the differential distribution tables of the S-boxes (see [11]) we can compute the probability that the XOR of the concerned three bits of β_ω (i.e., $x_{10} \oplus x_{11} \oplus x_{12} \oplus z$) is zero by performing a computer program over all the possible (truncated) differential characteristics. These probabilities are given in the third column of Table 2. The largest number of possible differential characteristics happens when $\omega = 0xF$, which is $7 \times 10 \times 4 \times 10 \times 6 \times 7 \times 2^6 \times 2 \approx 2^{23.9}$; and it takes a few seconds to check on a personal computer.

Finally, by Theorem 1 we have that the probability of the 11-round distinguisher $\varDelta\alpha \to \varGamma\delta$ is $\frac{1}{2} + 2 \times [2 \times \sum_\omega \Pr_{\mathrm{S}_1}(\varDelta 0x8 \to \varDelta\omega) \times \Pr(\varDelta\beta_\omega \odot \varGamma\gamma = 0 \vert \varDelta 0x8 \to \varDelta\omega) - 1] \times (2^{-8.04})^2 \approx \frac{1}{2} + 2 \times 2^{-8.97} \times (2^{-8.04})^2 \approx \frac{1}{2} + 2^{-24.05}$. Therefore, the 11-round distinguisher has a bias of $2^{-24.05}$.

4.2 Differential-Linear Attack on 13-Round DES

The 11-round distinguisher $\varDelta\alpha \to \varGamma\delta$ can be used to break 13-round DES. We assume the attacked rounds are the first thirteen rounds from Rounds 1 to 13. A simple analysis on the key schedule of DES reveals that $K_{1,1}$ and $K_{13,1}$ overlap in 2 bits (i.e. bits 17 and 34 of the user key), and thus given $K_{1,1}$ we know 2 bits of $K_{13,1}$. The attack procedure is as follows.

1. Choose $2^{47.1}$ structures $\mathcal{S}_i$, $(i = 1, 2, \cdots, 2^{47.1})$, where a structure is defined to be a set of 2^4 plaintexts $P_{i,j}$ with bits (9,17,23, 31) of the left half taking all the possible values, bit (2) of the right half fixed to 0 and the other 59 bits fixed, $(j = 1, 2, \cdots, 2^4)$. In a chosen-plaintext attack scenario, obtain all the ciphertexts for the 2^4 plaintexts in each of the $2^{47.1}$ structures; we denote by $C_{i,j}$ the ciphertext for plaintext $P_{i,j}$.
2. Choose $2^{47.1}$ structures $\widehat{\mathcal{S}_i}$, $(i = 1, \cdots, 2^{47.1})$, where a structure $\widehat{\mathcal{S}_i}$ is obtained by setting 1 to bit (2) of the right half of all the plaintexts $P_{i,j}$ in $\mathcal{S}_i$. In a chosen-plaintext attack scenario, obtain all the ciphertexts for the 2^4 plaintexts in each $\widehat{\mathcal{S}_i}$.
3. Guess a value for $K_{1,1}$, and do as follows.
   (a) Initialize 2^{20} counters to zero, which correspond to the 2^{20} possible pairs consisting of the possible values for a couple of the 10 ciphertext bits: bit (17) of the left half and bits (1,2,3,4,5,8,14,25,32) of the right half.
   (b) Partially encrypt every (remaining) plaintext $P_{i,j}$ with the guessed $K_{1,1}$ to get its intermediate value immediately after Round 1; we denote it by $\varepsilon_{i,j}$.
   (c) Partially decrypt $\varepsilon_{i,j} \oplus 0x4000000000000000$ with the guessed $K_{1,1}$ to get its plaintext, and find the plaintext in $\widehat{\mathcal{S}_i}$; we denote it by $\widehat{P_{i,j}}$, and denote by $\widehat{C_{i,j}}$ the corresponding ciphertext for $\widehat{P_{i,j}}$. Store $(C_{i,j}, \widehat{C_{i,j}})$ in a table.

(d) For every ciphertext pair $(C_{i,j}, \widehat{C}_{i,j})$, add 1 to the counter corresponding to the pair of the 10 ciphertext bits specified by $(C_{i,j}, \widehat{C}_{i,j})$.

(e) Guess a value for the unknown 4 bits of $K_{13,1}$, and do as follows.

i. For each of the 2^{20} pairs of the concerned 10 ciphertext bits, partially decrypt it with the guessed $K_{13,1}$ to get the pair of the 5 bits concerned by the output mask $\varGamma\delta$, and compute the XOR of the pair of the 5 bits (concerned by the output mask).

ii. Count the number of the ciphertext pairs $(C_{i,j}, \widehat{C}_{i,j})$ such that the XOR of the pair of the 5 bits concerned by $\varGamma\delta$ is zero, and compute its deviation from $2^{50.1}$.

iii. If the guess for $(K_{1,1}, K_{13,1})$ is the first guess for $(K_{1,1}, K_{13,1})$, then record the guess and the deviation computed in Step 3(e)(ii); otherwise, record the guess and its deviation only when the deviation is larger than that of the previously recorded guess, and remove the guess with the smaller deviation.

4. For the $(K_{1,1}, K_{13,1})$ recorded in Step 3(e)(iii), exhaustively search for the remaining 46 key bits with two known plaintext/ciphertext pairs. If a 56-bit key is suggested, output it as the user key of the 13-round DES.

The attack requires $2^{52.1}$ chosen plaintexts. The required memory for the attack is dominated by the storage of the plaintexts and ciphertexts, which is $2^{52.1} \times 16 = 2^{56.1}$ bytes. Steps 1 and 2 have a time complexity of $2^{52.1}$ 13-round DES encryptions. Steps 3(b) and 3(c) have a time complexity of $2 \times 2^{51.1} \times 2^6 \times \frac{1}{8\times13} \approx 2^{51.4}$ 13-round DES encryptions. Step 3(d) has a time complexity of $2^{51.1} \times 2^6 = 2^{57.1}$ memory accesses. Roughly, an extremely conservative estimate is: 13 memory accesses equal a 13-round DES encryption in terms of time, assuming that the 13-round DES is implemented with 8 parallel S-box lookups per round and one round is equivalent to one memory access. So the time complexity of Step 3(d) is equivalent to $\frac{2^{57.1}}{13} \approx 2^{53.4}$ 13-round DES encryptions. The time complexity of Step 3(e) is dominated by the time complexity of Step 3(e)(i), which is $2 \times 2^6 \times 2^4 \times 2^{20} \times \frac{1}{8\times13} \approx 2^{24.3}$ 13-round DES encryptions. Step 4 has a time complexity of 2^{46} 13-round DES encryptions. Therefore, the attack has a total time complexity of approximately $2^{54.2}$ 13-round DES encryptions, faster than exhaustive key search. There are $2^{51.1}$ plaintext pairs $(P_{i,j}, \widehat{P}_{i,j})$ for a guess of $(K_{1,1}, K_{13,1})$, and thus following Theorem 2 of [39], we can know that the attack has a success probability of about 99%.

This shows that our new methodology enables us to break more rounds of DES than Biham et al.'s or Langford and Hellman's methodology. Since our attack works under only two assumptions, it is more reasonable than Biham et al.'s attack.

Note. Using the new methodology we can obtain a few differential-linear distinguishers operating on a smaller number of rounds, for example, a 7-round distinguisher ($\varDelta\alpha = 0x4000000000000000, \varGamma\delta = 0x2104008000008000$) with bias $2^{-7.94}$ and an 8-round distinguisher ($\varDelta\alpha = 0x4000000000000000, \varGamma\delta = 0x2104008000008000$) with bias $2^{-12.83}$, both using the same 3-round linear approximation as used in Biham et al.'s and Langford and Hellman's differential-linear cryptanalysis of DES. These distinguishers can allow us to break DES with a smaller number of rounds at a smaller complexity, for example, the 8-round distinguisher can similarly be used to break 10-round DES with a data complexity of $2^{29.66}$ chosen plaintexts and a time complexity of 2^{44} 10-round DES encryptions at a success rate of about 99%.

5 Application to the CTC2 Block Cipher

The CTC2 [14] cipher was designed to show the strength of algebraic cryptanalysis [15] on block ciphers by the proposer of algebraic cryptanalysis, who described an algebraic attack on 6 rounds of the version of CTC2 that uses a 255-bit block size and a 255-bit key. Using Biham et al.'s methodology, in 2009 Dunkelman and Keller [19] described 6 and 7-round differential-linear distinguishers for the version of CTC2, and finally presented differential-linear attacks on 7 and 8 rounds of CTC2 (with a 255-bit block size and key). The 8-round attack is known as the best previously published cryptanalytic result on the version of CTC2 in terms of the numbers of attacked rounds.

In this section, first we describe a flaw in the previous differential-linear cryptanalysis of CTC2. Then, under the new methodology we present a 8.5-round differential-linear distinguisher with bias 2^{-68} for the CTC2 with a 255-bit block size and key, and finally give a differential-linear attack on 10-round CTC2 (with a 255-bit block size and a key). We first briefly describe the CTC2 cipher.

5.1 The CTC2 Block Cipher

The CTC2 [14] block cipher has a variable block size, a variable length key and a variable number of rounds. There are many combinations for the block size, key size and round number. As in [19], we only consider the version of CTC2 that uses a 255-bit block size and a 255-bit key. CTC2 uses the following two elementary operations to construct its round function.

- **S** is a non-linear substitution operation constructed by applying the same 3×3-bit bijective S-box 85 times in parallel to an input.
- **D** is a linear diffusion operation, which takes a 255-bit block $Y = (Y_{254}, \cdots, Y_1, Y_0)$ as input, and outputs a 255-bit block $Z = (Z_{254}, \cdots, Z_1, Z_0)$, computed as defined below.

$$\begin{cases} Z_{151} = Y_2 \oplus Y_{139} \oplus Y_{21} \\ Z_{(i \times 202 + 2) \bmod 255} = Y_i \oplus Y_{(i+137) \bmod 255} \qquad i = 0, 1, 3, 4, \cdots, 254 \end{cases}$$

CTC2 takes as input a 255-bit plaintext block P, and its encryption procedure for N_r rounds is, where $Z_0, X_i, Y_i, Z_i,\ X_{N_r}, Y_{N_r}, Z_{N_r}$ are 255-bit variables, and K_0, K_i, K_{N_r} are round keys generated from a user key K as $K_j = K \lll j$ in our notation, $(0 \leq j \leq N_r)$.

```
1. $Z_0 = P$.
2. For $i = 1$ to $N_r - 1$:
   – $X_i = Z_{i-1} \oplus K_{i-1}$,
   – $Y_i = \mathbf{S}(X_i)$,
   – $Z_i = \mathbf{D}(Y_i)$.
3. $X_{N_r} = Z_{N_r-1} \oplus K_{N_r-1}$, $Y_{N_r} = \mathbf{S}(X_i)$, $Z_{N_r} = \mathbf{D}(Y_{N_r})$.
4. Ciphertext $= Z_{N_r} \oplus K_{N_r}$.
```

To keep in accordance with [14], the ith iteration of Step 2 in the above description is referred to as Round i, $(1 \leq i \leq N_r - 1)$, and the transformations in Steps 3 and 4 are referred to as Round N_r. We number the 85 S-boxes in a round from 0 to 84 from right to left.

5.2 A Flaw in Previous Differential-Linear Cryptanalysis of CTC2

Observe that Dunkelman and Keller used the 0.5-round differential $e_{30,151} \overset{\mathbf{D}}{\rightarrow} e_2$ with probability 1 in their differential-linear attacks presented in [19]. However, we find that this differential is not correct: For the **D** operation, given the input difference $e_{30,151}$, we cannot get the output difference e_2; and the correct output difference should be $e_{25,63,159,197}$. On the other hand, for the **D** operation, given the output difference e_2, the input difference has over fifty non-zero bits, much more than the number two in $e_{30,151}$. As a consequence, the differential-linear cryptanalytic results are flawed.

Note that Dunkelman and Keller also described differential attacks on 5, 6 and 7-round CTC2 in [19], and this 0.5-round differential $e_{30,151} \overset{\mathbf{D}}{\rightarrow} e_2$ with probability 1 was also used and played a very important role in the differential results, thus they are flawed, too. It seems very hard to correct those differential and differential-linear cryptanalytic results.

5.3 A 8.5-Round Differential-Linear Distinguisher with Bias 2^{-68}

The 8.5-round differential-linear distinguisher with bias 2^{-68} is made up of a 5.5-round linear expression $\mathit{\Gamma}\gamma \to \mathit{\Gamma}\delta$ with bias 2^{-33} and all the 3-round differentials $\{\mathit{\Delta}\alpha \to \mathit{\Delta}\beta\}$ with $\mathit{\Delta}\alpha = e_0$. The 5.5-round linear expression $\mathit{\Gamma}\gamma \to \mathit{\Gamma}\delta$ is $e_{5,33,49,54,101,112,131,138,155,168,188,193,217,247,251} \to e_{32,151}$. Using the new methodology we can compute that the 8.5-round distinguisher $\mathit{\Delta}\alpha \to \delta$ has a bias of 2^{-68}, in a manner similar to that for the above 11-round DES distinguisher.

5.4 Differential-Linear Attack on 10-Round CTC2 with a 255-Bit Block Size and Key

The above 8.5-round distinguisher can be used as the basis for a differential-linear attack breaking the version of CTC2 that has a 255-bit block size, a 255-bit key and a total of 10 rounds.

We assume the attacked rounds are the first ten rounds from Rounds 1 to 10; and we use the distinguisher from Rounds 2 until before the **D** operation of Round 10. We can learn that the input difference α propagates to 16 bit positions after the inverse of the **D** operation of Round 1: Bits 17, 21, 40, 59, 78, 97, 116, 135, 139, 154, 158, 177, 196, 215, 234 and 253. The 16 active bits correspond to 16 S-boxes of Round 0: S-boxes 5, 7, 13, 19, 26, 32, 38, 45, 46, 51, 52, 59, 65, 71, 78 and 84; let $\mathit{\Theta}$ be the set of the 16 S-boxes, and $K_{\mathit{\Theta}}$ be the 48 bits of K_0 corresponding to the 16 S-boxes in $\mathit{\Theta}$. Another observation is that we do not need to guess the subkey bits from K_{10}, because the output mask $\mathit{\Gamma}\delta$ of the 8.5-round distinguisher concerns the intermediate value immediately after the **S** operation of Round 10, and for a pair of ciphertexts $(C, \widehat{C})$ the value of $\delta \odot \mathbf{D}^{-1}(C) \oplus \delta \odot \mathbf{D}^{-1}(\widehat{C})$ equals to $\delta \odot \mathbf{D}^{-1}(C \oplus \widehat{C})$, which is independent of K_{10}. The attack procedure is as follows.

1. Choose 2^{94} structures $\mathcal{S}_i$, $(i = 0, 1, \cdots, 2^{94} - 1)$, where a structure is defined to be a set of 2^{48} plaintexts $P_{i,j}$ with the 48 bits for the S-boxes in $\mathit{\Theta}$ taking all the possible values and the other 207 bits fixed, $(j = 0, 1, \cdots, 2^{48} - 1)$. In a chosen-plaintext attack scenario, obtain all the ciphertexts for the 2^{48} plaintexts in each of the 2^{94} structures; we denote by $C_{i,j}$ the ciphertext for plaintext $P_{i,j}$.
2. Initialize 2^{48} counters to zero, which correspond to all the possible values for $K_{\mathit{\Theta}}$.
3. For every structure $\mathcal{S}_i$, guess a value for $K_{\mathit{\Theta}}$, and do as follows.
   (a) Partially encrypt every (remaining) plaintext $P_{i,j}$ with the guessed $K_{\mathit{\Theta}}$ to get its intermediate value immediately after the **S** operation of Round 1; we denote it by $\varepsilon_{i,j}$.
   (b) Take bitwise complements to bits (17, 21, 40, 59, 78, 97, 116, 135, 139, 154, 158, 177, 196, 215, 234, 253) of $\varepsilon_{i,j}$, and keep the other bits of $\varepsilon_{i,j}$ invariant; we denote the resulting value by $\widehat{\varepsilon}_{i,j}$.
   (c) Partially decrypt $\widehat{\varepsilon}_{i,j}$ with the guessed $K_{\mathit{\Theta}}$ to get its plaintext, and find the plaintext in $\mathcal{S}_i$; we denote it by $\widehat{P}_{i,j}$, and denote by $\widehat{C}_{i,j}$ the corresponding ciphertext for $\widehat{P}_{i,j}$.
   (d) For $(C_{i,j}, \widehat{C}_{i,j})$, compute the XOR of bits 32 and 151 of $\mathbf{D}^{-1}(C_{i,j} \oplus \widehat{C}_{i,j})$. If the XOR is zero, add 1 to the counter corresponding to the guessed $K_{\mathit{\Theta}}$.
4. For the $K_{\mathit{\Theta}}$ with the highest deviation from 2^{140}, exhaustively search for the remaining 207 key bits with a known plaintext/ciphertext pair. If a 255-bit key is suggested, output it as the user key of CTC2.

The attack requires 2^{142} chosen plaintexts. Note that we start to collect another structure of plaintexts only after testing a structure of plaintexts, so that we can reuse the memory for storing the structure of plaintexts, hence the required memory of the attack is dominated by the storage of the 2^{48} counters and a structure of 2^{48} plaintext-ciphertext pairs, which is $2^{48} \times \frac{48}{8} + 2 \times 2^{48} \times \frac{255}{8} \approx 2^{54.2}$ bytes of memory. The time complexity of Step 3 is dominated by the time complexity of Steps 3(a), 3(c) and 3(d), which is approximately $2 \times 2^{141} \times 2^{48} \times \frac{16}{85 \times 10} + 2^{141} \times 2^{48} \times \frac{1}{10} \approx 2^{186.2}$

10-round CTC2 encryptions. Step 4 has a time complexity of 2^{207} 10-round CTC2 encryptions. Therefore, the attack has a total time complexity of 2^{207} 10-round CTC2 encryptions to find the 255-bit key. There are 2^{141} plaintext pairs $(P_{i,j}, \widehat{P}_{i,j})$ for a guess of K_{Θ}. Following Theorem 2 of [39], we can learn that the probability that the correct guess for K_{Θ} is recorded in Step 2(f) is about 99.9%. Thus, the attack has a success probability of about 99.9%.

6 Application to the Serpent Block Cipher

The Serpent [1] block cipher is one of the five Advanced Encryption Standard (AES) finalists, second to the Rijndael [16] cipher that was selected as the AES [38]. Serpent was designed in a rather conservative way, and it was included in the GNU project [21] for possible use in cryptographic applications in reality such as SNMP (Simple Network Management Protocol), LDAP (Lightweight Directory Access Protocol) and X.509 certificates.

In 2003, Biham et al. [8] described a 9-round differential-linear distinguisher of Serpent, and finally gave a differential-linear attack on 11-round Serpent with a 256-bit key. In 2008 Dunkelman et al. [20] presented an improved 9-round differential-linear distinguisher of Serpent, and finally used it as the basis for a differential-linear attack on 12-round Serpent with a 256-bit key. All these attacks are based on Biham et al.'s methodology. In terms of the numbers of attacked rounds, the 12-round attack is known as the best previously published cryptanalytic result on Serpent.

In this section, we present a 9-round differential-linear distinguisher with bias $2^{-59.41}$ under our new methodology, which can be used to break 12-round Serpent (with a 256-bit key) slightly faster than Dunkelman et al.'s attack at a higher success rate. We first briefly describe the Serpent block cipher.

6.1 The Serpent Block Cipher

The Serpent [1] block cipher has a 128-bit block size, a variable length key of up to 256 bits, and a total of 32 rounds; a shorter key can be used by appending one "1" bit to the most significant bit end, followed by as many "0" bits as required. Serpent uses the following elementary operations:

- **IP/FP** is the initial/final permutation; see [1] for their specifications.
- $\mathbf{S}_i$ is a non-linear substitution operation constructed by applying the same 4×4-bit bijective $\mathrm{S}_{i \bmod 8}$ S-box 32 times in parallel to an input, $(0 \leq i \leq 31)$. Refer to [1] for specifications of the S-boxes $\mathrm{S}_0, \mathrm{S}_1, \cdots, \mathrm{S}_7$.
- **L** is a linear diffusion operation, which takes as input a 128-bit block of four 32-bit words $X = (X_3, X_2, X_1, X_0)$, and outputs a 128-bit block of four 32-bit words $Y = (Y_3, Y_2, Y_1, Y_0)$, computed as follows.
  - $X_0 = X_0 \lll 13$,
  - $X_2 = X_2 \lll 3$,
  - $X_1 = X_0 \oplus X_1 \oplus X_2$,
  - $X_3 = X_3 \oplus X_2 \oplus (X_0 \ll 3)$,
  - $X_1 = X_1 \lll 1$,
  - $X_3 = X_3 \lll 7$,
  - $X_0 = X_0 \oplus X_1 \oplus X_3$,
  - $X_2 = X_2 \oplus X_3 \oplus (X_1 \ll 7)$,
  - $X_0 = X_0 \lll 5$,
  - $X_2 = X_2 \lll 22$,
  - $Y = (X_3, X_2, X_1, X_0)$.

Serpent takes as input a 128-bit plaintext block P, and its encryption procedure is, where $\widehat{B}_0, \widehat{B}_1, \cdots, \widehat{B}_{32}$ are 128-bit variables, and $K_0, K_1, \cdots, K_{32}$ are round keys.

1. $\widehat{B}_0 = \mathbf{IP}(P)$.
2. For $i = 0$ to 30:
   - $\widehat{B}_{i+1} = \mathbf{L}(\mathbf{S}_i(\widehat{B}_i \oplus K_i))$.
3. $\widehat{B}_{32} = \mathbf{S}_{31}(\widehat{B}_{31} \oplus K_{31}) \oplus K_{32}$.
4. Ciphertext $= \mathbf{FP}(\widehat{B}_{32})$.

The ith iteration of Step 2 in the above description is referred to below as Round i, $(0 \leq i \leq 30)$, and the transformation in Steps 3 and 4 is referred to below as Round 31; this is in accordance with [1]. We number the 32 S-boxes of a round from 0 to 31 from right to left. For simplicity, we describe a state S in a Serpent encryption operation as four 32-bit words (s_3, s_2, s_1, s_0), and write it as $(s_{3,31}||s_{2,31}||s_{1,31}||s_{0,31})|| \cdots ||(s_{3,1}|| \ s_{2,1}||s_{1,1}||s_{0,1})||(s_{3,0}||s_{2,0}||s_{1,0}||s_{0,0})$, where $s_{j,l}$ is the l-th bit of s_j, $(0 \leq j \leq 3, 0 \leq l \leq 31)$. We write $K_{i,m}$ for the 4-bit subkey of K_i that corresponds to S-box m of Round i, $(0 \leq m \leq 31)$. As the **IP** and **FP** operations are simply linear diffusion transformations, we omit them in our analysis.

6.2 A 9-Round Differential-Linear Distinguisher with Bias $2^{-59.41}$

The 9-round differential-linear distinguisher is made up of a 6-round linear approximation $\mathit{\Gamma}\gamma \rightarrow \mathit{\Gamma}\delta$ with bias 2^{-27} for Rounds 5 to 10 and all the 3-round differentials $\{\mathit{\Delta}\alpha \rightarrow \mathit{\Delta}\beta\}$ for Rounds 2 to 4 with $\mathit{\Delta}\alpha = 0x000000A0000000000000000000000000$. The 6-round linear approximation $\mathit{\Gamma}\gamma \rightarrow \mathit{\Gamma}\delta$ is $0x00400000000000000000000000000002 \rightarrow 0x000B0000B000030000B0200E00000010$. Finally, we can similarly use the new methodology to compute that the 9-round differential-linear distinguisher $\mathit{\Delta}\alpha \rightarrow \mathit{\Gamma}\delta$ has a bias of $2^{-59.41}$.

6.3 Differential-Linear Attack on 12-Round Serpent

The 9-round differential-linear distinguisher enables us to construct a differential-linear attack on 12-round Serpent (with a 256-bit key). We attack Rounds 0 to 11, and use the distinguisher from Rounds 2 to 10. The input difference α becomes $0x000000A2040008000000000000000000$ after being applied the $\mathbf{L}^{-1}$ operation of Round 1, and the 5 active bits correspond to S-boxes 18, 22, 24 and 25 of Round 1. It makes 27 active S-boxes of Round 0: S-boxes 0, 2, 3, 4, 5, 6, 7, 9, 11, 12, 13, 15, 16, $\cdots$, 29 and 31; let $\mathit{\Theta}$ be the set of the 27 S-boxes, and $K_{\mathit{\Theta}}$ be the 108 bits of K_0 corresponding to the 27 S-boxes in $\mathit{\Theta}$. The 16 bits concerned by the output mask correspond to S-boxes 1, 8, 11, 13, 18, 23 and 28 of Round 11. The attack procedure is as follows, where the values of parameters λ and ϕ will be specified in the subsequent analysis.

1. Choose λ structures $\mathcal{S}_i$, $(i = 0, 1, \cdots, \lambda - 1)$, where a structure is defined to be a set of 2^{108} plaintexts $P_{i,j}$ with the 108 bits for the 27 S-boxes in $\mathit{\Theta}$ taking all the possible values and the other 20 bits fixed, $(j = 0, 1, \cdots, 2^{108} - 1)$. In a chosen-plaintext attack scenario, obtain all the ciphertexts for the 2^{108} plaintexts in each of the λ structures; we denote by $C_{i,j}$ the ciphertext for plaintext $P_{i,j}$.
2. Guess a value for $(K_{\mathit{\Theta}}, K_{1,18}, K_{1,22}, K_{1,24}, K_{1,25})$, and do as follows.
   (a) Initialize 2^{56} counters to zero, which correspond to the 2^{56} possible pairs of the 28 ciphertext bits corresponding to S-boxes 1, 8, 11, 13, 18, 23 and 28 of Round 11.
   (b) Partially encrypt every (remaining) plaintext $P_{i,j}$ with the guessed $(K_{\mathit{\Theta}}, K_{1,18}, K_{1,22}, K_{1,24}, K_{1,25})$ to get its intermediate value immediately after the $\mathbf{S}$ operation of Round 1; we denote it by $\varepsilon_{i,j}$.

(c) Compute $\varepsilon_{i,j} \oplus 0x000000A2040008000000000000000000$, and we denote the resulting value by $\widehat{\varepsilon}_{i,j}$.

(d) Partially decrypt $\widehat{\varepsilon}_{i,j}$ with the guessed $(K_{\Theta}, K_{1,18}, K_{1,22}, K_{1,24}, K_{1,25})$ to get its plaintext, and find the plaintext in $\mathcal{S}_i$; we denote it by $\widehat{P}_{i,j}$, and denote by $\widehat{C}_{i,j}$ the corresponding ciphertext for $\widehat{P}_{i,j}$.

(e) For every ciphertext pair $(C_{i,j}, \widehat{C}_{i,j})$, add 1 to the counter corresponding to the pair of the 28 ciphertext bits specified by $(C_{i,j}, \widehat{C}_{i,j})$.

(f) Guess a value for $(K_{12,1}, K_{12,8}, K_{12,11}, K_{12,13}, K_{12,18}, K_{12,23}, K_{12,28})$, and do as follows.

   i. For each of the 2^{56} pairs of the concerned 28 ciphertext bits, partially decrypt it with the guessed $(K_{12,1}, K_{12,8}, \cdots, K_{12,28})$ to get the pair of the 16 bits concerned by the output mask, and compute the XOR of the pair of the 16 bits (concerned by the output mask).

   ii. Count the number of the ciphertext pairs $(C_{i,j}, \widehat{C}_{i,j})$ such that the XOR of the pair of the 16 bits concerned by the output mask is zero, and compute its deviation from $\lambda \cdot 2^{107}$.

   iii. If the guessed $(K_{\Theta}, K_{1,18}, K_{1,22}, K_{1,24}, K_{1,25}, K_{12,1}, K_{12,8}, \cdots, K_{12,28})$ belong to the first ϕ guesses for $(K_{\Theta}, K_{1,18}, K_{1,22}, K_{1,24}, K_{1,25}, K_{12,1}, K_{12,8}, \cdots, K_{12,28})$, then record the guess and the deviation computed in Step 2(f)(ii); otherwise, record the guess and its deviation only when the deviation is larger than the smallest deviation of the previously recorded ϕ guesses, and remove the guess with the smallest deviation from the ϕ guesses.

3. For every recorded $(K_{\Theta}, K_{1,18}, K_{1,22}, K_{1,24}, K_{1,25})$ in Step 2(f)(iii), exhaustively search for the remaining 132 key bits with two known plaintext-ciphertext pairs. If a 256-bit key is suggested, output it as the user key of the 12-round Serpent.

The attack requires $\lambda \times 2^{108}$ chosen plaintexts. The required memory for the attack is dominated by the storage of the plaintexts and ciphertexts, which is $\lambda \times 2^{108} \times 32 = \lambda \times 2^{113}$ bytes. The time complexity of Step 2 is dominated by the time complexity of Steps 2(b), 2(d) and 2(f)(i), which is $\lambda \times 2 \times 2^{107} \times 2^{124} \times \frac{27+4}{32\times 12} + 2 \times 2^{124} \times 2^{28} \times 2^{56} \times \frac{7}{32\times 12} \approx \lambda \times 2^{228.37}$ 12-round Serpent encryptions. Step 3 has a time complexity of at most $\phi \times 2^{132}$ 12-round Serpent encryptions. There are $\lambda \times 2^{107}$ plaintext pairs $(P_{i,j}, \widehat{P}_{i,j})$ for a guess of $(K_{\Theta}, K_{1,18}, K_{1,22}, K_{1,24}, K_{1,25}, K_{12,1}, K_{12,8}, \cdots, K_{12,28})$. Following Theorem 2 of [39], we have that the probability that the correct guess of $(K_{\Theta}, K_{1,18}, K_{1,22}, K_{1,24}, K_{1,25}, K_{12,1}, K_{12,8}, \cdots, K_{12,28})$ is recorded in Step 2(f)(iii) is about 96.6% when we set $\lambda = 2^{18.8}$ and $\phi = 1$, and is about 98.8% when we set $\lambda = 2^{16.5}$ and $\phi = 2^{104}$. Thus, when $\lambda = 2^{16.5}$ and $\phi = 2^{104}$, with a success probability of about 98.8% the attack requires $2^{124.5}$ chosen plaintexts, and has a total time complexity of approximately $2^{244.9}$ 12-round Serpent encryptions.

Note. For the purpose of then AES submission requirements, the Serpent designers also considered the cases of 128 and 192-bit keys, and we denote these versions by Serpent-128/192, respectively. There are some published cryptanalytic results on Serpent-128/192, and we are particularly interested in those differential-linear cryptanalytic results: Biham et al.'s and Dunkelman et al.'s differential-linear attacks on 10-round Serpent-128 and 11-round Serpent-192 given in [8, 20]. All these attacks are based on Biham et al.'s methodology. A detailed analysis shows that our 9-round differential-linear distinguisher with bias $2^{-59.41}$ can also be used to break 10-round Serpent-128 and 11-round Serpent-192; and more results can be obtained, in particular, we can break 10-round Serpent-128 with a data complexity of $2^{123.4}$ chosen plaintexts and a time complexity of $2^{123.4}$ 10-round Serpent encryptions at a success rate of 99.2%, and break 11-round Serpent-192 with a data complexity of $2^{125.5}$ chosen plaintexts and a time complexity of $2^{148.1}$ 11-round Serpent encryptions at a success rate of 99%.

7 Possible Extensions of Our Methodology

In this section we briefly discuss several possible extensions of our methodology, although particulars should be noticed.

The first possible extension is to consider the case when using two different values for the output mask δ in Definition 3, say δ_1, δ_2; that is, we might consider the event $\mathbb{E}(P) \odot \delta_1 = \mathbb{E}(P \oplus \alpha) \odot \delta_2$ for a randomly chosen $P \in \{0,1\}^n$. The resulting differential-linear distinguisher would have a bias of $2(2\widehat{p} - 1)\epsilon_1\epsilon_2$ for some ϵ_1 and ϵ_2 denoting the respective bias of the two linear approximations. From a theoretical point of view, there seems no need to use two different output masks, for we can always choose the output mask with a bigger bias, and a key-recovery attack based on a differential-linear distinguisher with two different output masks requires us to guess no less key bits than that based on a differential-linear distinguisher with one output mask; however, the case with two different output masks may depend on Assumption 2 to a lesser degree than the above discussed case with one output mask, for the two linear approximations can be independent somewhat, instead of two identical linear approximations used in the case with one output mask, and thus it may potentially be particularly helpful when making a practicable attack in reality.

The second possible extension is to consider the case when applying our methodology in a related-key [2, 26] attack scenario. The notion of the related-key differential-linear analysis appeared in [23], and later Kim [25] described an enhanced version based on Biham et al.'s enhanced methodology. Likewise, we can get a more reasonable and general version based on our new methodology.

Other possible extensions are to obtain new methodologies for the high-order differential-linear attack, the differential-bilinear attack and the differential-bilinear-boomerang attack, which were proposed in [9], in a way similar to the above new methodology for differential-linear cryptanalysis. At present, however, these attack techniques appear to be hard to apply to obtain good cryptanalytic results in practice.

8 Conclusions

In this paper we have given a new methodology for differential-linear cryptanalysis under only the two assumptions implicitly used in the very first published paper on this technique. The new methodology is more reasonable and more general than Biham et al.'s methodology, and it can lead to some better differential-linear cryptanalytic results for some block ciphers than the previously known methodologies.

Using the new methodology, we have presented differential-linear attacks on 10-round CTC2 with a 255-bit block size and key, 13-round DES, and 12-round Serpent. In terms of the numbers of attacked rounds, the 10-round CTC2 attack is the first published cryptanalytic attack on the version of CTC2; the 13-round DES attack is much better than any previously published differential-linear cryptanalytic results for DES, though it is inferior to the best previously published cryptanalytic results for DES; and the 12-round Serpent attack matches the best previously published cryptanalytic result for Serpent (that was obtained using Biham et al.'s methodology). In addition, an important merit for these new differential-linear cryptanalytic results is that they are obtained under only two assumptions and thus are more reasonable than those obtained using Biham et al.'s methodology. Like most cryptanalytic results on block ciphers, most of these attacks are far less than practical at present, but they provide a comprehensive understanding of the security of the block ciphers.

The new methodology can be potentially used to cryptanalyse other block ciphers, and block cipher designers should pay attention to this new methodology when designing ciphers.

The new methodology still requires Assumptions 1 and 2. As a direction for future research on differential-linear cryptanalysis, it would be interesting to investigate how to further reduce the number of assumptions used, making a more reasonable and more general methodology that could be used in practice.

Acknowledgments

The author is very grateful to Orr Dunkelman and Nathan Keller for their discussions on the flaw about CTC2 and to the anonymous referees for their comments on earlier versions of the paper.

References

1. Anderson, R., Biham, E., Knudsen, L.R.: Serpent: a proposal for the Advanced Encryption Standard, NIST AES proposal (1998)
2. Biham, E.: New types of cryptanalytic attacks using related keys. Journal of Cryptology 7(4), 229–246 (1994)
3. Biham, E., Biryukov, A.: An improvement of Davies' attack on DES. Journal of Cryptology 10(3), 195–206 (1997)
4. Biham, E., Dunkelman, O., Keller, N.: The rectangle attack — rectangling the Serpent. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 340–357. Springer, Heidelberg (2001)
5. Biham, E., Dunkelman, O., Keller, N.: Linear cryptanalysis of reduced round Serpent. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 16–27. Springer, Heidelberg (2002)
6. Biham, E., Dunkelman, O., Keller, N.: Enhancing differential-linear cryptanalysis. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 254–266. Springer, Heidelberg (2002)
7. Biham, E., Dunkelman, O., Keller, N.: New results on boomerang and rectangle attacks. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 1–16. Springer, Heidelberg (2002)
8. Biham, E., Dunkelman, O., Keller, N.: Differential-linear cryptanalysis of Serpent. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 9–21. Springer, Heidelberg (2003)
9. Biham, E., Dunkelman, O., Keller, N.: New combined attacks on block ciphers. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 126–144. Springer, Heidelberg (2005)
10. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 2–21. Springer, Heidelberg (1990)
11. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. Journal of Cryptology 4(1), 3–72 (1991)
12. Biham, E., Shamir, A.: Differential cryptanalysis of the full 16-round DES. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 487–496. Springer, Heidelberg (1993)
13. Collard, B., Standaert, F.-X., Quisquater, J.-J.: Improved and multiple linear cryptanalysis of reduced round Serpent. In: Pei, D., et al. (eds.) Inscrypt 2007. LNCS, vol. 4990, pp. 51–65. Springer, Heidelberg (2008)
14. Courtois, N.T.: CTC2 and fast algebraic attacks on block ciphers revisited. IACR ePrint report 2007/152 (2007)
15. Courtois, N.T., Pieprzyk, J.: Cryptanalysis of block ciphers with overdefined systems of equations. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 267–287. Springer, Heidelberg (2002)
16. Daemen, J., Rijmen, V.: AES proposal: Rijndael. Proceedings of The First Advanced Encryption Standard Candidate Conference, NIST (1998)
17. Davies, D.: Investigation of a potential weakness in the DES algorithm. (1987)
18. Dunkelman, O.: Techniques for cryptanalysis of block ciphers. PhD thesis, Technion — Israel Institute of Technology, 2006.
19. Dunkelman, O., Keller, N.: Cryptanalysis of CTC2. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 226–239. Springer, Heidelberg (2009)
20. Dunkelman, O., Indesteege, S., Keller, N.: A differential-linear attack on 12-round Serpent. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 308–321. Springer, Heidelberg (2008)
21. GNU Project, `http://www.gnupg.org/oids.html`.
22. Handschuh, H., Naccache, D.: SHACAL. In: Proceedings of the First Open NESSIE Workshop (2000)
23. Hawkes, P.: Differential-linear weak key classes of IDEA. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 112–126. Springer, Heidelberg (1998)
24. Kelsey, J., Kohno, T., Schneier, B.: Amplified boomerang attacks against reduced-round MARS and Serpent. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 75–93. Springer, Heidelberg (2001)

25. Kim, J.: Combined differential, linear and related-key attacks on block ciphers and MAC algorithms. PhD thesis, Katholieke Universiteit Leuven, 2006.
26. Knudsen, L.R.: Cryptanalysis of LOKI91". In: Seberry, J., Zheng, Y. (eds.) ASIACRYPT 1992. LNCS, vol. 718, pp. 196–208. Springer, Heidelberg (1993)
27. Knudsen, L.R.: Trucated and higher order differentials. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 196–211. Springer, Heidelberg (1995)
28. Knudsen, L.R., Mathiassen, J.E.: A chosen-plaintext linear attack on DES. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 262–272. Springer, Heidelberg (2001)
29. Lai, X., Massey, J.L., Murphy, S: Markov ciphers and differential cryptanalysis. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 17–38. Springer, Heidelberg (1991)
30. Langford, S.K.: Differential-linear cryptanalysis and threshold signatures. Ph.D. thesis, Stanford University (1995).
31. Langford, S.K., Hellman, M.E.: Differential-linear cryptanalysis. In: Desmedt, Y. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 17–25. Springer, Heidelberg (1994)
32. Lu, J.: Cryptanalysis of block ciphers. PhD thesis, University of London, UK (2008)
33. Lu, J.: New methodologies for differential-linear cryptanalysis and its extensions. Cryptology ePrint Archive, Report 2010/025 (2010). http://eprint.iacr.org/2010/025
34. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
35. Matsui, M.: The first experimental cryptanalysis of the Data Encryption Standard. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 1–11. Springer, Heidelberg (1994)
36. Matsui, M., Yamagishi, A.: A new method for known plaintext attack of FEAL cipher. In: Rueppel, R.A. (ed.) EUROCRYPT 1992. LNCS, vol. 658, pp. 81–91. Springer, Heidelberg (1993)
37. National Bureau of Standards (NBS), Data Encryption Standard (DES), FIPS-46 (1977)
38. National Institute of Standards and Technology (NIST). Advanced Encryption Standard (AES), FIPS-197 (2001).
39. Selçuk, A.A.: On probability of success in linear and differential cryptanalysis. Journal of Cryptology 21(1), 131–147 (2008)
40. Wagner, D.: The boomerang attack. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 156–170. Springer, Heidelberg (1999)

New Observations on Impossible Differential Cryptanalysis of Reduced-Round Camellia*

Ya Liu[1], Leibo Li[2,3]**, Dawu Gu[1], Xiaoyun Wang[2,3,4], Zhiqiang Liu[1], Jiazhe Chen[2,3], Wei Li[5,6]

[1] Department of Computer Science and Engineering,
Shanghai Jiao Tong University, Shanghai 200240, China
{liuya0611,dwgu,ilu_zq}@sjtu.edu.cn
[2] Key Laboratory of Cryptologic Technology and Information Security,
Ministry of Education, Shandong University, Jinan 250100, China
[3] School of Mathematics, Shandong University, Jinan 250100, China
{lileibo, jiazhechen}@mail.sdu.edu.cn
[4] Institute for Advanced Study, Tsinghua University, Beijing 100084, China
xiaoyunwang@mail.tsinghua.edu.cn
[5] School of Computer Science and Technology, Donghua University, Shanghai 201620, China
[6] Shanghai Key Laboratory of Integrate Administration Technologies
for Information Security, Shanghai 200240, China
liwei.cs.cn@gmail.com

Abstract. Camellia is one of the widely used block ciphers, which has been selected as an international standard by ISO/IEC. In this paper, by studying the properties of the key-dependent transformations FL/FL^{-1}, we improve the previous results on impossible differential cryptanalysis of reduced-round Camellia and gain some new observations. First, we introduce some new 7-round impossible differentials of Camellia for weak keys. These weak keys that work for the impossible differential take 3/4 of the whole key space, therefore, we further get rid of the weak-key assumption and leverage the attacks on reduced-round Camellia to all keys by utilizing a method that is called the multiplied method. Second, we build a set of differentials which contains at least one 8-round impossible differential of Camellia with two FL/FL^{-1} layers. Following this new result, we show that the key-dependent transformations inserted in Camellia cannot resist impossible differential cryptanalysis effectively. Based on these 8-round impossible differential, we present a new cryptanalytic strategy to mount impossible differential attacks on reduced-round Camellia.

Key words: Block Cipher, Camellia, Impossible Differential Cryptanalysis

1 Introduction

The block cipher Camellia was proposed by NTT and Mitsubishi in 2000 [1]. It was selected as an e-government recommended cipher by CRYPTREC in 2002 [4] and the NESSIE block cipher portfolio in 2003 [19]. In 2005, it was adopted as the international standard by ISO/IEC [6]. Camellia is a 128-bit block cipher. It supports variable key sizes and the number of the rounds depends on the key size, i.e., 18 rounds for a 128-bit key size and 24 rounds for 192/256-bit key sizes. For simplicity, they can be usually denoted as Camellia-128, Camellia-192 and Camellia-256, respectively. Camellia adopts the basic Feistel structure with some key-dependent functions FL/FL^{-1} inserted every six rounds, where these key-dependent transformations must be linear and reversible for any fixed key. The goals for such a design are to provide non-regularity across rounds and to thwart further unknown attacks.

* The authors Ya Liu, Dawu Gu, Zhiqiang Liu and Wei Li are supported by the National Natural Science Foundation of China (No. 61073150 and No. 61003278), the Opening Project of Shanghai Key Laboratory of Integrate Administration Technologies for Information Security and the Fundamental Research Funds for the Central Universities. The authors Leibo Li, Xiaoyun Wang and Jiazhe Chen are supported by the National Natural Science Foundation of China (Grant No. 61133013 and No. 60931160442), and Tsinghua University Initiative Scientific Research Program (2009THZ01002).

** Corresponding author.

Up to now, many cryptanalytic methods were used to evaluate the security of reduced-round Camellia such as linear cryptanalysis, differential cryptanalysis, higher order differential attack, truncated differential attack, collision attack, square attack and impossible differential attack. Among them, most attacks focused on the security of simplified versions of Camellia, which did not take the FL/FL^{-1} and whitening layers into account [9–11, 15–18, 20–23], and only a few involved in the study of the original Camellia. For instance, Hatano *et al.* gave an higher order differential attack on the last 11 rounds of Camellia-256 [5], Chen *et al.* constructed a 6-round impossible differential with FL/FL^{-1} layer to attack 10-round Camellia-192 and 11-round Camellia-256 [3], Liu *et al.* attacked 11-round Camellia-192 and 12-round Camellia-256 by constructing a 7-round impossible differential [14]. Li *et al.* presented impossible differential attacks on 10-round Camellia-192 and 11-round Camellia-256 with a 7-round impossible differential including two FL/FL^{-1} layers [12].

Impossible differential cryptanalysis was independently introduced by Biham [2] and Knudsen [7], which is one of the most popular cryptanalytic tool. In order to mount an attack, the adversary tries to seek for an input difference that can never result in an output difference. The differential which connects the input and output difference is impossible and called an impossible differential. When the adversary wants to launch an impossible differential attack on a block cipher, she adds rounds before and/or after the impossible differential, and collect enough pairs with required plaintext and ciphertext differences. Then she concludes that the guessed subkey bits in added rounds must be wrong, if there is a pair meets the input and output values of the impossible differential under these subkey bits. In this way, she discards as many wrong keys as possible and exhaustively searches the rest of the keys.

In this paper, we reevaluate the security of reduced-round Camellia with FL/FL^{-1} and whitening layers against impossible differential cryptanalysis from two aspects. On the one hand, we first construct some new 7-round impossible differentials of Camellia for weak keys, which work for 75% of the keys. Based on them, we mount an impossible differential attack on Camellia in the weak-key setting. Then we further propose a multiplied method to extend our attacks for the whole key space. The basic idea is that if the correct key belongs to the set of weak keys, then it will never satisfy the impossible differential. While if the correct key is not a weak key, we get 2-bit conditions about the key. Specifically, for the whole key space, we present an attack on 10-round Camellia-128 with about $2^{113.8}$ chosen plaintexts and 2^{120} 10-round encryptions, 11-round Camellia-192 with about $2^{114.64}$ chosen plaintexts and 2^{184} 11-round encryptions as well as 12-round Camellia-256 with about $2^{116.17}$ chosen plaintexts or chosen ciphertexts and 2^{240} 12-round encryptions, respectively. Meanwhile, we can also extend the attacks to 12-round Camellia-192 and 14-round Camellia-256 with two FL/FL^{-1} layers. On the other hand, by studying some properties of key-dependent functions FL/FL^{-1}, we build a set of differentials which contains at least one 8-round impossible differential of Camellia with two FL/FL^{-1} layers. The length of these impossible differentials with two FL/FL^{-1} layers is the same as the length of the longest known impossible differential of Camellia without FL/FL^{-1} layers given by Wu and Zhang [23]. Consequently, we show that the key-dependent transformations inserted in Camellia cannot resist impossible differential cryptanalysis effectively. On the basis of this differential set, we propose a new cryptanalytic strategy to attack 11-round Camellia-128 with 2^{122} chosen plaintexts and 2^{122} 11-round encryptions, 12-round Camellia-192 with 2^{123} chosen plaintexts and $2^{187.2}$ 12-round encryptions as well as 13-round Camellia-256 with 2^{123} chosen plaintexts and $2^{251.1}$ 13-round encryptions (not from the first round but with the whitening layers), respectively. In table 1, we summarize our results along with the former known ones on reduced-round Camellia.

The remainder of this paper is organized as follows. Section 2 gives some notations and a brief introduction of Camellia. Section 3 first presents 7-round impossible differentials of Camellia

Table 1. Summary of the attacks on Reduced-Round Camellia

Key Size	Rounds	Attack Type	Data	Time(Enc)	Memory (Bytes)	Source
Camellia-128	9†	Square	2^{48}CP	2^{122}	2^{53}	[10]
	10†	Impossible DC	2^{118}CP	2^{118}	2^{93}	[17]
	10†	Impossible DC	$2^{118.5}$CP	$2^{123.5}$	2^{127}	[12]
	10(Weak Key)	Impossible DC	$2^{111.8}$CP	$2^{111.8}$	$2^{84.8}$	Section 3.2
	10	Impossible DC	$2^{113.8}$CP	2^{120}	$2^{84.8}$	Section 3.2
	11	Impossible DC	2^{122}CP	2^{122}	2^{102}	Section 4.4
Camellia-192	10	Impossible DC	2^{121}CP	$2^{175.3}$	$2^{155.2}$	[3]
	10	Impossible DC	$2^{118.7}$CP	$2^{130.4}$	2^{135}	[12]
	11†	Impossible DC	2^{118}CP	$2^{163.1}$	2^{141}	[17]
	11(Weak Key)	Impossible DC	$2^{112.64}$CP	$2^{146.54}$	$2^{141.64}$	Section 3.3
	11	Impossible DC	$2^{114.64}$CP	2^{184}	$2^{141.64}$	Section 3.3
	12	Impossible DC	2^{123}CP	$2^{187.2}$	2^{160}	Section 4.3
	12†	Impossible DC	$2^{120.1}$CP	2^{184}	$2^{124.1}$	Section 3.5
Camellia-256	last 11 rounds	High Order DC	2^{93}CP	$2^{255.6}$	2^{98}	[5]
	11	Impossible DC	2^{121}CP	$2^{206.8}$	2^{166}	[3]
	11	Impossible DC	$2^{119.6}$CP	$2^{194.5}$	2^{135}	[12]
	12(Weak Key)	Impossible DC	$2^{121.12}$CP	$2^{202.55}$	$2^{142.12}$	Section 3.4
	12	Impossible DC	$2^{116.17}$CP/CC	2^{240}	$2^{150.17}$	Section 3.4
	13	Impossible DC	2^{123}CP	$2^{251.1}$	2^{208}	Section 4.2
	14†	Impossible DC	2^{120}CC	$2^{250.5}$	2^{125}	Section 3.5

DC: Differential Cryptanalysis; CP/CC: Chosen Plaintexts/Chosen Ciphertexts;
Enc: Encryptions; †: The attack doesn't include the whitening layers.

for weak keys. Based on them, impossible differential attacks on 10-round Camellia-128, 11-round Camellia-192 and 12-round Camellia-256 are elaborated. Section 4 first constructs a set of differentials which contains at least one 8-round impossible differential of Camellia with two FL/FL^{-1} layers, and then proposes impossible differential attacks on 11-round Camellia-128, 12-round Camellia-192 and 13-round Camellia-256, respectively. Section 5 summarizes this paper.

2 Preliminaries

2.1 Some Notations

- P, C: the plaintext and the ciphertext;
- L_{i-1}, R_{i-1}: the left half and the right half of the i-th round input;
- $\Delta L_{i-1}, \Delta R_{i-1}$: the left half and the right half of the input difference in the i-th round;
- $X \mid Y$: the concatenation of X and Y;
- $kw_1|kw_2, kw_3|kw_4$: the pre-whitening key and the post-whitening key;
- k_i: the subkey used in the i-th round;
- $kl_i (1 \leq i \leq 6)$: 64-bit keys used in the FL/FL^{-1} layers;
- $S_r, \Delta S_r$: the output and the output difference of the S-boxes in the r-th round;
- $X \lll j$: left rotation of X by j bits;
- $X_{L(\frac{n}{2})}, X_{R(\frac{n}{2})}$: the left half and the right half of a n-bit word X;
- $X_i, X_{\{i,j\}}, X_{\{i \sim j\}}$: the i-th byte, the i-th and j-th bytes and the i-th to the j-th bytes of X;
- $X^i, X^{(i,j)}, X^{(i \sim j)}$: the i-th bit, the i-th and j-th bits and the i-th to j-th bits of X;
- $\oplus, \cap, \cup$: bitwise exclusive-OR (XOR), AND, and OR operations, respectively;
- $0_{(i)}, 1_{(i)}$: consecutive i bits are zero or one.

2.2 Overview of Camellia

Camellia [1] is a 128-bit block cipher. Two keyed functions FL/FL^{-1} are inserted every 6 rounds. Camellia uses variable key sizes and the number of rounds depends on the key size, i.e., 18 rounds for a 128-bit key size and 24 rounds for 192/256-bit key sizes. The round function of Camellia uses a SPN structure. Among it, the linear transformation P and its inverse function P^{-1} are defined as follows.

$$P : (\{0,1\}^8)^8 \to (\{0,1\}^8)^8, y_1 \mid y_2 \mid y_3 \mid y_4 \mid y_5 \mid y_6 \mid y_7 \mid y_8 \to z_1 \mid z_2 \mid z_3 \mid z_4 \mid z_5 \mid z_6 \mid z_7 \mid z_8;$$

$$\begin{aligned}
z_1 &= y_1 \oplus y_3 \oplus y_4 \oplus y_6 \oplus y_7 \oplus y_8; & y_1 &= z_2 \oplus z_3 \oplus z_4 \oplus z_6 \oplus z_7 \oplus z_8;\\
z_2 &= y_1 \oplus y_2 \oplus y_4 \oplus y_5 \oplus y_7 \oplus y_8; & y_2 &= z_1 \oplus z_3 \oplus z_4 \oplus z_5 \oplus z_7 \oplus z_8;\\
z_3 &= y_1 \oplus y_2 \oplus y_3 \oplus y_5 \oplus y_6 \oplus y_8; & y_3 &= z_1 \oplus z_2 \oplus z_4 \oplus z_5 \oplus z_6 \oplus z_8;\\
z_4 &= y_2 \oplus y_3 \oplus y_4 \oplus y_5 \oplus y_6 \oplus y_7; & y_4 &= z_1 \oplus z_2 \oplus z_3 \oplus z_5 \oplus z_6 \oplus z_7;\\
z_5 &= y_1 \oplus y_2 \oplus y_6 \oplus y_7 \oplus y_8; & y_5 &= z_1 \oplus z_2 \oplus z_5 \oplus z_7 \oplus z_8;\\
z_6 &= y_2 \oplus y_3 \oplus y_5 \oplus y_7 \oplus y_8; & y_6 &= z_2 \oplus z_3 \oplus z_5 \oplus z_6 \oplus z_8;\\
z_7 &= y_3 \oplus y_4 \oplus y_5 \oplus y_6 \oplus y_8; & y_7 &= z_3 \oplus z_4 \oplus z_5 \oplus z_6 \oplus z_7;\\
z_8 &= y_1 \oplus y_4 \oplus y_5 \oplus y_6 \oplus y_7; & y_8 &= z_1 \oplus z_4 \oplus z_6 \oplus z_7 \oplus z_8;
\end{aligned}$$

The key-dependent function $FL : \{0,1\}^{64} \times \{0,1\}^{64}$ maps $(X_L \mid X_R, kl_L \mid kl_R) \mapsto Y_L \mid Y_R$, where $Y_R = ((X_L \cap kl_L) \lll 1) \oplus X_R, Y_L = (Y_R \cup kl_R) \oplus X_L$.

Key Schedule of Camellia The key schedule algorithm of Camellia applies a 6-round Feistel structure to generate two 128-bit intermediate variables K_A and K_B. These two variables K_A and K_B can be calculated by two 128-bit variables K_L and K_R defined by the main key K. For Camellia-128, the 128-bit key K is used as K_L and K_R is 0. For Camellia-192, the left 128-bit of the key K is used as K_L, and the concatenation of the right 64-bit of the key K and the complement of the right 64-bit of the key K is used as K_R. For Camellia-256, the main key K is separated into two 128-bit variables K_L and K_R, i.e., $K = K_L \mid K_R$.

3 7-Round Impossible Differentials of Camellia for Weak Keys and Their Applications [1]

In this section, we first construct some 7-round impossible differentials of Camellia in weak-key setting. Based on them, we present impossible differential attacks on 10-round Camellia-128, 11-round Camellia-192 and 12-round Camellia-256 which start from the first round. In addition, we can also extend the attack to 12-round Camellia-192 and 14-round Camellia-256 with two FL/FL^{-1} layers.

3.1 7-Round Impossible Differentials of Camellia for Weak Keys

This section introduces 7-round impossible differentials of Camellia in weak-key setting, which is based on the following propositions.

Lemma 1 ([8]). *Let X, X', K be l-bit values, and $\Delta X = X \oplus X'$, then the differential properties of AND and OR operations are:*
$(X \cap K) \oplus (X' \cap K) = (X \oplus X') \cap K = \Delta X \cap K,$
$(X \cup K) \oplus (X' \cup K) = (X \oplus K \oplus (X \cap K)) \oplus (X' \oplus K \oplus (X' \cap K)) = \Delta X \oplus (\Delta X \cap K).$

Lemma 2 ([3]). *Let ΔX and ΔY be the input and output differences of FL. Then*

$$\Delta Y_R = ((\Delta X_L \cap kl_L) \lll 1) \oplus \Delta X_R, \qquad \Delta Y_L = \Delta X_L \oplus \Delta Y_R \oplus (\Delta Y_R \cap kl_R);$$
$$\Delta X_L = \Delta Y_L \oplus \Delta Y_R \oplus (\Delta Y_R \cap kl_R), \qquad \Delta X_R = ((\Delta X_L \cap kl_L) \lll 1) \oplus \Delta Y_R.$$

[1] By Leibo Li, Xiaoyun Wang and Jiazhe Chen. See [13] for more details.

Proposition 1. *If the output difference of FL function is $\Delta Y = (0|0|0|0|d|0|0|0)$, where $d \neq 0$ and $d^{(1)} = 0$, then the input difference of FL function should satisfy $\Delta X_{\{2,3,4,6,7,8\}} = 0$.*

Proposition 2. *If the output difference of FL^{-1} function is $\Delta X = (0|e|e|e|0|e|e|e)$, and the subkeys of FL^{-1} function satisfy that $KL_L^{(9)}$ is 0 or $KL_R^{(8)}$ is 1, then the first byte of input difference ΔY should be zero, where e is a non-zero byte.*

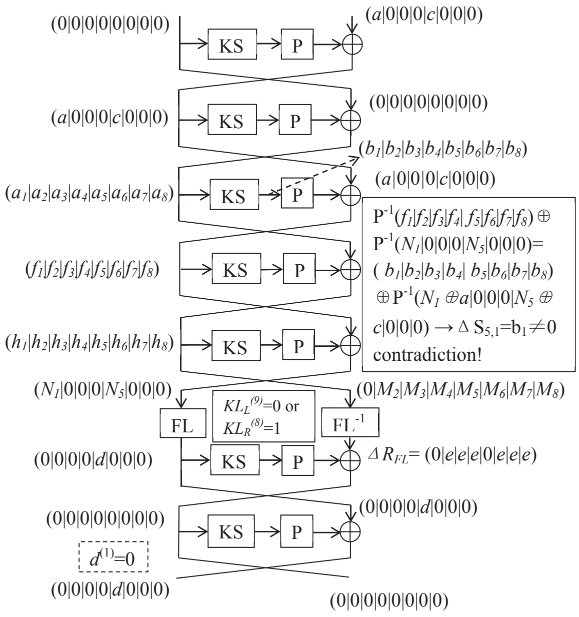


Fig. 1. A 7-Round Impossible Differential for Weak Keys

Proposition 3. *Given a 7-round Camellia encryption and a FL/FL^{-1} layer inserted between the fifth and sixth round. If the input difference of the first round is $(0|0|0|0|0|0|0|0, a|0|0|0|c|0|0|0)$, and the subkeys of FL^{-1} function satisfy $KL_L^{(9)} = 0$ or $KL_R^{(8)} = 1$, then the output difference $(0|0|0|0|d|0|0|0, 0|0|0|0|0|0|0|0)$ with $d^{(1)} = 0$ is impossible, where a and d are non-zero bytes, c is an arbitrary value (see Fig. 1).*

Proof. First, we analyze the forward direction. It is trivial that $(\Delta L_1, \Delta R_1) = (a|0|0|0|c|0|0|0, 0|0|0|0|0|0|0|0)$, then it propagates to $(\Delta L_2, \Delta R_2) = (a_1|a_2|a_3|a_4|a_5|a_6|a_7|a_8, a|0|0|0|c|0|0|0)$ after the second round, where a_1 and a_5 are non-zero values, a_i $(i = 2,3,4,6,7,8)$ are unknown values. Getting through the key addition and substitution layers of the third round, the output difference of S-box layer in the third round is $\Delta S_3 = (b_1|b_2|b_3|b_4|b_5|b_6|b_7|b_8)$, where b_1 and b_5 are non-zero values. Then we have $(\Delta L_3, \Delta R_3) = (f_1|f_2|f_3|f_4|f_5|f_6|f_7|f_8, a_1|a_2|a_3|a_4|a_5|a_6|a_7|a_8)$, and $(\Delta L_4, \Delta R_4) = (h_1|h_2|h_3|h_4|h_5|h_6|h_7|h_8, f_1|f_2|f_3|f_4\ |f_5|f_6|f_7|f_8)$, where f_i, h_i are unknown values.

Second, we consider the backward direction. The output difference of the seventh round is $(\Delta L_7, \Delta R_7) = (0|0|0|0|d|0|0|0, 0|0|0|0|0|0|0|0)$, then the output difference of the sixth round is $(\Delta L_6, \Delta R_6) = (0|0|0|0|0|0|0|0, 0|0|0|0|d|0|0|0)$, and the output difference of FL/FL^{-1} layer is $(0|0|0|0|d|0|0|0, 0|e|e|e|0|e|e|e)$. According to the condition $d^{(1)} = 0$ and Proposition 1, we obtain that the input difference of FL function is $(N_1|0|0|0|N_5|0|0|0)$. Since $KL_L^{(9)} = 0$ or $KL_R^{(8)} = 1$, in the light of Proposition 2, the input difference of FL^{-1} function is $(0|M_2|M_3|M_4|M_5|M_6|M_7|M_8)$, which means $\varDelta L_{4,1} = h_1 = 0$. Where N_1, N_5 and M_i $(i = 2, ..., 8)$ are unknown bytes.

Finally, we focus on the fifth round. The output difference of S-layer in the fifth round is

$$\begin{aligned}\varDelta S_5 &= P^{-1}(f_1|f_2|f_3|f_4|f_5|f_6|f_7|f_8) \oplus P^{-1}(N_1|0|0|0|N_5|0|0|0)\\ &= (b_1|b_2|b_3|b_4|b_5|b_6|b_7|b_8) \oplus P^{-1}(N_1 \oplus a|0|0|0|N_5 \oplus c|0|0|0).\end{aligned}$$

Then $\varDelta S_{5,1} = b_1 \neq 0$, which contradicts $\varDelta L_{4,1} = 0$. □

We also obtain three other impossible differentials under different weak-key assumptions:

- $(0|0|0|0|0|0|0|0, 0|a|0|0|0|c|0|0) \nrightarrow (0|0|0|0|0|d|0|0, 0|0|0|0|0|0|0|0)$ with conditions $KL_L^{(17)} = 0$ or $KL_R^{(16)} = 1$, and $d^{(1)} = 0$,
- $(0|0|0|0|0|0|0|0, 0|0|a|0|0|0|c|0) \nrightarrow (0|0|0|0|0|0|d|0, 0|0|0|0|0|0|0|0)$ with conditions $KL_L^{(25)} = 0$ or $KL_R^{(24)} = 1$, and $d^{(1)} = 0$,
- $(0|0|0|0|0|0|0|0, 0|0|0|a|0|0|0|c) \nrightarrow (0|0|0|0|0|0|0|d, 0|0|0|0|0|0|0|0)$ with conditions $KL_L^{(1)} = 0$ or $KL_R^{(32)} = 1$, and $d^{(1)} = 0$.

We denote this type of impossible differentials above as **5+2 WKID** (weak-key impossible differentials). Due to the feature of Feistel structure, we also deduce another type of 7-round impossible differentials with the FL/FL^{-1} layers inserted between the second and the third rounds. We call them **2+5 WKID**, which are depicted as follows.

- $(0|0|0|0|0|0|0|0, 0|0|0|0|d|0|0|0) \nrightarrow (a|0|0|0|c|0|0|0, 0|0|0|0|0|0|0|0)$ with conditions $KL'^{(9)}_L = 0$ or $KL'^{(8)}_R = 1$, and $d^{(1)} = 0$,
- $(0|0|0|0|0|0|0|0, 0|0|0|0|0|d|0|0) \nrightarrow (0|a|0|0|0|c|0|0, 0|0|0|0|0|0|0|0)$ with conditions $KL'^{(17)}_L = 0$ or $KL'^{(16)}_R = 1$, and $d^{(1)} = 0$,
- $(0|0|0|0|0|0|0|0, 0|0|0|0|0|0|d|0) \nrightarrow (0|0|a|0|0|0|c|0, 0|0|0|0|0|0|0|0)$ with conditions $KL'^{(25)}_L = 0$ or $KL'^{(24)}_R = 1$, and $d^{(1)} = 0$,
- $(0|0|0|0|0|0|0|0, 0|0|0|0|0|0|0|d) \nrightarrow (0|0|0|a|0|0|0|c, 0|0|0|0|0|0|0|0)$ with conditions $KL'^{(1)}_L = 0$ or $KL'^{(32)}_R = 1$, and $d^{(1)} = 0$,

where KL' represents the subkey used in FL-function.

3.2 Impossible Differential Attack on 10-Round Camellia-128

We first propose an attack that works for $3 \times 2^{126} (= \frac{3}{4} \times 2^{128})$ keys, which is mounted by adding one round on the top and two rounds on the bottom of the **5+2 WKID** (See Fig. 2). The attack procedure is as follows.

Data Collection.

1. Choose 2^n structures of plaintexts, and each structure contains 2^{32} plaintexts

$$(L_0, R_0) = (\alpha_1|x_1|x_2|x_3|\alpha_2|x_4|x_5|x_6, P(\beta_1|y_1|y_2|y_3|\beta_2|y_4|y_5|y_6)),$$

where x_i and y_i $(i = 1, ..., 6)$ are fixed values in each structure, while α_j, β_j $(j = 1, 2)$ takes all the possible values, and P is the linear transformation.

2. For each structure, ask for the encryption of the plaintexts and get 2^{32} ciphertexts. Store them in a hash table H indexed by $C_{L,\{1,5\}}$, the XOR of $C_{L,2}$ and $C_{L,3}$, the XOR of $C_{L,2}$ and $C_{L,4}$, the XOR of $C_{L,2}$ and $C_{L,6}$, the XOR of $C_{L,2}$ and $C_{L,7}$, the XOR of $C_{L,2}$ and

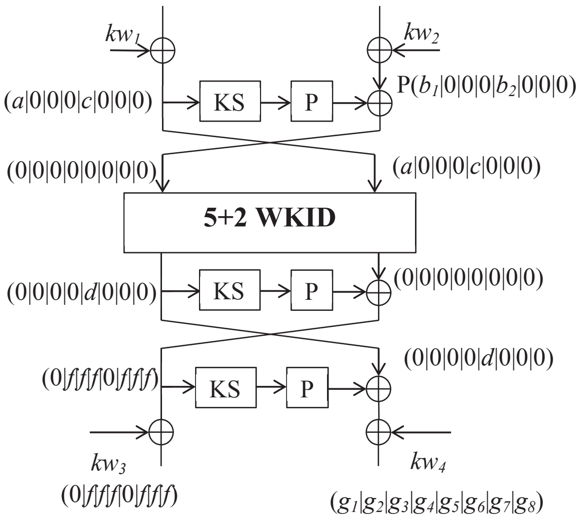


Fig. 2. Impossible Differential Attack on 10-Round Camellia-128 for Weak Keys

$C_{L,8}$. Then by birthday paradox, we get $2^{n+63} \times 2^{-56} = 2^{n+7}$ pairs of ciphertexts with the differences

$$(\Delta C_L, \Delta C_R) = (0|f|f|f|0|f|f|f, g_1|g_2|g_3|g_4|g_5|g_6|g_7|g_8),$$

and the differences of corresponding plaintext pairs satisfy

$$(\Delta L_0, \Delta R_0) = (a|0|0|0|c|0|0|0, P(b_1|0|0|0|b_2|0|0|0)),$$

where a, c, f and b_i $(i = 1, 2)$ are non-zero bytes, and g_i are unknown bytes. For every pair, compute the value

$$P^{-1}(\Delta C_R) = P^{-1}(g_1|g_2|g_3|g_4|g_5|g_6|g_7|g_8) = (g'_1|g'_2|g'_3|g'_4|g'_5|g'_6|g'_7|g'_8).$$

Keep only the pairs whose ciphertexts satisfy $g'_1 = 0$. The probability of this event is 2^{-8}, thus the expected number of remaining pairs is $2^{n+7} \times 2^{-8} = 2^{n-1}$.

Key Recovery.

1. For each pair obtained in the data collection phase, guess the 16-bit value $K_{1,\{1,5\}}$, partially encrypt its plaintext $(L_{0,\{1,5\}}, L'_{0,\{1,5\}})$ to get the intermediate value $(S_{1,\{1,5\}}, S'_{1,\{1,5\}})$ and the difference $\Delta S_{1,\{1,5\}}$. Then discard the pairs whose intermediate values do not satisfy $\Delta S_{1,1} = b_1$ and $\Delta S_{1,5} = b_2$. The probability of a pair being kept is 2^{-16}, so the expected number of remaining pairs is $2^{n-1} \times 2^{-16} = 2^{n-17}$.
2. In this step, the ciphertext of every remaining pair is considered.
   (a) Guess the 8-bit value $K_{10,8}$ for every remaining pair, partially decrypt the ciphertext $(C_{L,8}, C'_{L,8})$ to get the intermediate value $(S_{10,8}, S'_{10,8})$ and the difference $\Delta S_{10,8}$, and discard the pairs whose intermediate values do not satisfy $\Delta S_{10,8} = g'_8$. The expected number of remaining pairs is $2^{n-17} \times 2^{-8} = 2^{n-25}$.
   (b) For $l = 2, 3, 4, 6, 7$, guess the 8-bit value $K_{10,l}$. For every remaining pair, partially decrypt the ciphertext $(C_{L,l}, C'_{L,l})$ to get the intermediate value $(S_{10,l}, S'_{10,l})$ and the difference $\Delta S_{10,l}$, and keep only the pairs whose intermediate values satisfy $\Delta S_{10,l} = g'_l \oplus g'_5$. Since for each l, each pair will remain with probability 2^{-8}, the expected number of remaining pairs is $2^{n-25} \times 2^{5\times(-8)} = 2^{n-65}$.

(c) Guess the 8-bit value $K_{10,1}$, partially decrypt the ciphertext $C_{L,1}$ of every remaining pair to get the intermediate value $S_{10,1}$, which is also the value of $S'_{10,1}$.

(d) Partially decrypt (S_{10}, S'_{10}) to get the intermediate values $(R_{9,5}, R'_{9,5})$, and discard the pairs whose intermediate values do not satisfy $\Delta R_{9,5}^{(1)} = 0$. As the probability of a pair being discarded is 0.5, the expected number of remaining pairs is $2^{n-65} \times 2^{-1} = 2^{n-66}$.

3. For every remaining pair, guess the 8-bit value $K_{9,5}$, partially decrypt the output value $(R_{9,5}, R'_{9,5})$ to get the intermediate value $(S_{9,5}, S'_{9,5})$ and the difference $\Delta S_{9,5}$. If there is a pair satisfies $\Delta S_{9,5} = \Delta C_{L,2}$, we discard the guessed key and try another one. Otherwise we exhaustively search for the remaining 48 bits of the key under this guessed key, if the correct key is obtained, we halt the attack; otherwise, another key guess should be tried.

Complexity. Since the probability of the event $\Delta S_{9,5} = \Delta C_{L,2}$ happens in step 3 of key recovery phase is 2^{-8}, the expected number of remaining guesses for 72-bit target subkeys is about $\epsilon = 2^{80} \times (1 - 2^{-8})^{2^{n-66}}$. If we choose $\epsilon = 1$, then n is 79.8, and the proposed attack requires $2^{n+32} = 2^{111.8}$ chosen plaintexts. The time and memory complexities are dominated by step 2 of data collection phase, which are about $2^{111.8}$ 10-round encryptions and $2^{n-1} \times 4 \times 2^4 = 2^{84.8}$ bytes.

Extending the Attack to the Whole Key Space. On the basis of the above impossible differential attack for weak keys, we construct a multiplied attack on 10-Round Camellia-128.

- **Phase 1.** Perform an impossible differential attack by using the **5+2 WKID**

$$(0|0|0|0|0|0|0|0, a|0|0|0|c|0|0|0) \nrightarrow (0|0|0|0|d|0|0|0, 0|0|0|0|0|0|0|0).$$

  This phase is extremely similar to the weak-key attack that is described above. However, it is slightly different when the attack is finished. That is, if there is a key kept, then the key is the correct key, and we halt the procedure of the attack. Otherwise, we conclude that the correct key does not belong to this set of weak keys, which means that $kl_1^{(9)} = 1$ and $kl_2^{(8)} = 0$. In this case, we get 2-bit information of the key and perform the next phase.
- **Phases 2 to 4.** Perform an impossible differential attack by using each **5+2 WKID** in the following:

$$(0|0|0|0|0|0|0|0, 0|a|0|0|0|c|0|0) \nrightarrow (0|0|0|0|0|d|0|0, 0|0|0|0|0|0|0|0),$$
$$(0|0|0|0|0|0|0|0, 0|0|a|0|0|0|c|0) \nrightarrow (0|0|0|0|0|0|d|0, 0|0|0|0|0|0|0|0),$$
$$(0|0|0|0|0|0|0|0, 0|0|0|a|0|0|0|c) \nrightarrow (0|0|0|0|0|0|0|d, 0|0|0|0|0|0|0|0).$$

  The procedure is similar to Phase 1, and either recover the correct key or get another 2-bit information about the key and execute the next phase.
- **Phase 5.** Announce the intermediate key

$$K_A^{(95,103,111,119)} = 0 \text{ and } K_A^{(6,14,22,30)} = 1,$$

  then exhaustively search for the remaining 120 bit value of K_A and recover the key K_L.

The upper bound of the time complexity is $2^{111.8} \times 4 + 2^{120} \approx 2^{120}$. The data complexity is about $2^{113.8}$. The memory could be reused in different phase, so the memory requirement is about $2^{84.8}$ bytes.

3.3 Attack on 11-Round Camellia-192

We add one round on the bottom of 10-round attack and give an attack on 11-round Camellia-192.

Data Collection. Choose $2^{80.64}$ structures of plaintexts. Each structure contains 2^{32} plaintexts:

$$(L_0, R_0) = (\alpha_1|x_1|x_2|x_3|\alpha_2|x_4|x_5|x_6, P(\beta_1|y_1|y_2|y_3|\beta_2|y_4|y_5|y_6)),$$

where x_i and y_i $(i = 1, ..., 6)$ are fixed values in each structure, while α_j and β_j $(j = 1, 2)$ take all the possible values, and P is the linear transformation. Ask for the encryption of the corresponding ciphertext for each plaintext, compute $P^{-1}(C_L)$ and store the plaintext-ciphertext pairs (L_0, R_0, C_L, C_R) in a hash table indexed by 8-bit value $(P^{-1}(C_L))_1$. By birthday paradox, we get $2^{143.64} \times 2^{-8} = 2^{135.64}$ pairs whose ciphertext differences satisfy $P^{-1}(\Delta C_R) = (h'_1|h'_2|h'_3|h'_4|h'_5|h'_6|h'_7|h'_8)$ and $P^{-1}(\Delta C_L) = (0|g'_2|g'_3|g'_4|g'_5|g'_6|g'_7|g'_8)$, where h'_i and g'_i are unknown values.

Key Recovery.

1. For $l = 1, 5$, guess the 8-bit value of $K_{1,l}$, partially encrypt their plaintext $(L_{0,l}, L'_{0,l})$ and discard the pairs whose intermediate value do not satisfy $\Delta S_{1,l} = (P^{-1}(\Delta R_0))_l$. The expected number of remaining pairs is $2^{135.64} \times 2^{-16} = 2^{119.64}$.
2. In this step, we consider the ciphertext of each remaining pair.
   (a) For $l = 1, 2, 3, 4, 6, 7, 8$, guess the 8-bit value of $K_{11,l}$. Partially decrypt the ciphertext $(C_{L,l}, C'_{L,l})$ and keep only the pairs which satisfy $\Delta S_{11,l} = h'_l$. The expected number of remaining pairs is $2^{119.64} \times 2^{7\times(-8)} = 2^{63.64}$.
   (b) Guess the 8-bit value $K_{11,5}$. Partially decrypt the ciphertext $(C_{L,5}, C'_{L,5})$, then compute the intermediate value (R_{10}, R'_{10}), where $\Delta R_{10} = (0|f|f|f|0|f|f|f)$ and $f = \Delta S_{11,5} \oplus h'_5$.
3. Application of the 10-round attack.
   (a) Guess the 8-bit value $K_{10,8}$, partially decrypt $(R_{10,8}, R'_{10,8})$ and discard the pairs whose intermediate values do not satisfy $\Delta S_{10,8} = g'_8$. The expected number of remaining pairs is $2^{63.64} \times 2^{-8} = 2^{55.64}$.
   (b) For $l = 2, 3, 4, 6, 7$, guess the 8-bit value $K_{10,l}$. Partially decrypt the intermediate value $(R_{10,l}, R'_{10,l})$ and keep only the pairs whose intermediate values satisfy $\Delta S_{10,l} = g'_l \oplus g'_5$. The expected number of remaining pairs is $2^{55.64} \times 2^{5\times(-8)} = 2^{15.64}$.
   (c) Guess the 8-bit value $K_{10,1}$, partially decrypt the intermediate value $R_{10,1}$ and calculate the intermediate values $(R_{9,5}, R'_{9,5})$. Discard the pairs whose intermediate values do not satisfy $\Delta R^{(1)}_{9,5} = 0$. Then the expected number of remaining pairs is $2^{15.64} \times 2^{-1} = 2^{14.64}$.
   (d) Guess the 8-bit value $K_{9,5}$, partially decrypt the intermediate value $(R_{9,5}, R'_{9,5})$ to get the difference $\Delta S_{9,5}$. If there is a pair satisfies $\Delta S_{9,5} = \Delta R_{10,2}$, we discard the guessed key and try another one. Otherwise we exhaustively search for the rest 48 bits of K_L and K_R under this key, if the correct key is obtained, we halt the attack; otherwise, another key should be tried.

Complexity. The data complexity of the attack is $2^{112.64}$ chosen plaintexts. The time complexity is dominated by step 3 (d) which requires about $2^{144} \times (1 + (1 - 2^{-8}) + (1 - 2^{-8})^2 + ... + (1 - 2^{-8})^{2^{13.7}-1}) \times 2 \times \frac{1}{11} \times \frac{1}{8} \approx 2^{146.54}$ 11-round encryptions. The memory complexity is about $2^{133.56} \times 4 \times 2^4 = 2^{141.64}$ bytes.

Reduce the Time Complexity to $2^{138.54}$. Assume 16-bit value α_2 and β_2 are fixed in data collection phase of above attack, then we can collect $2^{n+31} \times 2^{-8} = 2^{n+23}$ pairs, where n represents the number of structures. Nevertheless, it is unnecessary for us to guess 8-bit subkey $K_{1,5}$ in this case. Then there are totally 136-bit values of subkey to be guessed in the attack, therefore, the expected number of remaining guesses of target subkey is about $\epsilon = 2^{136} \times (1 - 2^{-8})^{2^{n-90}}$ after the attack. If we chose $\epsilon = 1$, n is 104.56. Then the data complexity increases to $2^{n+16} = 2^{120.56}$, but the time complexity reduces to $2^{138.54}$, the memory requirement reduces to $2^{133.56}$ bytes.

Extending the Attack to the Whole Key Space. Similar to 10-round attack on Camellia-128, we mount a multiplied attack on Camellia-192 for the whole key space. The expected time of the attack is about $4 \times 2^{146.54} + 2^{192} \times (1 - \frac{3}{4})^4 = 2^{184}$. The expected data of the attack is $2^{114.64}$. The memory requirement is about $2^{141.64}$ bytes.

3.4 The Attack on 12-Round Camellia-256

We add one round on the bottom of 11-round attack, and present a 12-round attack on Camellia-256. The attack procedure is similar to the 11-round attack. First choose $2^{81.17}$ structures and collect $2^{144.17}$ plaintext-ciphertext pairs in data collection phase. After guessing the subkey $K_{1,\{1,5\}}$, we guess the 64-bit value K_{12} and compute the intermediate value (R_{11}, R'_{11}), then apply the 11-round attack to perform the remaining steps. In summary, the proposed attack requires $2^{81.17+32} = 2^{113.17}$ chosen plaintexts. The time complexity is about $2^{210.55}$ 12-round encryptions, and the memory requirement is about $2^{150.17}$ bytes. Similar to the above subsection, the time complexity and memory requirement can also reduce to $2^{202.55}$ and $2^{142.12}$, respectively, but data complexity increases to $2^{121.12}$ in this case.

We also construct another type of impossible differential attack of Camellia-256, which adds four rounds on the top and one round on the bottom of the **2+5 WKID** (see section 3.1). The attack is performed under the chosen ciphertext attack scenario. Similar to the attack based on the **5+2 WKID**, the data and time complexity are about $2^{113.17}$ and $2^{216.3}$, respectively.

Extending the Attack to the Whole Key Space. On the basis of two types of impossible differential attacks for weak keys, we mount a multiplied attack on 12-round Camellia-256 for the whole key space as below.

- **Phases 1 to 8.** Preform an impossible differential attack by using of all conditional impossible differentials **2+5 WKID** list in section 3.1. For each phase, if success, output the actual key, else perform the next phase.
- **Phase 9.** Announce 16-bit value of the master key

  $$K_R^{(31,39,47,55,95,103,111,119)} = 0 \text{ and } K_R^{(6,14,22,30,70,78,86,94)} = 1,$$

  then exhaustively search for the remaining 240 bit value of K_R, K_L and recover the actual key.

The expected time of the attack is $2^{216.3} \times 8 + 2^{256} \times (\frac{1}{4})^8 \approx 2^{240}$ encryptions, and the expected data complexity is about $2^{116.17}$.

3.5 The Attacks Including Two FL/FL^{-1} Layers

If we do not start from the first round, we can take the attacks that include two FL/FL^{-1} layers into account. We first illustrate some new observations of FL and FL^{-1} functions, then present attacks on variants of 14-round Camellia-256 and 12-round Camellia-192.

Proposition 4. *If the output difference of FL function is $\Delta Y = (a|0|0|0|0|0|0|0)$, then the input difference should satisfy $\Delta X = (b_1|0|0|0|b_5|0|0|b_8)$ with $b_1 = a$, $b_5^{(8)} = 0$ and $b_8^{(1\sim7)} = 0$, where a is a non-zero byte.*

Proposition 5. *If the output difference of FL^{-1} function is $\Delta X = (a|a|a|0|a|0|0|a)$, and the input difference $\Delta Y = (b_1|b_2|b_3|b_4|b_5|b_6|b_7|b_8)$, then $b_7^{(8)} = 0$, $b_3^{(8)} = a^{(8)}$ and $b_8^{(1\sim7)} = a^{(1\sim7)}$, where a is a non-zero byte, b_i are unknown bytes.*

Proposition 6. *Suppose the input difference of the i-round of Camellia satisfies* $(\Delta L_{i-1}, \Delta R_{i-1}) = (b_1|b_2|b_3|b_4|b_5|b_6|b_7|b_8, P(c'_1|c'_2|c'_3|c'_4|c'_5|c'_6|c'_7|c'_8))$*, and the output difference is* $(\Delta L_i, \Delta R_i) = (a_1|0|0|0|a_5|0|0|a_8, b_1|b_2|b_3|b_4|b_5|b_6|b_7|b_8)$ *with* $a_5^{(8)} = 0$ *and* $a_8^{(1\sim7)} = 0$*, where* b'_i*,* c'_i *are arbitrary bytes, and* a_1 *is a nonzero byte, then the following results hold.*

(1) The intermediate value $\Delta S_i = P^{-1}(\Delta L_i \oplus \Delta R_{i-1}) = (c'_1 \oplus a_8|c'_2 \oplus a_1 \oplus a_5 \oplus a_8|c'_3 \oplus a_1 \oplus a_5 \oplus a_8|c'_4 \oplus a_1 \oplus a_5|c'_5 \oplus a_1 \oplus a_5 \oplus a_8|c'_6 \oplus a_5 \oplus a_8|c'_7 \oplus a_5|c'_8 \oplus a_1 \oplus a_8)$*.*

(2) $\Delta S_{i,1}^{(1\sim7)} = c'^{(1\sim7)}_1$*, and* $a_8^{(8)} = \Delta S_{i,1}^{(8)} \oplus c'^{(8)}_1$*.*

(3) $\Delta S_{i,7}^{(8)} = c'^{(8)}_7$*, and* $a_5^{(1\sim7)} = \Delta S_{i,5}^{(1\sim7)} \oplus c'^{(1\sim7)}_7$*.*

(4) $a_1 = \Delta S_{i,8} \oplus c'_8 \oplus a_8$*.*

Attack on 14-Round Camellia-256 Our 14-round attack of Camellia-256 works from round 10 to round 23, where the **5+2 WKID** is applied from round 14 to round 20.

First of all, we demonstrate the relation of subkeys used in the round 10, 11, 12, 13, 21, 22, 23 and the second FL/FL^{-1} layer (KL_3, KL_4) as follows, i.e., $K_{10} = K_L^{(110\sim128,1\sim45)}$, $K_{11} = K_A^{(46\sim109)}$, $K_{12} = K_A^{(110\sim128,1\sim45)}$, $K_{13}^{(1\sim8)} = K_R^{(61\sim68)}$, $KL_{3,L}^{(1\sim9)} = K_L^{(61\sim69)}$, $KL_{3,R}^{(1\sim8)} = K_L^{(93\sim100)}$, $KL_{4,L} = K_L^{(125\sim128,1\sim28)}$, $KL_{4,R} = K_L^{(29\sim60)}$, $K_{21}^{(33\sim40)} = K_A^{(127,128,1\sim6)}$, $K_{22} = K_A^{(31\sim94)}$, $K_{23} = K_L^{(112\sim128,1\sim47)}$.

With the key relation, we can first launch the impossible differential attack in weak-key setting, then extend it to an attack for all keys, which is similar to above attacks.

Data Collection. We choose the chosen ciphertext scenario to perform the attack and begin with choosing one structure of ciphertexts which contains 2^{120} ciphertexts:

$$(C_L, C_R) = (P(y_1|\beta_2|\beta_3|\beta_4|\beta_5|\beta_6|\beta_7|\beta_8), \alpha_1|\alpha_2|\alpha_3|\alpha_4|\alpha_5|\alpha_6|\alpha_7|\alpha_8).$$

Where y_1 is fixed, while α_i ($i = 1, ..., 8$) and β_j ($j = 2, ...8$) take all possible values. Ask for the decryption to get the corresponding plaintext for each ciphertext, which results in 2^{239} pairs which satisfy the difference:

$$(\Delta C_L, \Delta C_R) = (P(0|g'_2|g'_3|g'_4|g'_5|g'_6|g'_7|g'_8), f_1|f_2|f_3|f_4|f_5|f_6|f_7|f_8).$$

Key Recovery.

1. Guess 130-bit value $(K_L^{(1\sim47,110\sim128)}|K_A^{(46\sim109)})$, for every plaintext-ciphertext pair (P, C), perform the following substeps.
   (a) Partially encrypt the plaintext P to get the intermediate value (L_{11}, R_{11}). Since 38 bits of the subkey used in FL^{-1} function, which are $KL_{4,R}^{(1\sim19)} = K_L^{(29\sim47)}$ and $KL_{4,L}^{(1\sim19)} = K_L^{(125\sim128,1\sim15)}$, have been guessed, 38-bit intermediate value $R_{FL,\{1,2\}}|R_{FL,3}^{(1\sim3)}|R_{FL,\{5,6\}}|R_{FL,7}^{(1,2)}|R_{FL,8}^{(8)}$ can be computed, where R_{FL} represents the value after the FL^{-1} function.
   (b) Partially decrypt the ciphertext C to get the intermediate values (L_{22}, R_{22}) and $P^{-1}(L_{22})$. Note that now we can compute $S_{22,\{3\sim8\}}$ as the 48-bit value $K_{22,\{3\sim8\}} = K_L^{(47\sim94)}$ is known.
   (c) Store the values (L_{11}, R_{11}) and (L_{22}, R_{22}) into a hash table Γ indexed by the following 143-bit values.
      - $R_{22,\{1,5\}}$, $R_{22,2} \oplus R_{22,3}$, $R_{22,2} \oplus R_{22,4}$, $R_{22,2} \oplus R_{22,6}$, $R_{22,2} \oplus R_{22,7}$, $R_{22,2} \oplus R_{22,8}$.
      - $S_{22,3} \oplus P^{-1}(L_{22})_3 \oplus P^{-1}(L_{22})_5$, $S_{22,4} \oplus P^{-1}(L_{22})_4 \oplus P^{-1}(L_{22})_5$, $S_{22,6} \oplus P^{-1}(L_{22})_6 \oplus P^{-1}(L_{22})_5$, $S_{22,7} \oplus P^{-1}(L_{22})_7 \oplus P^{-1}(L_{22})_5$, $S_{22,8} \oplus P^{-1}(L_{22})_8$.

   - $R_{12,7}^{(8)}$, $R_{FL,1} \oplus (R_{12,8}^{(1\sim7)}|R_{12,3}^{(8)})$, $R_{FL,2} \oplus (R_{12,8}^{(1\sim7)}|R_{12,3}^{(8)})$, $R_{FL,3}^{(1\sim3)} \oplus R_{12,8}^{(1\sim3)}$, $R_{FL,6}$, $R_{FL,7}^{(1,2)}$, $R_{FL,5} \oplus (R_{12,8}^{(1\sim7)}|R_{12,3}^{(8)})$, $R_{FL,8}^{(8)} \oplus R_{12,3}^{(8)}$.

   Then each two values lie in the same row of Γ form a pair that satisfies the following conditions.
   - The difference $\Delta R_{22} = (0|f|f|f|0|f|f|f)$, where f is a nonzero value.
   - The difference $P^{-1}(\Delta L_{22}) = (0|g'_2|g'_3|g'_4|g'_5|g'_6|g'_7|g'_8)$ satisfies $g'_3 \oplus g'_5 = \Delta S_{22,3}$, $g'_4 \oplus g'_5 = \Delta S_{22,4}$, $g'_6 \oplus g'_5 = \Delta S_{22,6}$, $g'_7 \oplus g'_5 = \Delta S_{22,7}$, $g'_8 = \Delta S_{22,8}$.
   - Assume the difference ΔR_{12} (equals to ΔL_{11}) is represented as $(b_1|b_2|b_3|b_4|b_5|b_6|b_7|b_8)$, then it satisfies $b_7^{(8)} = 0$, and the output difference of FL^{-1} function satisfies $\Delta R_{FL,1} = (b_8^{(1\sim7)}|b_3^{(8)})$, $\Delta R_{FL,2} = (b_8^{(1\sim7)}|b_3^{(8)})$, $\Delta R_{FL,3}^{(1\sim3)} = b_8^{(1\sim3)}$, $\Delta R_{FL,5} = (b_8^{(1\sim7)}|b_3^{(8)})$, $\Delta R_{FL,6} = 0$, $\Delta R_{FL,7}^{(1,2)} = 0$ and $\Delta R_{FL,8}^{(8)} = b_3^{(8)}$.

   This step performs a 135-bit filtration from 2^{239} pairs, so the expected number of remaining pairs is 2^{104}.
2. Guess 12-bit value $KL_{4,R}^{(20\sim23,25\sim32)}$, compute the output differences $\Delta R_{FL,3}^{(4\sim7)}$, $\Delta R_{FL,7}^{(3\sim7)}$ and $R_{FL,4}$ (from $b_7^{(8)} = 0$ we conclude $\Delta R_{FL,3}^{(8)} = b_3^{(8)}$). Discard the pairs that do not satisfy $\Delta R_{FL,3}^{(4\sim7)} = b_8^{(4\sim7)}$, $\Delta R_{FL,7}^{(3\sim7)} = 0$ and $\Delta R_{FL,4} = 0$, then the expected number of remaining pairs is 2^{87}. Moveover, from $\Delta R_{FL,4} = 0$ and $b_7^{(8)} = 0$, we get $\Delta R_{FL,7}^{(8)} = 0$ and $\Delta R_{FL,8}^{(1\sim7)} = b_8^{(1\sim7)}$. Therefore, at the end of this substep, all remaining pairs satisfy the condition $\Delta R_{FL} = (b|b|b|0|b|0|0|b)$, where $b = (b_8^{(1\sim7)}|b_3^{(8)})$.
3. Guess 7-bit value $K_{22}^{(9\sim15)}$, compute the intermediate value $\Delta S_{22,2}$ ($K_{22}^{(16)}$ ($K_A^{(46)}$) has already been guessed in the step 1), and discard the pairs which do not satisfy $\Delta S_{22,2} = g'_2 \oplus g'_5$. Each pair will be kept with probability 2^{-8}, so the expected number of remaining pairs is 2^{79}.
4. Compute the intermediate value $P^{-1}(\Delta R_{11}) = (c'_1|c'_2|c'_3|c'_4|c'_5|c'_6|c'_7|c'_8)$, then perform the following substeps.
   (a) Guess 17-bit subkeys $K_{12,1}$, $K_{12,7}$ and $K_{12,8}^{(1)}$, calculate the value $\Delta S_{12,\{1,7,8\}}$ (7-bit value $K_{12,8}^{(1\sim7)}$ ($K_A^{(39\sim45)}$) has been guessed in step 3), and discard the pairs which do not satisfy $\Delta S_{12,1}^{(1\sim7)} = c'^{(1\sim7)}_1$ and $\Delta S_{12,7}^{(8)} = c'^{(8)}_7$ according to proposition 7. The expected number of remaining pairs is 2^{71}. Then we compute the value $a_8 = \Delta S_{12,1} \oplus c'_1$, $a_5^{(1\sim7)} = \Delta S_{12,7}^{(1\sim7)} \oplus c'^{(1\sim7)}_7$ and $a_1 = \Delta S_{12,8} \oplus c'_8 \oplus a_8$.

   (b) For $i = 2$ to 6, guess 8-bit subkey $K_{12,i}$, compute the difference $\Delta S_{12,i}$ and discard the pairs which do not satisfy $\Delta S_{12,j} = c'_j \oplus a_1 \oplus a_5 \oplus a_8$ $(j = 2, 3, 4)$, $\Delta S_{12,5} = c'_5 \oplus a_1 \oplus a_8$ and $\Delta S_{12,6} = c'_6 \oplus a_5 \oplus a_8$. Then we expect about 2^{31} pairs remain.
5. Since all of the 128-bit value of K_A have been guessed in step 1, 3 and 4, we compute the values R_{21} and R'_{21} for every remaining pair and keep only the pairs whose $\Delta R_{21,5}^{(1)} = 0$. Then we partially decrypt $R_{21,5}$ and $R'_{21,5}$ to get the value $\Delta S_{21,5}$, keep only the pairs whose $\Delta S_{21,5} = f$, which results in 2^{22} remaining pairs.
6. Guess 17-bit value $KL_{3,L}^{(1\sim9)}$ and $KL_{3,R,1}$, compute $\Delta L_{FL,5}$, $\Delta L_{FL,8}^{(8)}$ and $\Delta L_{FL,1}$. Then discard the pairs whose $(\Delta L_{FL,5}^{(1\sim7)}|\Delta L_{FL,8}^{(8)}) \neq 0$. The expected number of remaining pairs is about 2^{14}.
7. Guess 8-bit value $K_{13,1}$, partially encrypt $L_{FL,1}$ and $L'_{FL,1}$ to get the value $\Delta S_{13,1}$ of every remaining pair. If $\Delta S_{13,1}$ equals to $\Delta R_{FL,\{1,2,3,5,8\}}$, delete this value from the list of all the 2^8 possible values $K_{13,1}$.
8. After analyzing of all remaining pairs, if the list is not empty, announce that the value in the list along with above 223-bit guessed values are the candidates of 231-bit target value of

subkey $K_A|K_R^{(61\sim68)}|K_L^{(1\sim51,53\sim69,93\sim100,110\sim128)}$, then recover the whole master key K_L and K_R by key searching. Otherwise, try the other 223-bit guess.

Complexity. The time complexity is dominated by step 1, which requires about 5 rounds' encryptions to compute the intermediate values for every plaintext and ciphertext pair. Then the time complexity is $2^{120} \times 2^{130} \times 5/14 \approx 2^{248.5}$ 14-round encryptions. The memory requirement is dominated by data collection, which needs 2^{125} bytes to store the known plaintexts and the corresponding ciphertexts. Similarly, the expected time of the attack for the whole key space is about $2^{250.5}$ 14-round encryptions.

Attack on 12-Round Camellia-192 Making use of **2+5 WKID**, we mount the weak-key impossible differential attack on 12-round Camellia-192, which is from round 3 to round 14, where the **2+5 WKID** is applied from round 5 to round 11. The attack procedure is similar to that of 14-round Camellia-256. To summarize, the time complexity of the attack is about $2^{180.1}$ 12-round encryptions. The memory requirement is dominated by step 1, which needs $2^{124.1}$ bytes to store the plaintext-ciphertext pairs. For the attack that works for the whole key space, the data complexity is about $2^{120.1}$ chosen plaintexts, and the time complexity is about 2^{184} 12-round encryptions.

4 8-Round Impossible Differentials of Camellia and Their Applications [2]

In this section, we first present a method to construct a set of differentials, which contains at least one 8-round impossible differential of Camellia with two FL/FL^{-1} layers for any fixed key. Based on this differential set, we propose a new attack strategy to recover the correct key. Finally, we mount impossible differential attacks on reduced-round Camellia-128/192/256 with the whitening and FL/FL^{-1} layers from some intermediate round.

4.1 The Construction of 8-Round Impossible Differentials of Camellia

In this section, we present some 8-round impossible differentials of Camellia with two key-dependent layers by exploiting some properties of the keyed transformation FL/FL^{-1}.

Proposition 7. *If the input difference of FL is $(a|0|0|0|a'|0|0|0)$, where $a^{(1)} = a'^{(8)} = 0$ and*

$$a'^{(i)} = \begin{cases} 0, & kl_L^{(i+1)} = 0; \\ a^{(i+1)}, & kl_L^{(i+1)} = 1; \end{cases} \quad \textit{for } 1 \le i \le 7, \tag{1}$$

then the output of FL is $(a|0|0|0|0|0|0|0)$.

Proof. By Lemma 2, we can obtain

$$\begin{aligned} \Delta Y_R &= ((\Delta X_L \cap kl_L) \lll 1) \oplus \Delta X_R = (((a|0|0|0) \cap kl_L) \lll 1) \oplus (a'|0|0|0) \\ &= ((a^{(2\sim8)}|0 \cap kl_{L,1}) \oplus a')|0|0|0. \end{aligned}$$

According to $a^{(1)} = a'^{(8)} = 0$ and the equation (1), we derive that $\Delta Y_R = 0$. Furthermore, $\Delta Y_L = \Delta X_L \oplus \Delta Y_R \oplus (\Delta Y_R \cap kl_R) = \Delta X_L = a|0|0|0$. Therefore, the output of FL is $(a|0|0|0|0|0|0|0)$. □

By Propositions 7, we construct an 8-round impossible differential of Camellia with two FL/FL^{-1} layers for any fixed subkey.

[2] By Ya Liu, Dawu Gu, Zhiqiang Liu and Wei Li.

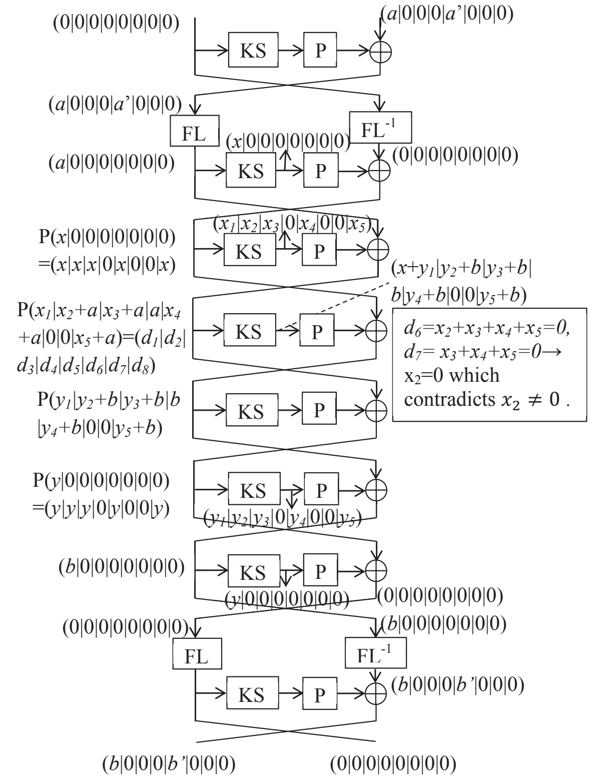


Fig. 3. The Structure of 8-Round Impossible Differentials of Camellia

Proposition 8. *For an 8 rounds of Camellia with two FL/FL^{-1} layers inserted after the first and seventh rounds, the input difference of the first round is* $(0|0|0|0|0|0|0|0, a|0|0|0|a'|0|0|0)$ *and the output difference of the eighth round is* $(b|0|0|0|b'|0|0|0, 0|0|0|0|0|0|0|0)$ *with* a *and* b *being nonzero bytes and* $a^{(1)} = b^{(1)} = a'^{(8)} = a'^{(8)} = 0$. *Four subkeys* $kl_i (i = 1, \cdots, 4)$ *are used in two* FL/FL^{-1} *layers. If* a' *and* b' *satisfy the following equations:*

$$a'^{(i)} = \begin{cases} 0, & \text{if } kl_1^{(i+1)} = 0; \\ a^{(i+1)}, & \text{if } kl_1^{(i+1)} = 1; \end{cases} \quad b'^{(i)} = \begin{cases} 0, & \text{if } kl_4^{(i+1)} = 0; \\ b^{(i+1)}, & \text{if } kl_4^{(i+1)} = 1; \end{cases} \quad \text{for } 1 \le i \le 7,$$

then

$$(0|0|0|0|0|0|0|0, a|0|0|0|a'|0|0|0) \nrightarrow_8 (b|0|0|0|b'|0|0|0, 0|0|0|0|0|0|0|0)$$

is an 8-round impossible differential of Camellia with two FL/FL^{-1} *layers (See Fig. 3).*

Proof. By proposition 7, we obtain that the input difference of the second round and the output difference of the seventh round are $(a|0|0|0|0|0|0|0, 0|0|0|0|0|0|0|0)$ and $(0|0|0|0|0|0|0|0, b|0|0|0|0|0|0|0)$, respectively. In [23], Wu *et al.* constructed an 8-round impossible differential of Camellia without the FL/FL^{-1} layers: $(0|0|0|0|0|0|0|0, a|0|0|0|0|0|0|0) \nrightarrow_8 (b|0|0|0|0|0|0|0, 0|0|0|0|0|0|0|0)$ where a and b are nonzero bytes. Thus, $(a|0|0|0|0|0|0|0, 0|0|0|0|0|0|0|0) \nrightarrow_6 (0|0|0|0|0|0|0|0, b|0|0|0|0|0|0|0)$ is a 6-round impossible differential. In other word, the input difference $(a|0|0|0|0|0|0|0, 0|0|0|0|0|0|0|0)$ cannot result in the output difference $(0|0|0|0|0|0|0|0, b|0|0|0|0|0|0|0)$ after six-round encryption. Therefore,

$$(0|0|0|0|0|0|0|0, a|0|0|0|a'|0|0|0) \nrightarrow_8 (b|0|0|0|b'|0|0|0, 0|0|0|0|0|0|0|0)$$

is an 8-round impossible differential of Camellia with two FL/FL^{-1} layers. □

For any fixed subkey, an 8-round impossible differential with two FL/FL^{-1} layers can be constructed. Each possible value of $kl_1^{(2\sim8)} \mid kl_4^{(2\sim8)}$ corresponds to the existence of an 8-round impossible differential. For example, if the subkeys $kl_1^{(2\sim8)} = kl_4^{(2\sim8)} = 0_{(7)}$, then $(0|0|0|0|0|0|0|0,\ a|0|0|0|0|0|0|0) \nrightarrow_8 (b|0|0|0|0|0|0|0,\ 0|0|0|0|0|0|0|0)$ is an 8-round impossible differential of Camellia with two keyed layers, where $a^{(1)} = b^{(1)} = 0$. If $kl_1^{(2\sim8)} = kl_4^{(2\sim8)} = 1_{(7)}$, then $(0|0|0|0|0|0|0|0,\ a|0|0|0|a'|0|0|0) \nrightarrow_8 (b|0|0|0|b'|0|0|0, 0|0|0|0|0|0|0|0)$ is an 8-round impossible differential of Camellia with two keyed layers, where a, b, a' and b' are nonzero bytes and satisfy $a^{(1)} = b^{(1)} = a'^{(8)} = b'^{(8)} = 0$, $a'^{(1\sim7)} = a^{(2\sim8)}$ and $b'^{(1\sim7)} = b^{(2\sim8)}$. All possible values of $kl_1^{(2\sim8)} \mid kl_4^{(2\sim8)}$ are from $0_{(14)}$ to $1_{(14)}$. Denote their corresponding impossible differentials by Δ_i for $0 \le i \le 2^{14} - 1$. However, it is possible that different values of $kl_1^{(2\sim8)}$ may result in the same values of a', and different values of $kl_4^{(2\sim8)}$ may lead to the same values of b'. Therefore, some of 2^{14} differentials are equal to each other. Let A be a set including all differentials $\Delta_i (0 \le i \le 2^{14} - 1)$.

$$A = \{\Delta_i \mid 0 \le i \le 2^{14} - 1\} \triangleq \{\delta_j \mid 1 \le j \le t\},\ \text{where } t \le 2^{14}.$$

According to Proposition 8, 8-round differentials of A must have the forms:

$$\Delta = (0|0|0|0|0|0|0|0, a|0|0|0|a'|0|0|0) \nrightarrow_8 (b|0|0|0|b'|0|0|0, 0|0|0|0|0|0|0|0)$$

with a and b being nonzero bytes and $a^{(1)} = b^{(1)} = a'^{(8)} = b'^{(8)} = 0$. Among them, a' and b' are either zero or nonzero bytes. We divide all differentials of A into three cases in order to simplify our analysis. The first one is $a' = b' = 0$. The second one is $a' = 0$ and $b' \ne 0$, or $a' \ne 0$ and $b' = 0$. The last one is $a' \ne 0$ and $b' \ne 0$.

By proposition 8, we only know the existence of an 8-round impossible differential of Camellia with two FL/FL^{-1} layers for any fixed key, but cannot distinguish it from other differentials of A. Therefore, we require to propose a new attack strategy to recover the correct key based on this differential set.

The Attack Strategy. Select a differential δ_i from A. Based on it, we mount an impossible differential attack on reduced-round Camellia given enough plaintext pairs. More concretely, we select enough plaintexts such that all wrong keys will be removed with high probability if δ_i is an impossible differential.

1. If one subkey remains, we recover the secret key by the key schedule and verify whether it is correct by some plaintext-ciphertext pairs. If success, end this attack. Otherwise, try another differential $\delta_j (j \ne i)$ of A and perform a new impossible differential attack.
2. If no subkey or more than one subkeys is left, select another differential of A to execute a new impossible differential attack.

□

Our attack strategy can really recover the correct key. As a matter of fact, if δ_i is an impossible differential, we make sure the expected number of remaining wrong keys will be almost zero given enough chosen plaintexts. Therefore, we only consider those differentials which result in one subkey remaining. By Proposition 8, we know the differential set A must contain an impossible differential. So we try each differential of A until the correct key is recovered. The worst scenario is that the correct key is retrieved from the last try.

4.2 Impossible Differential Attack on 13-round Camellia-256

Based on three scenarios of differentials in A, we present an impossible differential attack on 13-round Camellia-256 with the FL/FL^{-1} and whitening layers. For each of three cases, we

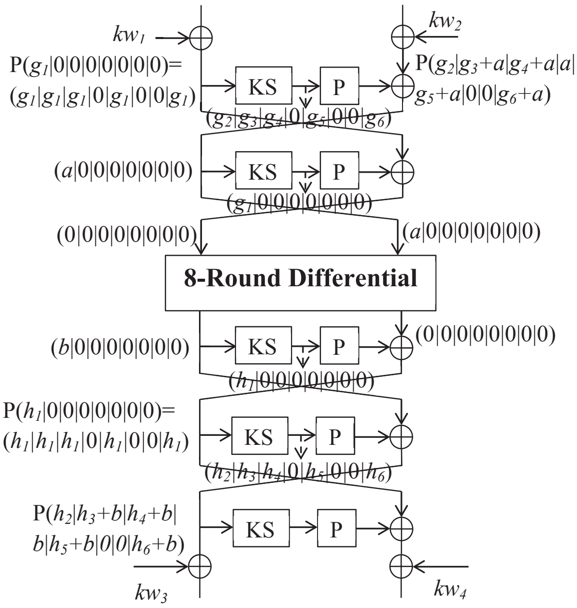


Fig. 4. Impossible Differential Attack on 13-round Camellia-256 for Case 1

put two additional rounds on the top and three additional rounds on the bottom of the 8-round differentials of A. On the basis of this structure, we can attack 13-round Camellia-256 from rounds 4 to 16 or from rounds 10 to 22. Similarly, we put three additional rounds on the plaintext side and two additional rounds on the ciphertext side to attack 13-round Camellia-256 from rounds 3 to 15 or from rounds 9 to 21. Some previously known skills such as building hash tables and the early abort technique [15] are also adopted in order to reduce the time complexity. In this section, we only elaborate the attack procedure of impossible differential cryptanalysis of 13-round Camellia-256 from rounds 4 to 16. Before introducing our attack, we list some notations, i.e.,

$$k_a \triangleq kw_1 \oplus k_4, k_b \triangleq kw_2 \oplus k_5, k_c \triangleq kw_3 \oplus k_{16}, k_d \triangleq kw_4 \oplus k_{15}, k_e \triangleq kw_3 \oplus k_{14}.$$

We use these equivalent subkeys k_a, k_b, k_c, k_d and k_e instead of the round subkeys k_4, k_5, k_{14}, k_{15} and k_{16} so as to remove the whitening layers. This new cipher acts as the original one.

Based on the attack strategy in section 4.1, we mount an impossible differential attack on 13-round Camellia-256 by using differentials of A until the correct key is recovered. In the following, we discuss this attack by three cases.

Case 1 $a' = b' = 0$: At this time, the differential $\Delta = (0|0|0|0|0|0|0|0, a|0|0|0|0|0|0|0) \rightarrow_8 (b|0|0|0|0|0|0|0, 0|0|0|0|0|0|0|0)$, where a and b are nonzero bytes and $a^{(1)} = b^{(1)} = 0$ (See Fig. 4).

Data Collection. Select a structure of plaintexts, which contains 2^{55} plaintexts with the following forms:

$$(P(\alpha_1|x_1|x_2|x_3|x_4|x_5|x_6|x_7), P(\alpha_2|\alpha_3|\alpha_4|\alpha_5|\alpha_6|x_8|x_9|\alpha_7)), \tag{2}$$

where $\alpha_5^{(1)}, x_i(1 \leq i \leq 9)$ are fixed and $\alpha_j(1 \leq j \leq 7, i \neq 5), \alpha_5^{(2\sim 8)}$ takes all possible values. Clearly, each structure forms 2^{109} plaintext pairs, the differences of which have the forms: $(P(g_1|0|0|0|0|0|0|0), P(g_2|g_3 \oplus a|g_4 \oplus a|a|g_5 \oplus a|0|0|g_6 \oplus a))$ with a and $g_i(1 \leq i \leq 6)$ being nonzero bytes and $a^{(1)}$=0. We take all possible values of $(\alpha_5^{(1)}, x_4, x_8, x_9)$ and 2^{43} different values of $x_i(1 \leq i \leq 7, i \neq 4)$ to obtain 2^{68} special structures. In total, there are 2^{123} chosen plaintexts which form 2^{177} plaintext pairs. Encrypt these plaintext pairs to obtain the corresponding ciphertext pairs. If the left halves of their ciphertexts differences have the form: $P(h_1|h_2 \oplus b|h_3 \oplus b|b|h_5 \oplus b|0|0|h_8 \oplus b)$ with $b^{(1)} = 0$, then these pairs will be kept. The expected number of remaining pairs is about 2^{160}.

Key Recovery.

1. Guess $k_{a,1}$. For each remaining pair, check whether the equation $\Delta S_{4,1} = (P^{-1}(\Delta P_R))_1$ holds. If $\Delta S_{4,1} \neq (P^{-1}(\Delta P_R))_1$ for some pair, then this pair will be discarded. Next guess each possible value of $k_{a,l}$ for $l = 2, 3, 5, 8$. Keep only the pairs satisfying $\Delta S_{4,l} = (P^{-1}(\Delta P_R))_l \oplus (P^{-1}(\Delta P_R))_4$. The total probability of this event is about 2^{-40}. Thus the expected number of remaining pairs is about 2^{120}. Finally, guess $k_{a,\{4,6,7\}}$ and compute the inputs of the fifth round.
2. Guess $k_{b,1}$ and test whether $\Delta S_{5,1}$ is equal to $(P^{-1}(\Delta P_L))_1$ for each remaining pair. If $\Delta S_{5,1} \neq (P^{-1}(\Delta P_L))_1$ for one pair, then this pair will be removed. The probability that to happen is about 2^{-8}. Thus about 2^{112} pairs will be kept.
3. Guess $k_{c,l}$ for $2 \leq l \leq 8$. Verify whether $\Delta S_{16,l}$ is equal to $(P^{-1}(\Delta C_R))_l$ for every remaining pair. If $\Delta S_{16,l} \neq (P^{-1}(\Delta C_R))_l$ for some pair, then this pair is discarded. The total probability of this event is 2^{-56}. Therefore, we expect about 2^{56} pairs remain. Next guess $k_{c,1}$ and compute the outputs of the 15-th round for each of the remaining pairs.
4. Guess $k_{d,l}$ for $l = 1, 2, 3, 5, 8$. Verify whether the equations, $\Delta S_{15,1} = (P^{-1}(\Delta C_L))_1$ and $\Delta S_{15,j} = (P^{-1}(\Delta C_L))_j \oplus (P^{-1}(\Delta C_L))_4$ for $j = 2, 3, 5, 8$, hold for every remaining pair. The total probability that to happen is about 2^{-40}. Thus there are about 2^{16} pairs remain. Next guess other bytes of k_d and calculate the outputs of the 14-th round.
5. Guess $k_{e,1}$ and compute the output difference of the S-Boxes in the 14-th round. If $\Delta S_{14,1}$ is equal to $(P^{-1}(\Delta L_{14}))_1$, then we remove this value of $k_{e,1}$ with $(k_a, k_{b,1}, k_c, k_d)$. The probability of this event is about 2^{-8}. After trying all possible values of $(k_a, k_{b,1}, k_c, k_d, k_{e,1})$, if only one joint subkey remains, then Δ is likely to be an impossible differential. At this time, we recover the secret key by the key schedule and verify whether it is correct by some plaintext-ciphertext pairs. If no subkey or more than one subkeys is left, then Δ is possible to exist. At this time, try another differential of A. As a matter of fact, if Δ is an impossible differential, the expected number of the wrong subkeys remaining is about $2^{208} \times (1 - 2^{-8})^{2^{16}} \approx 2^{-161.4}$. We consider that all wrong subkeys are removed and only the correct subkey is left. Therefore, we require to perform the following Step 6 only if one subkey is left.
6. We can recover the secret key from this unique 208-bit subkey $(k_a, k_{b,1}, k_c, k_d, k_{e,1})$. By the key schedule of Camellia-256, we can obtain:

$$k_a = kw_1 \oplus k_4 = (K_L \lll 0)_L \oplus (K_R \lll 15)_R, \tag{3}$$

$$k_b = kw_2 \oplus k_5 = (K_L \lll 0)_R \oplus (K_A \lll 15)_L, \tag{4}$$

$$k_c = kw_3 \oplus k_{16} = (K_B \lll 111)_L \oplus (K_B \lll 60)_R, \tag{5}$$

$$k_d = kw_4 \oplus k_{15} = (K_B \lll 111)_R \oplus (K_B \lll 60)_L, \tag{6}$$

$$k_e = kw_3 \oplus k_{14} = (K_B \lll 111)_L \oplus (K_R \lll 60)_R. \tag{7}$$

   We first guess each possible values of K_B. By the equations (5) and (6), we discard some wrong candidates of K_B with the probability 2^{-128}. Therefore, only one value of K_B is left. Then we calculate 8 bits of K_R by the equation (7). Guess the remaining unknown 120 bits of K_R. By property 4 of [17], we can compute the corresponding value for (K_L, K_A). According to the equations (3) and (4), we can discard some wrong candidates of (K_L, K_A). Therefore, the number of the remaining main keys is approximately $2^{120} \times 2^{-72} = 2^{48}$. By about 2^{48} trail encryptions, if some key is correct, stop the attack. Otherwise, try another differential of A.

Case 2 $a' = 0$ **and** $b' \neq 0$**, or** $a' \neq 0$ **and** $b' = 0$**:** We only attack a special scenario, i.e., $a' = 0$, $b' \neq 0$ and $b'^{(1\sim7)} = b^{(2\sim8)}$. The others can be attacked in the similar way. At this moment, the differential is $\Delta' = (0|0|0|0|0|0|0|0, a|0|0|0|0|0|0|0) \rightarrow_8 (b|0|0|0|b'|0|0|0, 0|0|0|0|0|0|0|0)$, where a, b and b' are non-zero bytes, $b'^{(1\sim7)} = b^{(2\sim8)}$ and $a^{(1)} = b^{(1)} = b'^{(8)} = 0$.

Data Collection. We apply 2^{68} special structures of Case 1 above. Totally, there are 2^{123} chosen plaintexts which form 2^{177} pairs. At this moment, the form of the ciphertext difference is random.

Key Recovery.

1. Guess $k_{c,l}$ for $2 \leq l \leq 8$ and $l \neq 5$. Verify whether the equation $\Delta S_{16,l} = (P^{-1}(\Delta C_R))_l$ holds for every remaining pair. If $\Delta S_{16,l} \neq (P^{-1}(\Delta C_R))_l$ for some pair, then this pair is discarded. The whole probability of this event is 2^{-48}. Therefore, we expect about 2^{129} pairs remain. Next guess $k_{c,\{1,5\}}$ and compute the outputs of the 15-th round for each of the remaining pairs.
2. We first guess $k_{d,1}$ and check whether the equation $\Delta S_{15,1} = (P^{-1}(\Delta C_L))_1$ holds for each remaining pair. If $\Delta S_{15,1} = (P^{-1}(\Delta C_L))_1$ for one pair, then this pair will be kept. Otherwise, this pair will be discarded. Second, guess $k_{d,8}$ and keep only the pairs satisfying $\Delta S^{(1)}_{15,8} = (P^{-1}(\Delta C_L))^{(1)}_8$. Third, guess $k_{d,\{2\sim7\}}$. Test whether $\Delta S_{15,l} = (P^{-1}(\Delta C_L))_l \oplus (((P^{-1}(\Delta C_L))_8 \oplus \Delta S_{15,8})^{(2\sim8)}|0)$ for $l = 6, 7$ and $\Delta S_{15,l} = (P^{-1}(\Delta C_L))_l \oplus (P^{-1}(\Delta C_L))_8 \oplus \Delta S_{15,8} \oplus (P^{-1}(\Delta C_L))_7 \oplus \Delta S_{15,7}$ for $l = 2, 3, 4, 5$. The total probability of this step is about 2^{-57}. So the expected number of remaining pairs is approximately 2^{72}. Compute the outputs of the 14-th round for each remaining pair.
3. Guess $k_{e,l}$ for $l = 1, 5$. Verify whether the equation $\Delta S_{14,l} = (P^{-1}(\Delta L_{14}))_l$ holds for each remaining pair. If this equation is correct for some pair, then this pair will be kept. The probability of this event is about 2^{-16}. About 2^{56} pairs will be kept.
4. Guess each of possible values k_a as like Case 1 for all remaining pairs. Finally, we expect about 2^{16} pairs remain and calculate the inputs of the fifth round.
5. Guess $k_{b,1}$. This step is similar to Step 5 of Case 1. If only one joint subkey is left, then we consider Δ' is an impossible differential and recover the secret key by the key schedule. Otherwise try another differential of A. In fact, the expected number of the wrong subkeys remaining is approximately $2^{216} \times (1 - 2^{-8})^{2^{16}} \approx 2^{-153.4}$ if Δ' is an impossible differential.
6. This step is similar to Step 6 of Case 1. The difference is that the equation (7) can give 16 bits of K_R. Therefore, we only require to guess 112 bits of K_R. About 2^{40} keys will be left. By about 2^{40} trail encryptions, if some key is correct, stop the attack. Otherwise, try another differential of A.

Case 3 $a' \neq 0$ **and** $b' \neq 0$**:** We only discuss an example, i.e., $a'^{(1\sim7)} = a^{(2\sim8)}$ and $b'^{(1\sim7)} = b^{(2\sim8)}$. At this moment, the differential is $\Delta'' = (0|0|0|0|0|0|0|0, a|0|0|0|a'|0|0|0) \rightarrow_8 (b|0|0|0|b'|0|0|0, 0|0|0|0|0|0|0|0)$, where a, b, a' and b' are nonzero bytes and $a^{(1)} = b^{(1)} = a'^{(8)} = b'^{(8)} = 0$.

Data Collection. Continue to adopt 2^{123} chosen plaintexts in Case 1. Because each structure of Case 1 takes all possible values of $\alpha^{(1)}_5, x_4, x_8$ and x_9, 2^{123} chosen plaintexts of Case 1 are equivalent to 2^{43} structures, each of which contains 2^{80} plaintexts with the forms: $(P(\beta_1|y_1|y_2|y_3|\beta_2|y_4|y_5|y_6), \beta_3|\beta_4|\beta_5|\beta_6|\beta_7|\beta_8|\beta_9|\beta_{10})$, where $y_i (1 \leq i \leq 6)$ are fixed and $\beta_j (1 \leq j \leq 10)$ takes all possible values. It is obvious that one structure generates 2^{159} pairs. Totally, there are approximately 2^{202} plaintext pairs satisfying the input differences.

Key Recovery.

1. Guess each byte of $k_c, k_d, k_{e,\{1,5\}}$. This step is similar to Case 2 above. After guessing these subkeys, we expect about 2^{81} pairs remain.
2. Guess $k_{a,1}$, $k_{a,8}$, $k_{a,\{6,7\}}$ and $k_{a,\{2\sim5\}}$ in turn. After our test, about 2^{24} pairs will be kept. Compute the inputs of the fifth round for every remaining pair.
3. Guess $k_{b,5}$ and kept these pairs satisfying $\Delta S_{5,5} = (P^{-1}(\Delta P_L))_5$. Finally, there are about 2^{16} pairs remain. Next guess $k_{b,1}$ and test whether $\Delta S_{5,1}$ is equal to $(P^{-1}(\Delta P_L))_1$ for the remaining pairs. If $\Delta S_{5,1} = (P^{-1}(\Delta P_L))_1$ for some pair, then this value $k_{b,1}$ with the guessed value $(k_a, k_{b,5}, k_c, k_d, k_{e,\{1,5\}})$ are removed. After guessing all possible values $(k_a, k_{b,\{1,5\}}, k_c, k_d, k_{e,\{1,5\}})$, if only one joint subkey is left, then we consider Δ'' is an impossible differential. At this moment, we execute the following step. Otherwise try another differential of A. As a matter of fact, the expected number of the wrong subkeys remaining is approximately $2^{224} \times (1 - 2^{-8})^{2^{16}} \approx 2^{-145.4}$ if Δ'' is an impossible differential.
4. Similarly, we require to recover the secret key only if one subkey is left. Compared with Step 6 of Case 2 above, the difference is the equation (5) can give 16 bits of K_L. Therefore, the number of the remaining main keys is approximately 2^{32}. By about 2^{32} trail encryptions, if some key is correct, stop the attack. Otherwise, try another differential of A.

The Algorithm of Impossible Differential Attack on 13-Round Camellia-256:

For each differential δ_i of A, do

If δ_i belongs to Case 1, we perform the attacking procedure of Case 1.

If δ_i belongs to Case 2, we perform the similar attacking procedure of Case 2.

If δ_i belongs to Case 3, we perform the similar attacking procedure of Case 3.

If the correct key is recovered, end this algorithm. Otherwise, try another differential of A.

□

Table 2. Time Complexity of Cases 1

Step	Time Complexity (1-round encryptions)
2	$2^{160} \times 2 \times 2^8 \times 5 \times \frac{1}{8} + 2^{120} \times 2 \times 2^{64} \times \frac{3}{8} \approx 2^{183.6}$
3	$2^{120} \times 2 \times 2^{64} \times 2^8 \times \frac{1}{8} = 2^{190}$
4	$2^{112} \times 2 \times 2^{72} \times 2^8 \times 7 \times \frac{1}{8} + 2^{56} \times 2 \times 2^{136} \times \frac{1}{8} = 2^{193}$
5	$2^{56} \times 2 \times 2^{136} \times 2^8 \times 5 \times \frac{1}{8} + 2^{16} \times 2 \times 2^{200} \times \frac{3}{8} \approx 2^{215.6}$
6	$2^{208} \times 2 \times (1 + (1 - 2^{-8}) + \cdots + (1 - 2^{-8})^{2^{16}}) \times \frac{1}{8} \approx 2^{214}$
7	$2^{120} \times 6 + 2^{48} \times 13 \approx 2^{122.4}$

Analysis of Complexity In table 2, we list the time complexity of each step in Case 1. We find that the total time complexity is about 2^{216} 1-round encryptions. Similarly, we can compute the time complexities of Case 2 and Case 3. For Case 2, the total time complexity is approximately 2^{224} 1-round encryptions. For Case 3, the total time complexity is approximately $2^{240.8}$ 1-round encryptions. Thus the total time complexity is at most $2^{14} \times 2^{240.8} \times \frac{1}{13} \approx 2^{251.1}$ 13-round encryptions. Furthermore, the total data and memory complexities are 2^{123} chosen plaintexts and 2^{208} bytes, respectively.

4.3 Impossible Differential Attack on 12-round Camellia-192

In this part, an impossible differential attack on 12-round Camellia-192 is executed. We set two additional rounds on the top and on the bottom of our 8-round differentials, respectively. By applying it, we can attack 12-round Camellia-192 from rounds 4 to 15 with the 8-round impossible differentials inserted rounds 6 to 13. Similarly, we can also attack 12-round Camellia-192 from rounds i to $i+11$ where $i=3,5,9,10$. Some equivalent subkeys k_a and k_b are defined as before. In addition, let

$$k'_d = kw_3 \oplus k_{15} = (K_B \lll 111)_L \oplus (K_B \lll 60)_L, \tag{8}$$

$$k'_e = kw_4 \oplus k_{14} = (K_B \lll 111)_R \oplus (K_R \lll 60)_R. \tag{9}$$

Case 1 $a' = b' = 0$**:** The differential is Δ.

Data Collection. We select the same plaintexts of Case 1 mentioned in section 4.2. I.e., 2^{123} chosen plaintexts can form 2^{177} pairs. Encrypt these plaintext pairs. Keep only the pairs which have the form of ciphertext differences: $(P(h_1|0|0|0|0|0|0|0), P(h_2|h_3 \oplus b|h_4 \oplus b|b|h_5 \oplus b|0|0|h_6 \oplus b))$, where b and $h_i (1 \le i \le 6)$ are nonzero bytes and $b^{(1)} = 0$. The expected number of remaining pairs is 2^{104}.

Key Recovery. Guess all possible values $(k_a, k_{b,1}, k'_d, k'_{e,1})$ and discard those subkeys which acquire the input and output differences of Δ. This step is similar to section 4.2. If Δ is an impossible differential, about $2^{144} \times (1-2^{-8})^{2^{16}} \approx 2^{-225.4}$ wrong subkeys are expected to remain. Therefore, we will recover the secret key by the key schedule of Camellia-192 only if one subkey is left. Otherwise, try another differential of A. By the key schedule of Camellia-192, we can recover the secret key from the 144-bit subkey $(k_a, k_{b,1}, k'_d, k'_{e,1})$. We first guess all possible values of K_B. By the equation (8), we can get rid of some wrong candidates of K_B with the probability 2^{-64}. So about 2^{64} values of K_B remain. Then we can compute 8 bits of K_R by the equation (9). Guessing the remaining unknown 56 bits of K_R, we calculate (K_L, K_A) and remove some wrong values of (K_L, K_A, K_R) by the equations (3) and (4). The expected number of remaining secret keys is approximately $2^{64} \times 2^{56} \times 2^{-64} \times 2^{-8} = 2^{48}$. By about 2^{48} trail encryptions, if the correct key is retrieved, end the attack. Otherwise, try another differential of A.

Case 2 $a' = 0, b' \neq 0$ **or** $a' \neq 0, b' = 0$**:** For simplicity, we consider a special differential Δ'.

We still select 2^{123} plaintexts above. In total, there are 2^{68} special structures, each of which contains 2^{55} plaintexts. Encrypt these plaintext pairs. If the left halves of their ciphertexts differences have the forms: $P(h|0|0|0|h'|0|0|0)$ with h and h' being nonzero bytes, then these pairs will be kept. Consequently, the expected number of remaining pairs is about 2^{129}. Similarly, we can remove some subkeys $(k_a, k_{b,1}, k'_d, k'_{e,\{1,5\}})$ which obtain the input and output differences of Δ' for some pair. If only one subkey is left, we recover the secret key by the key schedule. Otherwise, try another differential of A. In fact, if Δ' is an impossible differential, about $2^{-217.4} (\approx 2^{152} \times (1-2^{-8})^{2^{16}})$ wrong subkeys will be left.

Case 3 $a' \neq 0, b' \neq 0$**:** A special differential Δ'' will be considered.

The similar attacking procedure can be performed as before. We select 2^{43} structure, each of which contains 2^{80} plaintexts. Totally, they can form 2^{202} pairs. After filtering some pairs by the ciphertext differences, about 2^{154} pairs are expected to remain. The following steps can be preformed in the similar way.

By the careful analysis, we found that the time complexity of Case 3 is maximal. Therefore, the total time complexity is at most $2^{14} \times 2^{173.2} \approx 2^{187.2}$ 12-round encryptions. The data and memory complexities are 2^{123} chosen plaintexts and 2^{160} bytes, respectively.

4.4 Impossible Differential Attack on 11-round Camellia-128

For Camellia-128, we put two additional rounds on the top and one additional round on the bottom of 8-round differentials. Based on it, we attack 11-round Camellia-128 from rounds 4 to 14 or rounds 10 to 20. Similarly, we can also attack Camellia-128 from rounds 5 to 15 and rounds 11 to 21 by setting one additional round on the top and two rounds on the bottom. Here we present an attack on 11-round Camellia-128 from rounds 4 to 14 briefly. Similarly, we divide all possible differentials into three different cases as before. For Case 1, we take 2^{67} special structures (2). Totally, the data complexity is 2^{122} chosen plaintexts which form 2^{176} pairs. Their input differences have the form $(P(g_1|0|0|0|0|0|0|0), P(g_2|g_3 \oplus a|g_4 \oplus a|a|g_5 \oplus a|0|0|g_6 \oplus a))$, where a and $g_i (1 \leq i \leq 6)$ are nonzero bytes and $a^{(1)} = 0$. Encrypt these pairs to acquire the corresponding ciphertext pairs. Then we discard some pairs whose ciphertext differences don't satisfy these form: $(b|0|0|0|0|0|0|0, P(h|0|0|0|0|0|0|0))$ with b and h being non-zero bytes and $b^{(1)} = 0$. The number of remaining pairs after this test is 2^{63}. Guess $k_{e,1}$ and verify whether the equation $\Delta S_{14,1} = (P^{-1}(\Delta C_R))_1$ holds. It is obvious that there are about 2^{55} pairs remain. Next guess $(k_a, k_{b,1})$, operate the similar step as section 4.2. If only one subkey is left, we retrieve the secret key by the key schedule. Otherwise, try anther differential of A. As a matter of fact, if Δ is an impossible differential, the expected number of remaining pairs is $2^{80} \times (1 - 2^{-8})^{15} \approx 2^{-104.7}$. For other two cases, we can accomplish the similar attack procedure.

We find that the dominant time complexity of all steps in three cases is the data collection. Therefore, the total data, time and memory complexities are 2^{122} chosen plaintext, 2^{122} 11-round encryptions and 2^{102} bytes, respectively.

5 Conclusion

In this paper, we have presented new insight on impossible differential cryptanalysis of reduced-round Camellia with the FL/FL^{-1} and whitening layers. First, we propose impossible differential attacks on reduced-round Camellia for 75% of the keys, which are then extended to attacks that work for the whole key space. Specifically, we attack 10-round Camellia-128, 11-round Camellia-192 and 12-round Camellia-256 which start from the first round and include the whitening layers. Meanwhile, we also attack 12-round Camellia-192 and 14-round Camellia-256 with two FL/FL^{-1} layers. Second, we construct a set of differentials including at least one 8-round impossible differential of Camellia with two layers FL/FL^{-1}. These impossible differentials have the same length as the best known impossible differential of Camellia without FL/FL^{-1} layers. Therefore, our result shows that the keyed functions cannot thwart impossible differential attack effectively. Based on it, we propose a new strategy to derive an effective attack on reduced-round Camellia which do not start the first round but include the whitening and FL/FL^{-1} layers. More concretely, we mount impossible differential attacks on 11-round Camellia-128, 12-round Camellia-192 and 13-round Camellia-256.

References

1. Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: Camellia: A 128-bit block cipher suitable for multiple platforms - design and analysis. In: Stinson, D.R., Tavares, S.E. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 2012, pp. 39–56. Springer (2000)
2. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In: EUROCRYPT. pp. 12–23 (1999)
3. Chen, J., Jia, K., Yu, H., Wang, X.: New impossible differential attacks of reduced-round Camellia-192 and Camellia-256. In: Parampalli, U., Hawkes, P. (eds.) ACISP. Lecture Notes in Computer Science, vol. 6812, pp. 16–33. Springer (2011)
4. CRYPTREC-Cryptography Research and Evaluation Committees: report. Archive (2002), http://www.ipa.go.jp/security/enc/CRYPTREC/index-e.html

5. Hatano, Y., Sekine, H., Kaneko, T.: Higher order differential attack of Camellia (II). In: Nyberg, K., Heys, H.M. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 2595, pp. 129–146. Springer (2002)
6. International Standardization of Organization (ISO): International standard - ISO/IEC 18033-3. Tech. rep., Information technology - Security techniques - Encryption algrithm - Part 3: Block Ciphers (July 2005)
7. Knudsen, L.R.: DEAL - a 128-bit block cipher. Tech. rep., Department of Informatics, University of Bergen, Norway (1998), technical report
8. Kühn, U.: Improved cryptanalysis of MISTY1. In: Daemen, J., Rijmen, V. (eds.) FSE. Lecture Notes in Computer Science, vol. 2365, pp. 61–75. Springer (2002)
9. Lee, S., Hong, S., Lee, S., Lim, J., Yoon, S.: Truncated differential cryptanalysis of Camellia. In: Kim, K. (ed.) ICISC. Lecture Notes in Computer Science, vol. 2288, pp. 32–38. Springer (2001)
10. Lei, D., Li, C., Feng, K.: New observation on Camellia. In: Preneel, B., Tavares, S.E. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 3897, pp. 51–64. Springer (2005)
11. Lei, D., Li, C., Feng, K.: Square like attack on Camellia. In: Qing, S., Imai, H., Wang, G. (eds.) ICICS. Lecture Notes in Computer Science, vol. 4861, pp. 269–283. Springer (2007)
12. Li, L., Chen, J., Jia, K.: New impossible differential cryptanalysis of reduced-round Camellia. In: Lin, D., Tsudik, G., Wang, X. (eds.) CANS. Lecture Notes in Computer Science, vol. 7092, pp. 26–39. Springer (2011)
13. Li, L., Chen, J., Wang, X.: Security of reduced-round Camellia against impossible differential attack. IACR Cryptology ePrint Archive 2011, 524 (2011)
14. Liu, Y., Gu, D., Liu, Z., Li, W., Man, Y.: Improved results on impossible differential cryptanalysis of reduced-round Camellia-192/256. IACR Cryptology ePrint Archive 2011, 671 (2011)
15. Lu, J., Kim, J., Keller, N., Dunkelman, O.: Improving the efficiency of impossible differential cryptanalysis of reduced Camellia and MISTY1. In: Malkin, T. (ed.) CT-RSA. Lecture Notes in Computer Science, vol. 4964, pp. 370–386. Springer (2008)
16. Lu, J., Wei, Y., Kim, J., Fouque, P.A.: Cryptanalysis of reduced versions of the Camellia block cipher. In: Preproceeding of SAC (2011)
17. Lu, J., Wei, Y., Kim, J., Pasalic, E.: The higher-order meet-in-the-middle attack and its application to the Camellia block cipher. In: Presented in part at the First Asian Workshop on Symmetric Key Cryptography (ASK 2011) (Augst 2011), a full version is available at https://sites.google.com/site/jiqiang/
18. Mala, H., Shakiba, M., Dakhilalian, M., Bagherikaram, G.: New results on impossible differential cryptanalysis of reduced-round Camellia-128. In: Jr., M.J.J., Rijmen, V., Safavi-Naini, R. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 5867, pp. 281–294. Springer (2009)
19. NESSIE: New european schemes for signatures, integrity, and encryption, final report of eurpean project IST-1999-12324. Archive (1999), http://www.cosic.esat.kuleuven.be/nessie/Bookv015.pdf
20. Shirai, T.: Differential, linear, boomerange and rectangle cryptanalysis of reduced-round Camellia. Proceedings of 3rd NESSIE Workshop, Munich, Germany (November 6-7 2002)
21. Sugita, M., Kobara, K., Imai, H.: Security of reduced version of the block cipher Camellia against truncated and impossible differential cryptanalysis. In: Boyd, C. (ed.) ASIACRYPT. Lecture Notes in Computer Science, vol. 2248, pp. 193–207. Springer (2001)
22. Wu, W., Feng, D., Chen, H.: Collision attack and pseudorandomness of reduced-round Camellia. In: Handschuh, H., Hasan, M.A. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 3357, pp. 252–266. Springer (2004)
23. Wu, W., Zhang, W., Feng, D.: Impossible differential cryptanalysis of reduced-round ARIA and Camellia. J. Comput. Sci. Technol. 22(3), 449–456 (2007)

Improved Rebound Attack on the Finalist Grøstl

Jérémy Jean[1,⋆,⋆⋆], María Naya-Plasencia[2,⋆], and Thomas Peyrin[3,⋆⋆⋆]

[1] École Normale Supérieure, France
[2] University of Versailles, France
[3] Nanyang Technological University, Singapore

Abstract. Grøstl is one of the five finalist hash functions of the SHA-3 competition. For entering this final phase, the designers have tweaked the submitted versions. This tweak renders inapplicable the best known distinguishers on the compression function presented by Peyrin [18] that exploited the internal permutation properties. Since the beginning of the final round, very few analysis have been published on Grøstl. Currently, the best known rebound-based results on the permutation and the compression function for the 256-bit version work up to 8 rounds, and up to 7 rounds for the 512-bit version. In this paper, we present new rebound distinguishers that work on a higher number of rounds for the permutations of both 256 and 512-bit versions of this finalist, that is 9 and 10 respectively. Our distinguishers make use of an algorithm that we propose for solving three fully active states in the middle of the differential characteristic, while the Super-Sbox technique only handles two.

Keywords: Hash Function, Cryptanalysis, SHA-3, Grøstl, Rebound Attack.

1 Introduction

Hash functions are one of the main families in symmetric cryptography. They are functions that, given an input of variable length, produce an output of a fixed size. They have many important applications, like integrity check of executables, authentication, digital signatures.

Since 2005, several new attacks on hash functions have appeared. In particular, the hash standards MD5 and SHA-1 were cryptanalysed by Wang et al. [21,22]. Due to the resemblance of the standard SHA-2 with SHA-1, the confidence in the former has also been somewhat undermined. This is why the American National Institute of Standards and Technology (NIST) decided to launch in 2008 a competition for finding a new hash standard, SHA-3. This competition received 64 hash function submissions and accepted 51 to enter the first round. Now, three years and two rounds later, only 5 hash functions remain in the final phase of the competition.

Amongst these finalists, there is only one AES-based function, though many were proposed. This hash function is Grøstl [2], and is at the origin of the introduction of a new cryptanalysis technique that has been widely deployed, improved and applied to a large number of SHA-3 candidates, hash functions and other types of constructions. This new technique, called rebound attack, was introduced by Mendel et al. [11] and has become one of the most important tools used to analyze the security margin of many SHA-3 candidates as well as their building blocks. As for Grøstl itself, it has been applied and improved in several occasions [3, 12, 13, 15, 18]. Grøstl is undoubtedly one of the SHA-3 candidates that have received the largest amount of cryptanalysis. When entering the final round, a tweak of the function was proposed, which prevents the application of the attacks from [18]; we denote Grøstl-0 the original submission of the algorithm and Grøstl its tweaked version. Apart from the rebound results, the other main

[⋆] Supported by the French Agence Nationale de la Recherche through the SAPHIR2 project under Contract ANR-08-VERS-014.

[⋆⋆] Supported by the French *Délégation Générale pour l'Armement* (DGA).

[⋆⋆⋆] The author is supported by the Lee Kuan Yew Postdoctoral Fellowship 2011 and the Singapore National Research Foundation Fellowship 2012.

analysis communicated on Grøstl was at the presentation of [1] where a higher order property on 10 rounds of Grøstl-256 permutation with a complexity of 2^{509} was shown. In Table 1, we report a summary of the best known results on both 256 and 512-bit tweaked versions of Grøstl, including the ones that we will present in the following.

In this paper, we propose new results regarding both versions of the finalist Grøstl. First, on Grøstl-256, we provide the best known rebound distinguishers on 9 rounds of the permutation. From these results, we show how to make some nontrivial observations on the the compression function, providing the best known analysis on the compression function exploiting the properties of the internal permutations. For Grøstl-512, we considerably increase the number of analyzed rounds, from 7 to 10, providing the best analysis known on the permutation. Both results are obtained using rebound-like attack techniques and an algorithm that we introduce that allows to solve three fully active rounds in the middle of the differential characteristic with a much lower cost than a generic algorithm. Additionnally, we provide in Appendix A the direct application of our new techniques to the AES-based hash function PHOTON.

These results do not threaten the security of Grøstl, but we believe they will have an important role in better understanding Grøstl, and AES-based functions in general. In particular, we believe that our work will help determining the bounds and limits of rebound-like attacks in these types of constructions.

Target	**Subtarget**	**Rounds**	**Time**	**Memory**	**Ideal**	**Reference**
Grøstl-256	Permutation	8 (dist.)	2^{112}	2^{64}	2^{384}	[3]
		8 (dist.)	2^{48}	2^{8}	2^{96}	[19]
		9 (dist.)	2^{368}	2^{64}	2^{384}	Section 3
		10 (zero-sum)	2^{509}	–	2^{512}	[1]
Grøstl-512	Permutation	8 (dist.)	2^{280}	2^{64}	2^{448}	Section 4
		9 (dist.)	2^{328}	2^{64}	2^{384}	Section 4
		10 (dist.)	2^{392}	2^{64}	2^{448}	Section 4

Table 1: Best known analysis on the finalist Grøstl. By best analysis, we mean the ones on the highest number of rounds.

2 Generalities

2.1 Description of Grøstl

The hash function Grøstl-0 has been submitted to the SHA-3 competition under two different versions: Grøstl-0-256, which outputs a 256-bit digest and Grøstl-0-512 with a 512-bit fingerprint. For the final round of the competition, the candidate have been tweaked to Grøstl, with corresponding versions Grøstl-256 and Grøstl-512.

The Grøstl hash function handles arbitrary long messages by diving them into blocks after some padding and uses them to update iteratively an internal state (initialized to a predefined IV) with a compression function. This function is itself built upon two different permutations, namely P and Q. Each of those two permutations updates a large internal state using the well-understood wide-trail strategy of the AES. As an AES-like Substitution-Permutation Network, Grøstl enjoys a strong diffusion in each of the two permutations and by its wide-pipe design, the size of the internal states is ensured to be at least twice as large as the final digest.

The compression function f_{256} of `Grøstl-256` uses two permutations P_{256} and Q_{256}, which are similar to the two permutations P_{512} and Q_{512} used in the compression function f_{512} of `Grøstl-512`. More precisely, for a chaining value h and a message block m, the compression functions (Figure 1) produce the output ($\oplus$ denotes the `XOR` operation):

$$f_{256}(h,m) = P_{256}(h \oplus m) \oplus Q_{256}(m) \oplus h, \qquad \text{or:} \qquad f_{512}(h,m) = P_{512}(h \oplus m) \oplus Q_{512}(m) \oplus h.$$

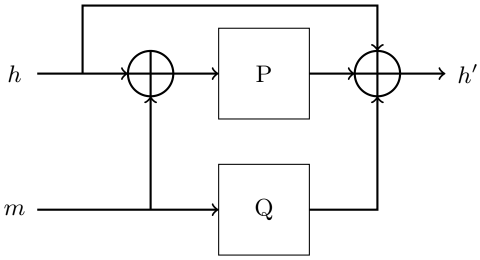


Figure 1: The compression function of `Grøstl` hash function using the two permutations P and Q.

The internal states are viewed as byte matrices of size 8×8 for the 256-bit version and 8×16 for the 512-bit one. The permutations strictly follow the design of the `AES` and are constructed as N_r iterations of the composition of four basic transformations:

$$R \overset{\text{def}}{:=} \textbf{MixBytes} \circ \textbf{ShiftBytes} \circ \textbf{SubBytes} \circ \textbf{AddRoundConstant}.$$

All the linear operations are performed in the same finite field $GF(2^8)$ as in the `AES`, defined via the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$ over $GF(2)$. The **AddRoundConstant** (`AC`) operation adds a predefined round-dependent constant, which significantly differs between P and Q to prevent the internal differential attack [18] taking advantage of the similarities in P and Q. The **SubBytes** (`SB`) layer is the non-linear layer of the round function R and applies the same SBox as in the `AES` to all the bytes of the internal state. The **ShiftBytes** (`Sh`) transformation shifts bytes in row i by $\tau_P[i]$ positions to the left for permutation P and $\tau_Q[i]$ positions for permutation Q. We note that τ also differs from P to Q to emphasize the asymmetry between the two permutations. Finally, the **MixBytes** (`Mb`) operation applies a maximum-distance separable (MDS) circulant constant matrix M independently to all the columns of the state. In `Grøstl-256`, $N_r = 10$, $\tau_P = [0,1,2,3,4,5,6,7]$ and $\tau_Q = [1,3,5,7,0,2,4,6]$, whereas for `Grøstl-512`, $N_r = 14$ and $\tau_P = [0,1,2,3,4,5,6,11]$ and $\tau_Q = [1,3,5,11,0,2,4,6]$.

Once all the message blocks of the padded input message have been processed by the compression function, a final output transformation is applied to the last chaining value h to produce the final n-bit hash value $h' = \text{trunc}_n(P(h) \oplus h)$, where trunc_n only keeps the last n bits.

2.2 Distinguishers

In this article, we will describe algorithms that find input pairs (X, X') for the permutation P (or the permutation Q), such that the input difference $\Delta_{IN} = X \oplus X'$ belongs to a subset of size IN and the output difference $\Delta_{OUT} = P(X) \oplus P(X')$ belongs to a subset of size OUT. The best known generic algorithm (this problem is different than the one studied in [8] where linear subspaces are considered) in order to solve this problem, known as limited-birthday problem, has been given in [3] and later a very close lower bound has been proven in [16]. For a randomly chosen n-bit permutation π, the generic algorithm can find such a pair with complexity

$\max\{\min\{\sqrt{2^n/IN}, \sqrt{2^n/OUT}\}, 2^n/(IN \cdot OUT)\}$. If one is able to describe an algorithm requiring less computation power, then we consider that a distinguisher exists on the permutation π.

In the case of `Grøstl`, it is also interesting to look at not only the internal permutations P and Q, but also the compression function f itself. For that matter, we will generate compression function input values (h, m) such that $\varDelta_{IN} = m \oplus h$ belongs to a subset of size IN, and such that $\varDelta_{IN} \oplus \varDelta_{OUT} = f(h, m) \oplus f(m, h) \oplus h \oplus m$ belongs to a subset of size OUT. Then, one can remark that:

$$\begin{aligned} f(h,m) \oplus f(m,h) &= P_{256}(h \oplus m) \oplus Q_{256}(m) \oplus P_{256}(m \oplus h) \oplus Q_{256}(h) \oplus h \oplus m, \\ f(h,m) \oplus f(m,h) &= Q_{256}(m) \oplus Q_{256}(h) \oplus h \oplus m, \\ f(h,m) \oplus f(m,h) \oplus h \oplus m &= Q_{256}(m) \oplus Q_{256}(h). \end{aligned}$$

Since the permutation Q is supposed to have no structural flaw, the best known generic algorithm requires $\max\{\min\{\sqrt{2^n/IN}, \sqrt{2^n/OUT}\}, 2^n/(IN \cdot OUT)\}$ operations (the situation is exactly the same as the permutation distinguisher with permutation Q) to find a pair (h, m) of inputs such that $h \oplus m \in IN$ and $f(h, m) \oplus f(m, h) \oplus h \oplus m \in OUT$. Note that both IN and OUT are specific to our attacks.

We emphasize that even if trivial distinguishers are already known for the `Grøstl` compression function (for example fixed-points), no distinguisher is known for the internal permutations. Moreover, our observations on the compression function use the differential properties of the internal permutations.

3 Distinguishers for reduced `Grøstl-256` internal permutations

In this section, we describe a distinguisher for the permutation P_{256} of the `Grøstl`-256 compression function reduced to 9 rounds. We emphasize that in the latest version of the `Grøstl` submission [20], the permutation Q_{256} has different coefficients in the **ShiftRows** transformation, but the technique we describe in the following applies to Q_{256} as well.

3.1 The truncated differential characteristic

In the following, we will consider truncated differential characteristics, originally introduced by Knudsen [7] for block cipher analysis. With this technique, already proven to be efficient for `AES`-based hash functions cryptanalysis [5, 6, 10, 17], the attacker only checks if there is a difference in a byte (active byte, denoted by a black square in the Figures) or not (inactive byte, denoted by an empty square in the Figures) without caring about the actual value of the difference.

The truncated differential characteristic we use has the sequence of active bytes

$$8 \xrightarrow{R_1} 1 \xrightarrow{R_2} 8 \xrightarrow{R_3} 64 \xrightarrow{R_4} 64 \xrightarrow{R_5} 64 \xrightarrow{R_6} 8 \xrightarrow{R_7} 1 \xrightarrow{R_8} 8 \xrightarrow{R_9} 64,$$

where the size in the input and output differences subsets are both $IN = OUT = 2^{8 \times 8} = 2^{64}$, since there are eight active bytes in each extreme state of the truncated characteristic. The actual truncated characteristic is reported in Appendix B.

Note that we have three fully active internal states in the middle of the differential characteristic, thus impossible to handle with the classical rebound or **SuperSBox** techniques.

3.2 Finding a conforming pair

The method to find a pair of inputs conforming to this truncated differential characteristic is similar to the rebound technique: we first find many solutions for the middle rounds (round 3 to round 6) and then we filter them out during the outwards probabilistic transitions through the **MixBytes** layers (round 2 and round 7). We denote $x \to y$ a non-null truncated differential transition mapping x active bytes to y active bytes in a column through a **MixBytes** (or **MixBytes**$^{-1}$) layer, and the MDS property ensures $x + y \geq 9$. Its differential probability is determined by the number $(8 - y)$ of inactive bytes on the output: $2^{-8(8-y)}$ if the MDS property is verified, 0 otherwise.

Therefore, since in our case we have two transitions $8 \to 1$ (see Figure 2), the outbound phase has a success probability of $\left(2^{-8\times 7}\right)^2 = 2^{-112}$ and is straightforward to handle once we found enough solutions for the inbound phase.

In order to find solutions for the middle rounds (see Figure 2), we propose an algorithm inspired by the ones in [14,15]: As in [3,8], instead of dealing with the classical 8-bit **SubBytes** SBoxes, one can consider 64-bit SBoxes (named **SuperSBoxes**) each composed of two AES SBox layers surrounding one **MixBytes** and one **AddRoundConstant** function[1]. Indeed, the **ShiftBytes** can be taken out from the **SuperSBoxes** since it commutes with **SubBytes**.

We start by choosing the input difference δ_{IN} after the first **SubBytes** layer in state S1 and the output difference δ_{OUT} after the last **MixBytes** layer in state S12 in a way that the truncated characteristic holds in S0 and S12. Note that since we have 8 active bytes in S1 and S12, there are as many as $2^{2\times 64} = 2^{128}$ different ways of choosing $(\delta_{IN}, \delta_{OUT})$. We continue by constructing the 8 forward **SuperSBox** independently by considering the 2^{64} possible input values for each of them in state S3: differences in S1 can be directly propagated to S3 since **MixBytes** is linear. This generates 8 independent lists, each of size 2^{64} and composed by paired values. Doing the same for the 8 backwards **SuperSBoxes** from state S12, we again get 8 independent lists of 2^{64} elements each, and we end up in state S8 where the 8 forward and the 8 backward lists overlap. In the sequel, we denote L_i the ith forward **SuperSBox** list and L'_i the ith backward one, for $1 \leq i \leq 8$.

In terms of freedom degrees in state S8, we want to merge 16 lists of 2^{64} elements each for a merging condition on $2 \times 512 = 1024$ bits (512 for values and 512 for differences): we then expect $2^{16\times 64}\, 2^{-1024} = 1$ solution as a result of the merging process. We detail a method in order to find this solution in time 2^{256} and memory 2^{64} (see Figure 3).

Step 1. We start by considering every possible combination of elements in each of the four lists L'_1, L'_2, L'_3 and L'_4. There are 2^{256} possibilities.

Step 2. This fully constraints 2×4 bytes in each of the 8 lists L_i, $1 \leq i \leq 8$ (i.e. the first 4 columns of the internal state). For each of them, we then expect $2^{64}\, 2^{-8\times 8} = 1$ element to match the randomized bytes. These elements can be found with one operation by sorting the lists L_i beforehand. At this point, note that the second half of the state S8 has been fully determined by the choice in $L_1, \ldots, L_8$.

Step 3. We now need to ensure that the 4 last lists L'_5, L'_6, L'_7 and L'_8 contain the elements imposed: those lists being of size 2^{64} each, this happens with probability $2^{64}\, 2^{-8\times(2\times 8)} = 2^{-64}$ independently on each list. Again, these elements can be found with one operation by sorting the lists L'_i beforehand.

[1] These **SuperSBoxes** are 64-bit large in the case of Grøstl, but only $4 \times 8 = 32$ bits for the AES.

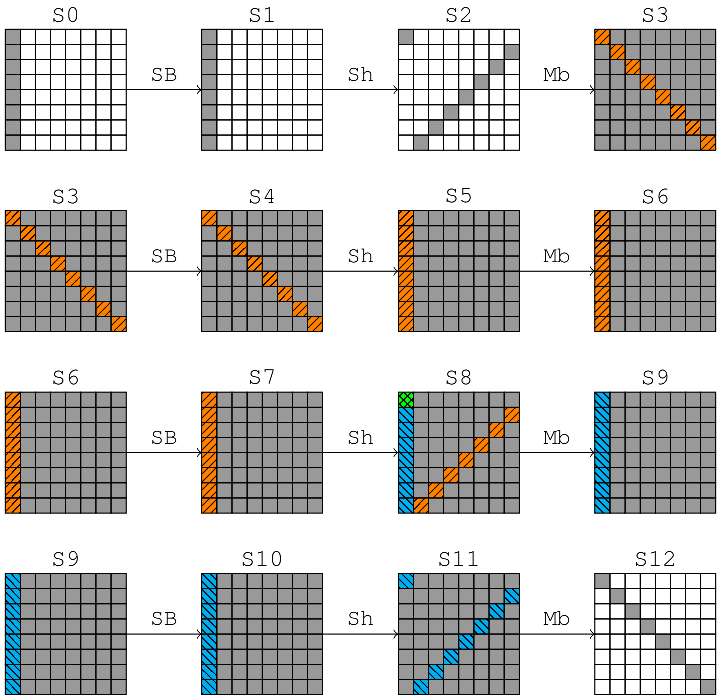
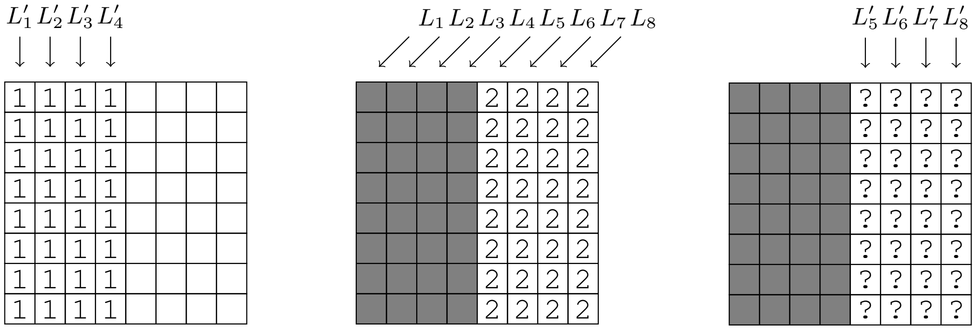


Figure 2: Inbound phase for the 9-round distinguisher attack on the Grøstl permutation P_{256}. The four rounds represented are the rounds 3 to 6 from the whole truncated differential characteristic. A gray byte indicates an active byte; hatched and coloured bytes emphasize one **SuperSBox**: there are seven similar others.


L'_1 L'_2 L'_3 L'_4
1 1 1 1
L_1 L_2 L_3 L_4 L_5 L_6 L_7 L_8
2 2 2 2
L'_5 L'_6 L'_7 L'_8
? ? ? ?

(a) Step 1. (b) Step 2. (c) Step 3.

Figure 3: Steps to merge the 16 lists. Grey cells denote bytes fully constrained by a choice of elements in $L'_1, \ldots, L'_4$ during the first step.

All in all, trying all the 2^{256} elements in (L'_1, L'_2, L'_3, L'_4), we expect to find $2^{256}\,2^{-64\times 4} = 1$ solution that will verify the 1024 bits of condition and we can find this solution with only a few operations.

Hence, from random differences $(\delta_{IN}, \delta_{OUT})$, we find a pair of internal states of the permutation that conforms to the middle rounds in time 2^{256} and memory 2^{64}. To pass the probabilistic transitions of the outbound phase, we need to repeat the merging 2^{112} times by picking another couple of differences $(\delta_{IN}, \delta_{OUT})$. In total, we find a pair of inputs to the permutation that conforms to the truncated differential characteristic in time complexity 2^{368} and memory complexity 2^{64}.

3.3 Comparison with ideal case

In the ideal case, obtaining a pair whose input and output differences lie in a subset of size $IN = OUT = 2^{64}$ for a 512-bit permutation requires 2^{384} computations: we can directly conclude that this leads to a distinguishing attack on the 9-round reduced version of the `Grøstl-256` permutation with 2^{368} computations and 2^{64} memory. Similarly, as explained in Section 2.2, this result also induces a nontrivial observation on the 9-round reduced version of the `Grøstl-256` compression function with identical complexity.

Finally, one can also derive slightly cheaper distinguishers by aiming less rounds: instead of using the 9-round truncated characteristic from Appendix B, it is possible to remove either round 2 or 8 and spare one 8 → 1 truncated differential transition. Overall, the generic complexity remains the same and this gives a distinguishing attack on the 8-round reduced version of the `Grøstl-256` permutation with 2^{312} computations and 2^{64} memory. Unfortunately, this is worse than previously known results.

4 Distinguishers for reduced `Grøstl-512` internal permutations

The 512-bit version of the `Grøstl` hash function uses a non-square 8×16 matrix as 1024-bit internal state, which therefore presents a lack of optimal diffusion: a single difference generates a fully active state after three rounds where a square-state would need only two. This enables us to add an extra round to the generalization of the regular 9-round characteristic of `AES`-like permutation (Section 3) to reach 10 rounds.

4.1 The truncated differential characteristic

To distinguish its permutation P_{512} [2] reduced to 10 rounds, we use the truncated differential characteristic with the sequence of active bytes

$$64 \xrightarrow{R_1} 8 \xrightarrow{R_2} 1 \xrightarrow{R_3} 8 \xrightarrow{R_4} 64 \xrightarrow{R_5} 128 \xrightarrow{R_6} 64 \xrightarrow{R_7} 8 \xrightarrow{R_8} 1 \xrightarrow{R_9} 8 \xrightarrow{R_{10}} 64.$$

where the size of the input differences subset is $IN = 2^{512}$ and the size of the output differences subset is $OUT = 2^{64}$.

The actual truncated characteristic is appended in Appendix C. Again, we split the characteristic into two parts: the inbound phase involving a merging of lists in the four middle rounds (round 4 to round 7), and an outbound phase that behaves as a probabilistic filter ensuring both $8 \longrightarrow 1$ transitions in the outward directions. Again, passing those two transitions with random values occurs with probability 2^{-112}.

4.2 Finding a conforming pair

In the following, we present an algorithm to solve the middle rounds in time 2^{280} and memory 2^{64}. In total, we will need to repeat this process 2^{112} times to get a pair of internal states that conforms to the whole truncated differential characteristic, which would then cost $2^{280+112} = 2^{392}$ in time and 2^{64} in memory. The strategy of this algorithm (see Figure 4) is similar to the ones presented in [14, 15] and the one from the previous section: we start by fixing the difference to a random value δ_{IN} in `S1` and δ_{OUT} in `S12` and linearly deduce the difference δ'_{IN} in `S3` and δ'_{OUT} in `S10`. Then, we construct the 32 lists corresponding to the 32 **SuperSBoxes**: the 16 forward **SuperSBoxes** have an input difference fixed to δ'_{IN} and cover states `S3` to `S8`, whereas the 16 backward **SuperSBoxes** spread over states `S10` to `S6` with an output difference fixed

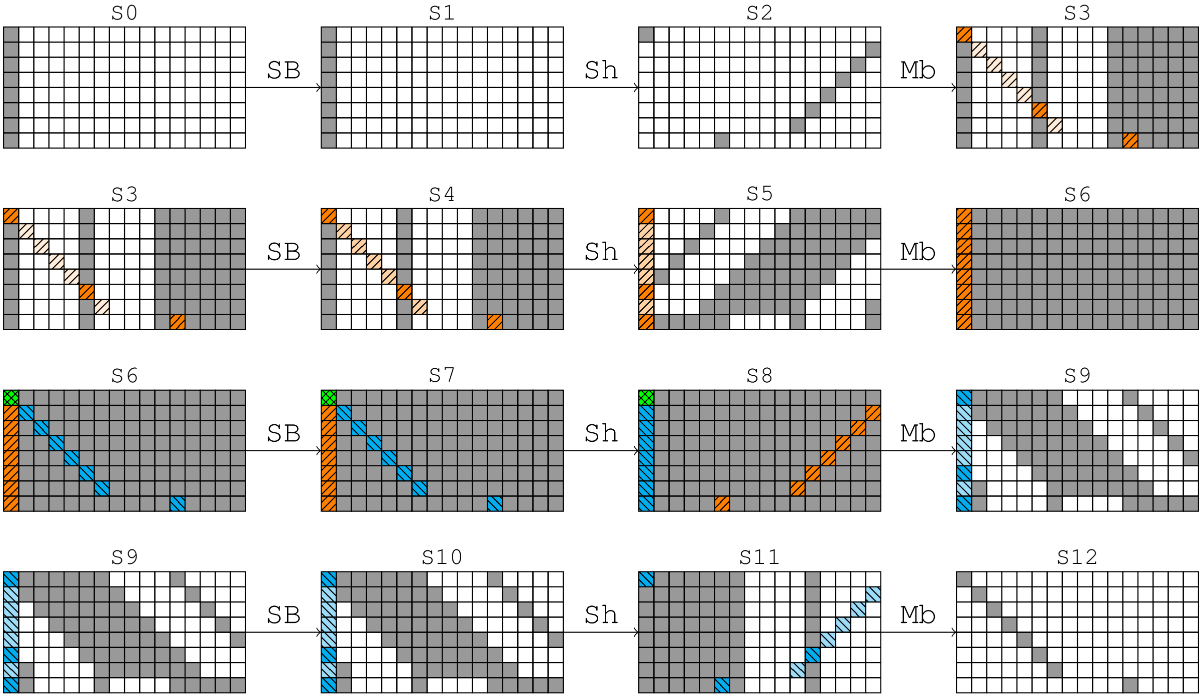


Figure 4: Inbound phase for the 10-round distinguisher attack on the Grøstl-512 permutation P_{512}. The four rounds represented are the rounds 4 to 7 from the whole truncated differential characteristic C. A gray byte indicates an active byte; hatched and coloured bytes emphasize the **SuperSBoxes**.

to δ'_{OUT}. In the sequel, we denote L_i the 16 forward **SuperSBoxes** and L'_i the backward ones, $1 \leq i \leq 16$.

The 32 lists overlap in S8, where we merge them on 2048 bits[3] to find $2^{64\times 32}\, 2^{-2048} = 1$ solution, since each list is of size 2^{64}. The naive way to find the solution would cost 2^{1024} in time by considering each element of the Cartesian product of the 16 lists L_i to check whether it satisfies the output 1024 bit difference condition. We describe now the algorithm that achieves the same goal in time 2^{280}.

First, we observe that due to the geometry of the non-square state, any list L_i intersects with only half of the L'_i. For instance, the first list L_1 associated to the first column of state S7 intersects with lists L'_1, L'_6, L'_{11}, L'_{12}, L'_{13}, L'_{14}, L'_{15} and L'_{16}. We represent this property with a 16×16 array on Figure 5: the 16 columns correspond to the 16 lists L'_i and the lines to the L_i, $1 \leq i \leq 16$. The cell (i, j) is white if and only if L_i has a non-null intersection with the list L'_j, otherwise it is gray.

Then, we note that the **MixBytes** transition between the states S8 and S9 constraints the differences in the lists L'_i : in the first column of S9 for example, only three bytes are active, so that the same column in S8 can only have $2^{3\times 8}$ different differences, which means that knowing three out of the eight differences in an element of L'_1 is enough to deduce the other five. For a column-vector of differences lying in a n-dimensional subspace, we can divide the 2^{64} elements of the associated lists in 2^{8n} disjointed sets of 2^{64-8n} values each. So, whenever we know the n independent differences, the only freedom that remains lie in the values. The bottom line of Figure 5 reports the subspace dimensions for each L'_i.

[2] It would work exactly the same way for the other permutation Q_{512}.
[3] The 2048 bits come from 1024 bits of values and 1024 bits of differences.

Using a guess-and-determine approach, we derive a way to use the previous facts to find the solution to the merge problem in time 2^{280}. As stated before, we expect only one solution; that is, we want to find a single element in each of the 32 lists. We start by guessing the values and the differences of the elements associated to the lists L'_2, L'_3, L'_4 and L'_5. For this, we will try all the possible combinations of their elements, there are $2^{4\times 64} = 2^{256}$ in total. For each one of the 2^{256} tries, all the checked cells ✓ now have known value and difference. From here, 8 bytes are known in each of the four lists L_5, L_6, L_7 and L_8: this imposes a 64-bit constraint on those lists, which filter out a single element in each. Thereby, we determined the value and difference in the other 16 bytes marked by ✓ in Figure 5. In lists L'_1 and L'_{16}, we have reached the maximum number of independent differences (three and two, respectively), so we can determine the differences for the other bytes of those columns: we mark them by ●. In L_4, the 8 constraints (three ✓ and two ●) filter out one element; then, we deduce the correct element in L_4 and mark it by ✓. We can now determine the differences in L'_{15} since the corresponding subspace has a dimension equals to two.

L'_i

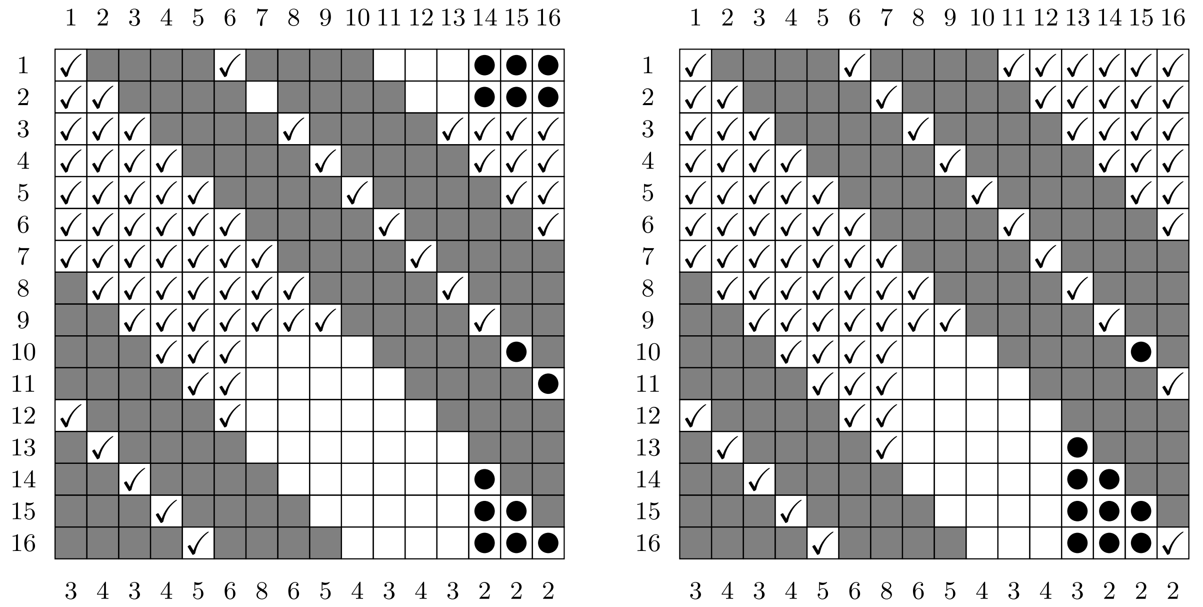
L_i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	●	▨	▨	▨	▨		▨	▨	▨	▨					●	●
2	●	✓	▨	▨	▨	▨		▨	▨	▨	▨				●	●
3	●	✓	✓	▨	▨	▨	▨		▨	▨	▨	▨			●	●
4	✓	✓	✓	✓	▨	▨	▨	▨	✓	▨	▨	▨	▨	✓	✓	✓
5	✓	✓	✓	✓	✓	▨	▨	▨	▨	✓	▨	▨	▨	▨	✓	✓
6	✓	✓	✓	✓	✓	✓	▨	▨	▨	▨	✓	▨	▨	▨	▨	✓
7	✓	✓	✓	✓	✓	✓	✓	▨	▨	▨	▨	✓	▨	▨	▨	▨
8	▨	✓	✓	✓	✓	✓	✓	✓	▨	▨	▨	▨	✓	▨	▨	▨
9	▨	▨	✓	✓	✓					▨	▨	▨	▨		▨	▨
10	▨	▨	▨	✓	✓						▨	▨	▨	▨	●	▨
11	▨	▨	▨	▨	✓							▨	▨	▨	▨	●
12	●	▨	▨	▨	▨								▨	▨	▨	▨
13	▨	✓	▨	▨	▨	▨								▨	▨	▨
14	▨	▨	✓	▨	▨	▨	▨								▨	▨
15	▨	▨	▨	✓	▨	▨	▨	▨							●	▨
16	▨	▨	▨	▨	✓	▨	▨	▨	▨						●	●
	3	4	3	4	5	6	8	6	5	4	3	4	3	2	2	2

Number of different differences in each L'_i

Figure 5: A ✓ means we know both value and difference for that byte, a ● means that we only determined the difference for that byte and white bytes are not constrained yet.

At this point, no more byte can be determined based on the information propagated so far. We continue by guessing the elements remaining in L'_6. Since there are already six byte-constraints on that list (three ✓), only 2^{16} elements conform to the conditions. The time complexity until now is thus $2^{256+16} = 2^{272}$.

Guessing the list L'_6 implies a 64-bit constraint of the list L_9 so that we get a single element out of it and determine four yet-unknown other bytes. This enables to learn the independent differences in L'_{14} and therefore, we filter an element from L_3 (two ✓ and four ●). At this stage, the list L'_1 is already fully constrained on its differences, so that we are left with a set of



(a) End of the second guess.

(b) Near the end.

Figure 6: A ✓ means we know both value and difference for that byte, a ● means that we only determined the difference for that byte and white bytes are not constrained yet.

$2^{64-3\times 8} = 2^{40}$ values constrained on five bytes (five ✓). Hence, we are able to determine all the unset values in L'_1 (Figure 6a).

Again, the lack of constraints prevent us to determine more bytes. We continue by guessing the 2^8 elements left in L_1 (two ✓ and three ●), which makes the time complexity increase to 2^{280}. The list L_1 being totally known, we derive the vector of differences in L'_{13}, which adds an extra byte-constraint on L_2 where only one element was left, and so fully determines it. From here, L'_7 becomes fully determined as well (four ✓) and so is L_{16}. In the latter, the differences being known, we were left with a set of $2^{64-2\times 8} = 2^{48}$ values, which are now constrained on six bytes (six ✓).

We describe in Figure 6b the knowledge propagated so far, with time complexity 2^{280} and probability 1. We observe that L_{10} is overdetermined (four ✓ and one ●) by one byte. This means that we get the correct value with probability 2^{-8}, whereas L_{11} is filtered with probability 1. Similarly, the element of L'_8 happens to be correctly defined with probability 2^{-16}; as for L'_9 and L'_{15}, with probability 1. We continue in L'_{11} by learning the full vector of differences, which constraints L_{12} on 11 bytes (five ✓ and one ●) so that we get a valid element with probability 2^{-24}. Finishing the guess and determine technique is done by filtering L'_{10} and L_{12} with probability 1, L_{16} with probability 2^{-40} and L_{13}, L_{14} and L_{15} with probability 2^{-64} each.

In total, for each guess, we successfully merge the 32 lists with probability

$$2^{-8-16-24-40-64-64-64} = 2^{-280},$$

but the whole procedure is repeated $2^{64\times 4+16+8} = 2^{280}$ times, so we expect to find the one existing solution. All in all, we described a way to do the merge with time complexity 2^{280} and memory complexity 2^{64}. The final complexity to find a valid candidate for the whole characteristic is then 2^{392} computations and 2^{64} memory.

4.3 Comparison with ideal case

In the ideal case, obtaining a pair whose input difference lies in a subset of size $IN = 2^{512}$ and whose output difference lies in a subset of size $OUT = 2^{64}$ for a 1024-bit permutation requires 2^{448} computations. We can directly conclude that this leads to a distinguishing attack on the 10-round reduced version of the Grøstl-512 permutation with 2^{392} computations and 2^{64} memory. Similarly, as explained in Section 2.2, this results also induces a nontrivial observation on the 10-round reduced version of the Grøstl-512 compression function with identical complexity.

One can also derive slightly cheaper distinguishers by aiming less rounds while keeping the same generic complexity: instead of using the 10-round truncated characteristic from Appendix C, it is possible to remove either round 3 or 9 and spare one $8 \rightarrow 1$ truncated differential transition. Overall, this gives a distinguishing attack on the 9-round reduced version of the Grøstl-512 permutation with 2^{336} computations and 2^{64} memory. By removing both rounds 3 and 9, we achieve 8 rounds with 2^{280} computations.

One can further gain another small factor for the 9-round case by using a $8 \rightarrow 2$ truncated differential transition instead of $8 \rightarrow 1$, for a final complexity of 2^{328} computations and 2^{64} memory. Indeed, the generic complexity drops to 2^{384} because we would now have $OUT = 2^{128}$.

5 Conclusion

In this paper, we have provided new and improved cryptanalysis results on the building blocks of both 256 and 512-bit versions of the finalist Grøstl. This is done by using a rebound-like approach as well as an algorithm that allows us to pass three fully active states in the middle of the differential characteristic with lower complexity than a general probabilistic approach. To the best of our knowledge, all previously known methods only manage to control two fully active states in the middle of the differential characteristic.

On Grøstl-256, we could provide the best known rebound distinguishers on 9 rounds of the permutation. For Grøstl-512, we have considerably increased the number of analyzed rounds, from 7 to 10, providing the best analysis known the permutation.

These results do not threaten the security of Grøstl, but we believe they will have an important role in better understanding AES-based functions in general. In particular, we believe that our work will help determining the bounds and limits of rebound-like attacks in these types of constructions. Future works could include the study of more AES-like functions in regards to this new cryptanalysis method.

References

1. Boura, C., Canteaut, A., Cannière, C.D.: Higher-order Differential Properties of Keccak and Luffa. In: FSE. Volume 6733 of LNCS., Springer (2011) 252–269
2. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl – a SHA-3 candidate
3. Gilbert, H., Peyrin, T.: Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations. In Hong, S., Iwata, T., eds.: FSE. Volume 6147 of Lecture Notes in Computer Science., Springer (2010) 365–383
4. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON Family of Lightweight Hash Functions. In Rogaway, P., ed.: CRYPTO. Volume 6841 of Lecture Notes in Computer Science., Springer (2011) 222–239
5. Jean, J., Fouque, P.A.: Practical Near-Collisions and Collisions on Round-Reduced ECHO-256 Compression Function. In Joux, A., ed.: FSE. Volume 6733 of Lecture Notes in Computer Science., Springer (2011) 107–127
6. Jean, J., Naya-Plasencia, M., Schläffer, M.: Improved Analysis of ECHO-256. In Miri, A., Vaudenay, S., eds.: Selected Areas in Cryptography. Volume 7118 of Lecture Notes in Computer Science., Springer (2011) 19–36
7. Knudsen, L.R.: Truncated and Higher Order Differentials. In Preneel, B., ed.: FSE. Volume 1008 of Lecture Notes in Computer Science., Springer (1994) 196–211

8. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound Distinguishers: Results on the Full Whirlpool Compression Function. [9] 126–143
9. Matsui, M., ed.: Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings. In Matsui, M., ed.: ASIACRYPT. Volume 5912 of Lecture Notes in Computer Science., Springer (2009)
10. Matusiewicz, K., Naya-Plasencia, M., Nikolic, I., Sasaki, Y., Schläffer, M.: Rebound Attack on the Full LANE Compression Function. [9] 106–125
11. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In: Fast Software Encryption - FSE 2009. Volume 1008 of Lecture Notes in Computer Science., Springer (5665)
12. Mendel, F., Peyrin, T., Rechberger, C., Schläffer, M.: Improved Cryptanalysis of the Reduced Grøstl Compression Function, ECHO Permutation and AES Block Cipher. In Jacobson, Jr., M.J., Rijmen, V., Safavi-Naini, R., eds.: Selected Areas in Cryptography. Volume 5867 of Lecture Notes in Computer Science., Springer (2009) 16–35
13. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Rebound Attacks on the Reduced Grøstl Hash Function. In Pieprzyk, J., ed.: CT-RSA. Volume 5985 of Lecture Notes in Computer Science., Springer (2010) 350–365
14. Naya-Plasencia, M.: How to Improve Rebound Attacks. Cryptology ePrint Archive, Report 2010/607 (2010) (extended version). urlhttp://eprint.iacr.org/.
15. Naya-Plasencia, M.: How to Improve Rebound Attacks. In: Advances in Cryptology: CRYPTO 2011. Volume 6841 of Lecture Notes in Computer Science., Springer (2011) 188–205
16. Nikolic, I., Pieprzyk, J., Sokolowski, P., Steinfeld, R.: Known and Chosen Key Differential Distinguishers for Block Ciphers. In Rhee, K.H., Nyang, D., eds.: ICISC. Volume 6829 of Lecture Notes in Computer Science., Springer (2010) 29–48
17. Peyrin, T.: Cryptanalysis of Grindahl. In Kurosawa, K., ed.: ASIACRYPT. Volume 4833 of Lecture Notes in Computer Science., Springer (2007) 551–567
18. Peyrin, T.: Improved Differential Attacks for ECHO and Grøstl. In Rabin, T., ed.: CRYPTO. Volume 6223 of Lecture Notes in Computer Science., Springer (2010) 370–392
19. Sasaki, Y., Li, Y., Wang, L., Sakiyama, K., Ohta, K.: Non-full-active Super-Sbox Analysis: Applications to ECHO and Grøstl. In Abe, M., ed.: ASIACRYPT. Volume 6477 of Lecture Notes in Computer Science., Springer (2010) 38–55
20. Schläffer, M.: Updated Differential Analysis of Grøstl. Grøstl website (January 2011)
21. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In Shoup, V., ed.: CRYPTO. Volume 3621 of Lecture Notes in Computer Science., Springer (2005) 17–36
22. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In Cramer, R., ed.: EUROCRYPT. Volume 3494 of Lecture Notes in Computer Science., Springer (2005) 19–35

A Distinguishers for other AES-like permutations

Using the same cryptanalysis technique, it is possible to study other AES-like schemes using permutations similar to the Grøstl ones. For example, the recent lightweigth hash function family PHOTON [4] is based on five different versions of AES-like permutations. We denote s the size of the cells ($s = 8$ for AES) and c the size of the square matrix representing the internal state ($c = 4$ for AES), the five versions (s, c) for PHOTON are then $(4, 5)$, $(4, 6)$, $(4, 7)$, $(4, 8)$ and $(8, 6)$ for increasing versions. All versions are defined to apply 12 rounds of an AES-like process, where the subkey additions are replaced by constant additions. Since the internal state is always square, by trivially adapting the method from Section 3 to the specific parameters of PHOTON, one can hope to obtain distinguishers for 9 rounds of the PHOTON internal permutations. However, we are able to do so only for the parameters $(4, 8)$ used in PHOTON-224/32/32 (see Table 2 with the comparison to previously known results). Indeed, the size c of the matrix plays an important role in the gap between the complexity of our algorithm and the generic one. The bigger is the matrix, the better will be the gap between the algorithm complexity and the generic one.

The same effect applies on AES in the known-key model, for which distinguishers on only 8 rounds are known as of today [3]. When attacking 9 rounds with the method from Section 3, the middle rounds will cost about 2^{64} operations per solution, while the two 4 → 1 truncated differential transitions during the outbound will be verified with probability $(2^{-24})^2 = 2^{-48}$.

Target	**Subtarget**	**Rounds**	**Time**	**Memory**	**Ideal**	**Reference**
PHOTON-224/32/32	permutation	8 (dist.)	2^8	2^4	2^{10}	[4]
		9 (dist.)	2^{184}	2^{32}	2^{192}	Section A

Table 2: Distinguishers on PHOTON internal permutation when applying the method from Section 3.

Overall, one solution for the whole characteristic is found with 2^{112} computation and 2^{32} memory, but the generic algorithm can find such a pair with only 2^{64}.

B 9-round Grøstl-256 permutation truncated characteristic

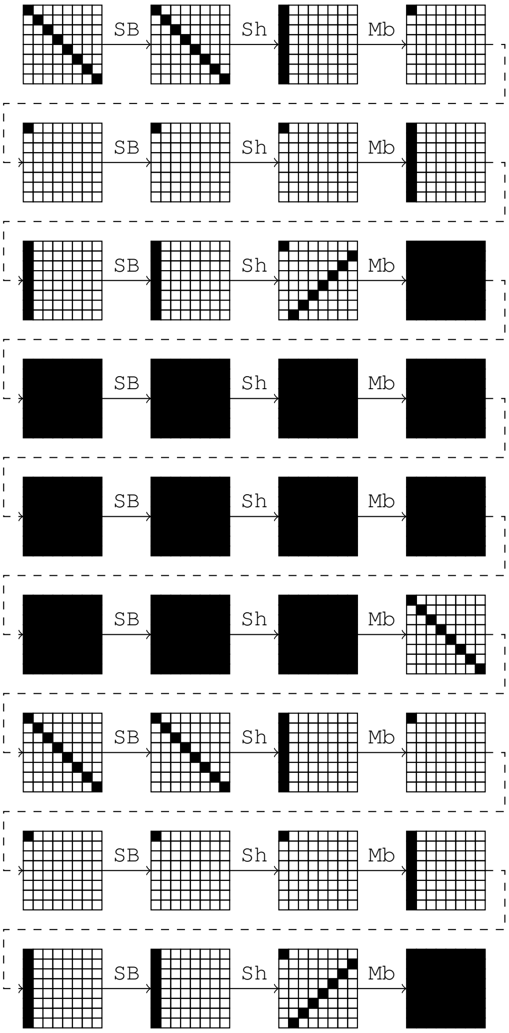


Figure 7: The 9-round truncated differential characteristic used to distinguish the permutation P of Grøstl-256 from an ideal permutation.

C 10-round Grøstl-512 permutation truncated characteristic

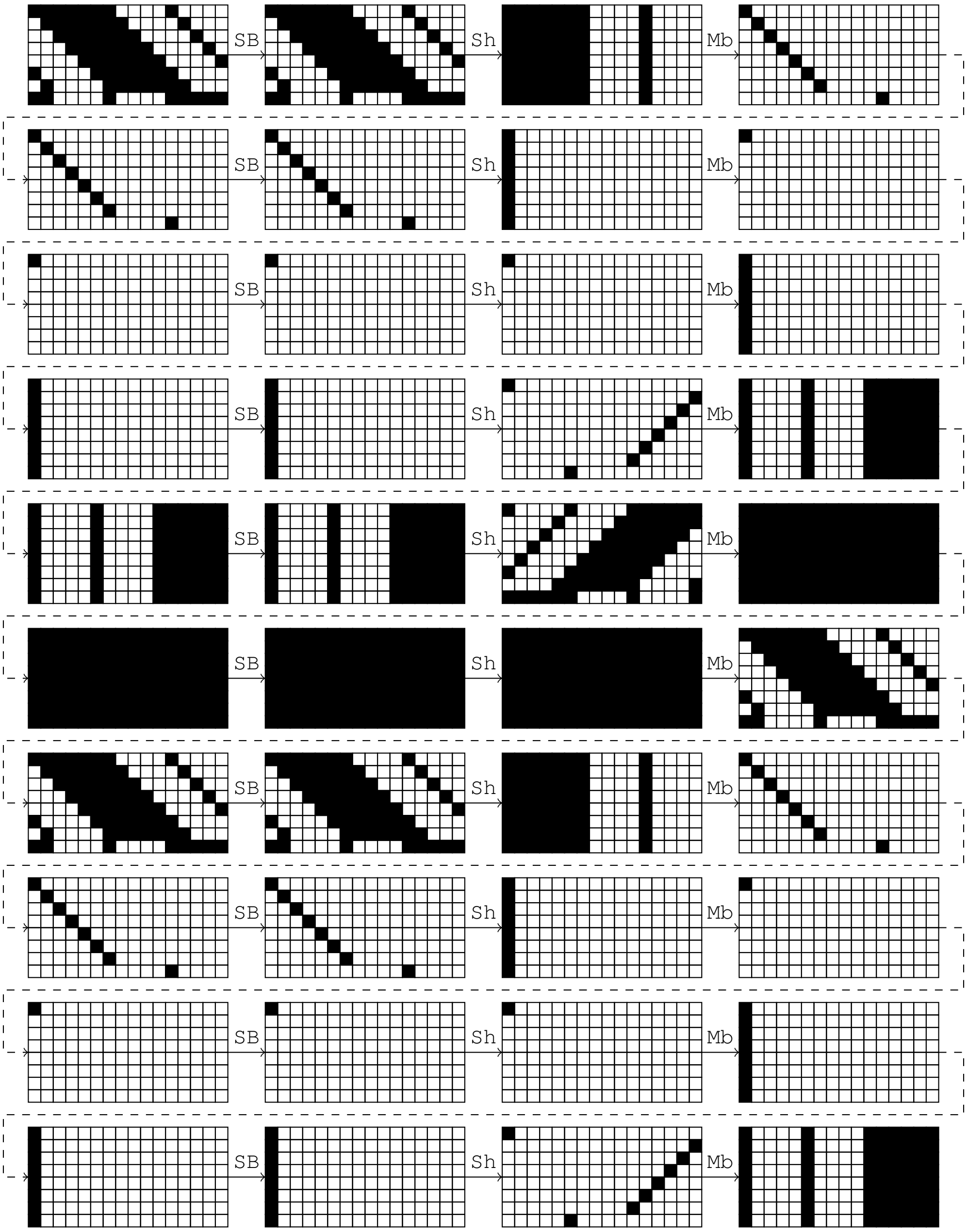


Figure 8: The 10-round truncated differential characteristic used to distinguish the permutation P of Grøstl-512 from an ideal permutation.

(Pseudo) Preimage Attack on Round-Reduced Grøstl Hash Function and Others

Shuang Wu[1], Dengguo Feng[1], Wenling Wu[1], Jian Guo[2], Le Dong[1], and Jian Zou[1]

[1] State Key Laboratory of Information Security, Institute of Software,
Chinese Academy of Sciences.
[2] Institute for Infocomm Research, Singapore.

wushuang@is.iscas.ac.cn

Abstract. The Grøstl hash function is one of the 5 final round candidates of the SHA-3 competition hosted by NIST. In this paper, we study the preimage resistance of the Grøstl hash function. We propose pseudo preimage attacks on Grøstl hash function for both 256-bit and 512-bit versions, *i.e.*, we need to choose the initial value in order to invert the hash function. Pseudo preimage attack on 5(out of 10)-round Grøstl-256 has a complexity of $(2^{244.85}, 2^{230.13})$ (in time and memory) and pseudo preimage attack on 8(out of 14)-round Grøstl-512 has a complexity of $(2^{507.32}, 2^{507.00})$. To the best of our knowledge, our attacks are the first (pseudo) preimage attacks on round-reduced Grøstl hash function, including its compression function and output transformation. These results are obtained by a variant of meet-in-the-middle preimage attack framework by Aoki and Sasaki. We also improve the time complexities of the preimage attacks against 5-round Whirlpool and 7-round AES hashes by Sasaki in FSE 2011.

Key words: hash function, meet-in-the-middle, preimage attack, Grøstl, Whirlpool, AES

1 Introduction

In FSE 2008, Gaëtan Leurent proposed the first preimage attack on the full MD4 hash function [12]. Based on this pioneering work, Aoki and Sasaki invented the technique of Meet-int-the-middle (MitM) preimage attack [2]. The basic idea of this technique is to divide the compression function into two concatenated sub-functions. The output values of two sub-functions can be independently calculated from the given input value in the forward direction and the backward direction. The steps of the forward and backward computation are called forward chunk and backward chunk. Then the MitM attack is applied to the output values of two sub-functions at the concatenating point of two chunks.

For hash functions based on block ciphers, the feedforward operations in the mode of operations like Davis-Meyer, Matyas-Meyer-Oseas and Miyaguchi-Preneel provide a chance for the applications of new technique called *splice-and-cut* [2]. The input and output of a compression function can be regarded as concatenated through the feed-forward operation in these modes of operations. Then the compression function is in the form of a circle and any step can be selected as either the starting point or the matching point.

Improvements have been developed on both the starting point and the matching point. The *initial structure* technique [16] (also called *message stealing* [7]) and the *local collision*[3] [15] technique allows two sub-functions to share several steps without violating the independency in computing their own values, which provides more attackable rounds. The *partial matching* technique [2, 16, 7, 1] takes advantage of the compression function's diffusion properties at the matching point. Due to slow diffusion of the Feistel-like round function, part of the state value can remain independent of the other chunk while proceeding with more reversed rounds. The

[3] The local collision technique is proposed by Joux et al. [5], which is originally used in the collision attacks. The similar idea can be used to construct the initial structure in the MitM preimage attack.

deterministic part of the state is used as the matching point. After finding a match of the partial values, the equality of the remaining part is calculated and checked. These techniques used in the MitM preimage attacks are illustrated in Fig. 1.

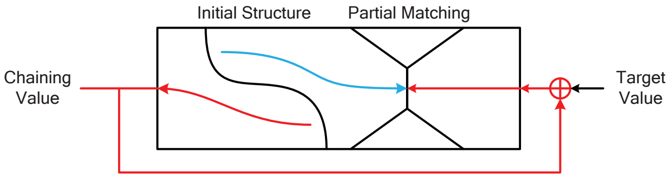


Fig. 1. Advanced techniques for MitM preimage attack

The MitM preimage attacks have been applied to full `HAVAL-3/4` [15], `MD4` [2, 7], `MD5` [16], `Tiger` [7], and round-reduced `HAS-160` [8], `RIPEMD` [21], `SHA-0/1` [3], `SHA-2` [7, 1]. The compression functions of these hash functions all use Feistel-like structures. In FSE 2011, Yu Sasaki proposed MitM preimage attack on `AES` hash mode for the first time [14]. He discussed how initial structure and partial matching can be used on `AES`-like structures and proposed direct applications to `AES` in different hash modes and round-reduced `Whirlpool` [4]. The development of the MitM attacks on hash functions has also inspired several attacks on block ciphers, such as `KTANTAN` [22] and `XTEA` [19].

Our contributions In this paper, we found a way to reduce the complexity of the MitM preimage attack on `AES`-like hash functions. By finding the optimal chunk separation with best balance between freedom degrees and the size of the matching point, the freedom degrees in the internal states are fully utilized.

`Grøstl` [6] is one of the five finalists in the third round of `SHA-3` [13] competition hosted by NIST. The `Grøstl` hash function has been tweaked in the third round. The original version is renamed to `Grøstl-0` and the tweaked version is called `Grøstl`.

We found that `Grøstl`'s round-reduced output transformation can be inverted using the MitM techniques. Then we noticed that if we can control the initial value, preimage of the output transformation can be connected with a compression function. The `Grøstl` hash function uses double-pipe chaining values, so we can actually match 2n-bit chaining value with a time complexity less than 2^n compression function calls. Since the initial value is chosen by us, this attack is a pseudo preimage attack.

The matching of double sized states is based on a method of variant generalized birthday attack. The special property of `Grøstl`'s compression function makes this approach possible. We found that the matching can be regarded as a special three-sum problem. Since the elements in one of the three sets can be restricted in a subspace, we can reduce the complexity to less than 2^n.

The comparison of previous best attacks and our attacks on `Grøstl` are shown in Table 1. Note that the attacks on `Grøstl-0` are not included in this table, since our attack is on the tweaked version.

We also improve the existing attacks against 5-round `Whirlpool` and 7-round `AES` hashing modes. While the previous result on 5-round `Whirlpool` applies to second preimage only, we improve the time complexity and also make the attack work for first preimages. We also improve the time complexity for the attacks against 7-round `AES` hashing modes. The details are presented in Appendix due to space limit.

Table 1. Comparison of the attacks on Grøstl-256 and Grøstl-512

Algorithm	Target	Attack Type	Rounds	Time	Memory	Source
Grøstl-256	Hash Function	Collision	3	2^{64}	-	[18]
	Compression Function	Semi-Free-Start Collision	6	2^{112}	2^{64}	[18]
	Permutation	Distinguisher	8	2^{48}	2^{8}	[17]
	Output Transformation	Preimage	5	2^{206}	2^{48}	Sect. 4.1
	Hash Function	Pseudo Preimage	5	$2^{244.85}$	$2^{230.13}$	Sect. 4
Grøstl-512	Hash Function	Collision	3	2^{192}	-	[18]
	Compression Function	Semi-Free-Start Collision	7	2^{152}	2^{56}	[17]
	Output Transformation	Preimage	8	2^{495}	2^{16}	Sect. 5.1
	Hash Function	Pseudo Preimage	8	$2^{507.32}$	$2^{507.00}$	Sect. 5

Outline of this paper In Sect. 2, we describe the specification of the Grøstl hash function. In Sect. 3, we introduce the attack outline of the pseudo preimage attack on reduced round Grøstl. Attacks on Grøstl-256 and Grøstl-512 are illustrated in Sect. 4 and Sect. 5 respectively. Sect. 6 is the conclusion.

2 Specification of Grøstl

Grøstl is a double-pipe design, i.e., the size of the chaining value ($2n$-bit) is twice as the hash size (n-bit). Message length should be less than 2^{64}. The padding rule is not introduced here, since it's not important in our attack.

The compression function of Grøstl is written as:

$$F(H, M) = P(H \oplus M) \oplus Q(M) \oplus H$$

Where H is the chaining value and M is the message block, both are of $2n$ bits. After all message blocks are processed, the last chaining value X is used as input of the output transformation, which is written as

$$\Omega(X) = Trunc_n(P(X) \oplus X)$$

The right half of $P(X) \oplus X$ is used as the hash value. The compression function and output transformation are illustrated in Fig. 2.

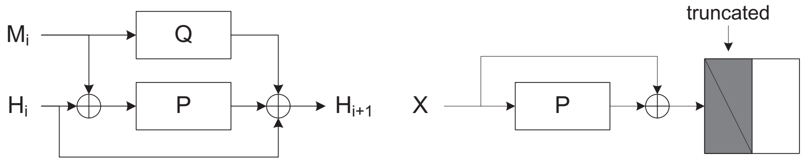


Fig. 2. Compression function and output transformation of Grøstl

P and Q are AES-like permutations with 8×8 and 8×16 sized state for Grøstl-256 and Grøstl-512 separately. Grøstl-256 uses 10-round P, Q and Grøstl-512 uses 14-round P, Q. The round function of the permutations consists of the four operations:

- SubBytes(SB): applies the Substitution-Box to each byte.
- ShiftBytes(SR): cyclically shifts the i-th row leftwards for i positions.
- MixBytes(MC): multiplies each column of the state matrix by an MDS matrix:

$$C = circ(02, 02, 03, 04, 05, 03, 05, 07)$$

- AddRoundConstant(AC): XOR the round constant to the state.

The shift vectors used in P and Q are different. P in Grøstl-256 uses (0,1,2,3,4,5,6,7) and P in Grøstl-512 uses (0,1,2,3,4,5,6,11). In the description of our attack, we skip Q's detail since it's not required.

An important property of the compression function has been pointed out in the submission document of Grøstl hash function [6]. Note that with $H' = H \oplus M$, the compression function can be written as

$$F(H, M) = P(H') \oplus H' \oplus Q(M) \oplus M.$$

So the generic preimage attack on the compression function with $2n$-bit state costs 2^n computations, since solving the equation $F(H, M) = T$ can be regarded as a birthday problem. Then the collision attack on the compression function costs $2^{2n/3}$ computations, since $F(H_1, M_1) \oplus F(H_2, M_2) = 0$ is a (four-sum) generalized birthday problem [20].

3 Outline of the Attack on the Grøstl Hash Function

Suppose the hash size is n-bit and the state size is $2n$-bit. In order to find a pseudo preimage (H, M) of the Grøstl hash function, let $X = F(H, M)$, then X is the preimage of the output transformation: $P(X) \oplus X = *||T$ where T is the target hash value and $*$ stands for arbitrary n-bit value. With $H' = H \oplus M$, we have

$$(P(H') \oplus H') \oplus (Q(M) \oplus M) \oplus X = 0 \qquad (1)$$

If we have collected enough candidates for $P(H') \oplus H'$, $Q(M) \oplus M$ and X, the pseudo preimage attack turns into a three-sum problem. As we know, there is no generic solution for three-sum problem faster than birthday attack. But if we can restrict $P(H') \oplus H'$ in a subspace, it is possible to break the birthday bound. Here we restrict $P(H') \oplus H'$ in a subspace by finding its partial zero preimages.

As illustrated in Fig. 3, the attack process is similar to the generalized birthday attack [20]. With four parameters x_1, x_2, x_3 and b, this attack can be described in four steps:

1. Find 2^{x_1} preimages X of the output transformation and store them in lookup table L_1.
2. Find 2^{x_3} H' such that leftmost b bits of $P(H') \oplus H'$ are all zero. Then store all $P(H') \oplus H'$ and H' in lookup table L_2. This step can be regarded as finding partial zero preimages on $P(H') \oplus H'$.
3. Choose 2^{x_2} random M with correct padding and calculate $Q(M) \oplus M$. Then check if there is an X in L_1 with the same leftmost b bits as $Q(M) \oplus M$. We expect to find $2^{x_1+x_2-b}$ partial matches $Q(M) \oplus M \oplus X$ here, whose left most b bits are all zero.
4. For each of the $2^{x_1+x_2-b}$ $Q(M) \oplus M \oplus X$ found in step 3, check if its remaining $(2n - b)$-bit value can be found in L_2.

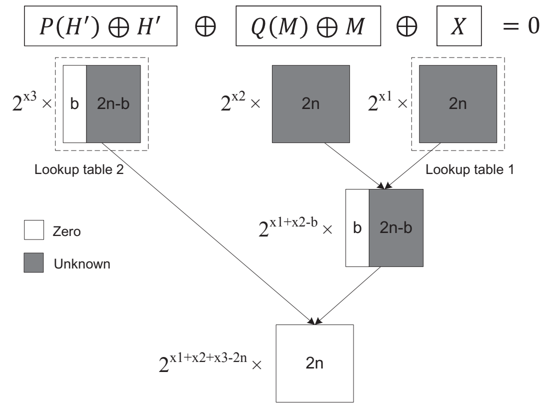


Fig. 3. Outline for pseudo preimage attack on the Grøstl hash function

Once a final match is found, we have H', M and X which satisfies equation (1). So, $(H' \oplus M, M)$ is a pseudo preimage of Grøstl.

Note that to find an X is to find an n-bit partial preimage of $P(X) \oplus X$ and the truncation bits are fixed (the leftmost n-bits are truncated). But for $P(H') \oplus H'$, it's not necessary to find partial preimage for the leftmost b bits. In fact, we can choose any b bits as the zero bits. We will further discuss the differences between fixed position and chosen position partial preimage attacks later.

Suppose that for Grøstl with $2n$-bit state, it takes $2^{C_1(2n,n)}$ computations to find a fixed position n-bit partial preimage and it takes $2^{C_2(2n,b)}$ computations to find a chosen position b-bit partial preimage of $P(X) \oplus X$. Now we calculate the complexity for each of the four attacking steps:

1. Step 1, building the look-up table 1 takes $2^{x_1+C_1(2n,n)}$ computations and 2^{x_1} memory.
2. Step 2, building the look-up table 2 takes $2^{x_3+C_2(2n,b)}$ computations and 2^{x_3} memory.
3. Step 3, calculating $Q(M) \oplus M$ for 2^{x_2} M and checking the partial match in table 1 takes 2^{x_2} Q calls, which is equivalent to 2^{x_2-1} compression function calls.
4. Step 4, checking the final match for $2^{x_1+x_2-b}$ candidates requires $2^{x_1+x_2-b}$ table look-ups, which can be equivalently regarded as $2^{x_1+x_2-b}C_{TL}$ compression function calls. C_{TL} is the complexity of one table lookup, where unit one is one compression function call. For 5-round Grøstl-256 and 8-round Grøstl-512(the attacked versions), C_{TL} is chosen as 1/640 and 1/2048 respectively[4].

Then the overall complexity is:

$$2^{x_1+C_1(2n,n)} + 2^{x_3+C_2(2n,b)} + 2^{x_2-1} + 2^{x_1+x_2-b} \cdot C_{TL} \tag{2}$$

with memory requirement of $2^{x_1} + 2^{x_3}$.

In the following sections, we first show how to find partial preimages of the function $P(X) \oplus X$ and calculate the complexity $C_1(2n, n)$ and $C_2(2n, b)$. Then we need to choose optimal parameters x_1, x_2, x_3 and b to minimize the complexity with the restriction of $x_1 + x_2 + x_3 \geq 2n$ and $0 \leq b \leq 2n$. Since in order to find one final match, we need $2^{x_1+x_2+x_3-2n} \geq 1 \Rightarrow x_1+x_2+x_3 \geq 2n$.

[4] The constant C_{TL} is chosen as the upper bound of the complexity that one table lookup takes, due to the fact that 5-round Grøstl-256 software implementation composes of $(8 * 8) * 5 * 2 = 640$ s-box lookups, and other operations. In 8-round Grostl-512, there are $(8 * 16) * 8 * 2 = 2048$ s-box lookups.

4 Pseudo Preimage Attack on 5-round `Grøstl-256`

In this section, first, we introduce the preimage attack on the output transformation, *i.e.*, the fixed position partial preimage attack on $P(X) \oplus X$ and calculate the complexity $C_1(512, 256)$. Then we introduce the chosen position partial preimage attack on $P(H') \oplus H'$ and give the expression of the function $f(b) = C_2(512, b)$. At last, we try to minimize the overall complexity by finding proper parameters for the generic attack introduced in Section 3.

4.1 Fixed Position Partial Preimage Attack on $P(X) \oplus X$

The chunk separation for this attack is shown in Fig. 4. Note that the yellow cells with a diagonal line are the truncated bytes, which can be regarded as free variables. In the last state of Fig. 4, the equations for the truncated byte can be directly removed since they are automatically fulfilled. The size of the full match is 256-bits for this MitM attack.

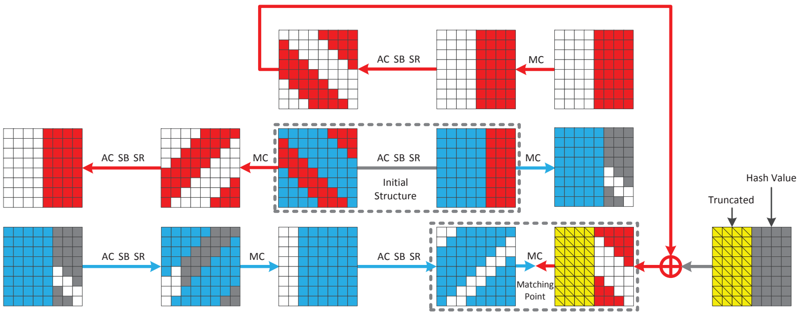


Fig. 4. Chunk separation of preimage attack on `Grøstl-256`'s output transformation

The Colors in the Chunk Separation First, we explain what the colors stand for. Actually, we use the same colors as in [14] to illustrate the chunk separations. The blue bytes in the forward chunk can be determined by the blue bytes in the initial structure. The white color in the forward chunk stands for the bytes whose values are affected by both red bytes and blue bytes in the initial structure, and can't be precomputed until the partial match is found. Similarly, in the backward chunk, red and white cells stand for the certain and uncertain bytes. The gray cells are constant bytes in the target value, the chaining value and the initial structure, which are known or can be chosen before the MitM attack.

Freedom Degrees and Size of the Matching Point Before we apply the MitM attack, we need to know the freedom degrees in the forward and backward directions and the bit size of the matching point. The calculation method has been explained in [14]. More details about this is in Appendix A.

With the method introduced in appendix A, we can find that , in Fig. 4, there are $D_2 = 2^{48}$ and $D_1 = 2^{64}$ freedom degrees in red and blue bytes respectively. In each of the four available columns, there are two bytes of matching point. So the size of the matching point is $m = 4 \times (2 \times 8) = 64$ bits.

The Attack Algorithm and Its Complexity In this section, we consider a generic MitM attack algorithm with partial matching technique. Suppose there are 2^{D_1} and 2^{D_2} freedom degrees in the forward and backward chunks. The size of the matching point is m-bit and the full matching size is b-bit. Without loss of generality, assume that $D_1 \geq D_2$. Note that if $D_1 + D_2 \geq b$, we can't fully use all the freedom degrees. Here we use d_1 and d_2 to denote the actually used freedom degrees:

$$(d_1, d_2) = \begin{cases} (D_1, D_2), & if D_1 + D_2 \leq b; \\ (b/2, b/2), & if D_1 + D_2 > b \ \ and \ \ D_2 \geq b/2; \\ (b - D_2, D_2), & if D_1 + D_2 > b \ \ and \ \ D_2 < b/2. \end{cases} \tag{3}$$

This MitM preimage attack can be described in four steps.

1. Choose random constants in the initial structure.
2. With the chosen constants, for all 2^{d_2} values v_j^2 of the forward direction, calculate all the partial values p_j^2 and the full values f_j^2 at the matching point and store all the pairs (v_j^2, p_j^2) in a look up table L;
3. For all 2^{d_1} values v_i^1 of the backward direction, calculate p_i^1. Then check if p_i^1 is in table L. If we found one partial match that $p_i^1 = p_j^2$ for some j, calculate the full value f_i^1 using v_i^1 and check if $f_i^1 = f_j^2$;
4. If no full match has been found yet, go to step 1.

Then we calculate the complexity. Step 2 costs 2^{d_2} f_2 calls and 2^{d_2} memory. Step 3 costs 2^{d_1} f_1 calls. Consider two kinds of circumstances separately.

– If $d_1 + d_2 \geq m$. After step 3 is done, we expect $2^{d_1+d_2-m}$ good candidates that satisfy the m-bit matching point. Now check if the full value of all good candidates are matched. This step requires $2^{d_1+d_2-m}$ computations. The probability that a good candidate is a full match is 2^{m-b}. Then the probability that there exists one full match in $2^{d_1+d_2-m}$ good candidates is about $2^{(d_1+d_2-m)+(m-b)} = 2^{d_1+d_2-b}$. So, we need to repeat the attack $2^{b-d_1-d_2}$ times in order to find a full match. The complexity is:

$$2^{b-d_1-d_2} \cdot (2^{d_1} + 2^{d_2} + 2^{d_1+d_2-m}) = 2^b \cdot (2^{-d_1} + 2^{-d_2} + 2^{-m})$$

– If $d_1 + d_2 < m$. After step 3 is done, we can find one good candidate with probability of $2^{d_1+d_2-m}$. So, we need to repeat the attack $2^{m-d_1-d_2}$ times to find one good candidate, then we calculate the full value of the good candidate at the matching point to check if it is a full match, which cost one computation. So the complexity to find one good candidate and check its full value is $2^{m-d_1-d_2}(2^{d_1} + 2^{d_2}) + 1$. Then find and check 2^{b-m} good candidates to get a full match. The complexity is:

$$2^{b-m} \cdot (2^{m-d_1-d_2}(2^{d_1} + 2^{d_2}) + 1) = 2^b \cdot (2^{-d_1} + 2^{-d_2} + 2^{-m})$$

So, no matter in which case, the complexity to find one full match using this algorithm is always

$$2^b \cdot (2^{-d_1} + 2^{-d_2} + 2^{-m}) \tag{4}$$

computations and 2^{d_2} memory.

Application to `Grøstl`'s Output Transformation In Fig. 4, the freedom degrees are $D_1 = 48, D_2 = 64$, the partial and full matching size are $m = 64$ and $b = 256$ bits. Using the attack algorithm introduced in Section 4.1, we can calculate the complexity to invert 5-round `Grøstl`'s output transformation. Here the complexity is measured by compression function calls. In the MitM attack it takes about half P calls, `i.e.` 1/4 compression function calls to evaluate the matching point for one direction. Thus we can multiply 2^{-2} to the complexity: $2^{C_1(512,256)} = 2^{-2} \cdot 2^{256}(2^{-64} + 2^{-48} + 2^{-64}) \approx 2^{206}$ compression function calls with 2^{48} memory.

On the Choice of the Chunk Separation We can prove that our chunk separation in Fig. 4 is optimal, which minimizes the complexity of inverting the output transformation.

Suppose there are b blue bytes and r red bytes in each column of the matching point. Then we show the relation between b, r, freedom degrees D_1, D_2 and the partial matching size m.

In the forward direction, r red bytes in one column of the matching point $\xrightarrow{AC,SB,SR,MC}$ r full red columns $\xrightarrow{AC,SB,SR}$ r red bytes in one column. Here we stops at the left end of the initial structure. In order to produce at least one byte of freedom degrees in the blue color, there are at least $r+1$ blue columns in the initial structure. Then there would be at most $8-(r+1)=7-r$ red columns in the initial structure.

In the backward direction, b blue bytes in one column of the matching point $\xrightarrow{SR^{-1},SB^{-1},AC^{-1}}$ $8-b$ white columns $\xrightarrow{MC^{-1},SR^{-1},SB^{-1},AC^{-1}}$ $8-b$ white bytes in each columns.

Now we count the freedom degrees. There are $(7-r)$ red columns in the initial structure and each column produces $8-b$ free bytes. So, freedom degrees in red color is $D_2 = 8(7-r)(8-b)$ bits. The minimum freedom degrees in the blue color here is $D_1 = 2^{64}$. Size of the matching point in one column is $8(b+r-8)$ bits, so there are $4 \times 8(b+r-8)$ bits of matching point in total.

So the complexity is $2^{-2} \cdot 2^{256}(2^{-64} + 2^{-8(7-r)(8-b)} + 2^{-32(b+r-8)})$. The minimum complexity is 2^{206} when $b=6, r=4$ or $b=5, r=5$. Fig. 4 is the case of $b=6, r=4$.

4.2 Chosen Position Partial Preimage Attack on $P(H') \oplus H'$

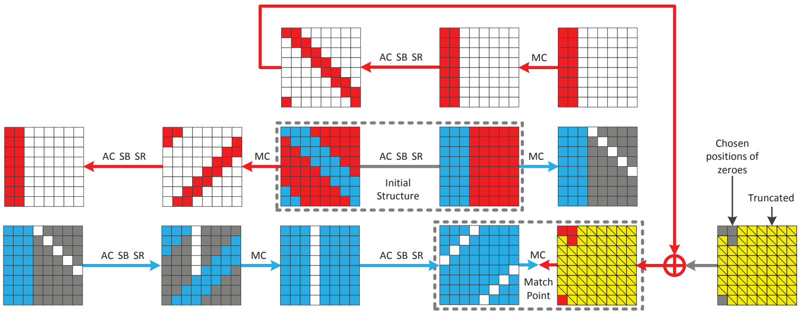


Fig. 5. Chunk separation of chosen position partial preimage attack on $P(H') \oplus H'$ for `Grøstl-256`

Now, consider the attack model of chosen position partial preimage. In the partial preimage attack of $P(H') \oplus H'$, we can choose the positions of the target bits. In order to minimize the complexity, we choose this chunk separation to maximize the size of the matching point $m(b)$ within all possible b target bits.

First, we discuss the size of matching point and chosen positions in one column. If less than 8 bits of the red byte in one column are chosen, no matching point can be derived. if $b > 8$ bits of the red bytes are chosen, as explained in appendix A, there are $b-8$ bits of matching point. Since there are only two red bytes in one column in the last state of Fig. 5, even if $b > 16$, no more than 8 bits of matching point can be derived. In order to maximize $m(b)$, we choose at most 2 red bytes in one column and then chose the red bytes from another column. When $b > 128$, $m(b) = 64$, because there are 64 bits of matching point in total. The graph of $m(b)$ is shown in Fig. 6.

In this Figure, freedom degrees in the red and blue color are $D_2 = 40$ and $D_1 = 64$. Then we can calculate the complexity of chosen position partial preimage:

$$2^{C_2(512,b)} = 2^{-2} \cdot 2^{b}(2^{-d_1} + 2^{-d_2} + 2^{-m(b)})$$

where d_1 and d_2 are chosen according to equation (3), `i.e.`:

$$(d_1, d_2) = \begin{cases} (64, 40), & if \quad b \geq 104; \\ (b-40, 40), & if \quad 80 \leq b < 104; \\ (b/2, b/2), & if \quad b < 80. \end{cases}$$

The graph of $C_2(512, b)$ is shown in Fig. 7. When $b > 80$, $C_2(512, b) \approx b - 42$.

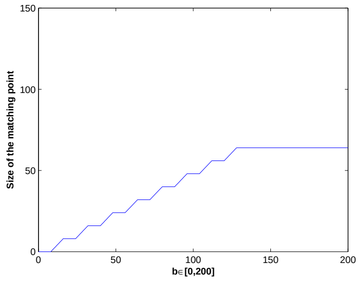


Fig. 6. Size of the matching point for chosen position truncations for `Grøstl-256`

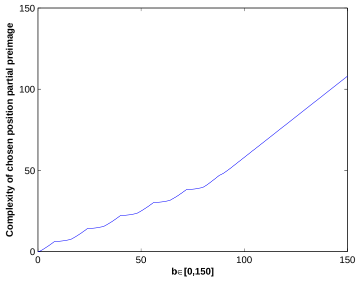


Fig. 7. Complexity of chosen position partial preimage of $P(H') \oplus H'$ for `Grøstl-256`

4.3 Minimizing the Overall Complexity

By now, we have found $C_1(512, 256)$ and $C_2(512, b)$. So we can start to deal with the overall complexity in equation (2). In the expression of the complexity, b can be integers from 0 to 512. For all $b \in [0, 512]$, optimal x_1, x_2 and x_3 are chosen to minimize the overall complexity. The graph of the minimum overall complexity for $b \in [0, 120]$ is shown in Fig. 8.

When $b = 31, x_1 \approx 36.93, x_2 \approx 244.93$ and $x_3 \approx 230.13$, the complexity is the lowest: $2^{244.85}$ compression function calls. Memory requirement is $2^{230.13}$. The chosen positions for the 31 bits $\approx$ 4 bytes are marked in Fig. 5.

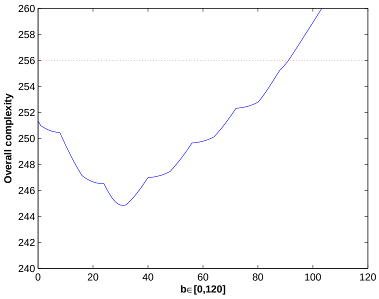


Fig. 8. Overall complexity of pseudo preimage attack on 5-round `Grøstl-256`

5 Pseudo Preimage Attack on 8-round `Grøstl-512`

The attack on `Grøstl-512` uses the same method for the three-sum phase as in the attack on `Grøstl-256`. Here we skip the details of the attack algorithm and introduce the difference between the attacks on them only.

5.1 Fixed Position Partial Preimage Attack on $P(X) \oplus X$

The chunk separation for 8-round `Grøstl-512` is shown in Fig. 9. Note that in this figure, we use a 2-round initial structure. Freedom degrees in the red and blue bytes are both 2^{16}. There are 4 bytes of matching point in total, as shown in Table 2 in Appendix B.

The parameters for the MitM preimage attack on the output transformation are $D_1 = D_2 = 16, m = 32$ and $n = b = 512$. So the complexity is $2^{C_1(1024,512)} = 2^{-2} \cdot 2^{512}(2^{-16} + 2^{-16} + 2^{-32}) \approx 2^{495}$ compression function calls and 2^{16} memory.

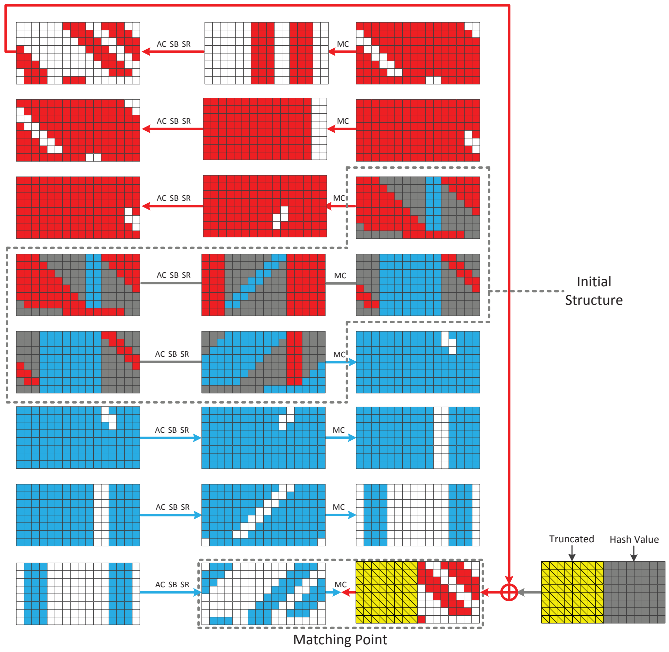


Fig. 9. Chunk separation of preimage attack on `Grøstl-512`'s output transformation

On the Choice of the Chunk Separation Actually, we searched for all the possible patterns of the chunk separation for 8-round `Grøstl-512`. The chunk separation in Fig. 9 is one of the best we found. The search algorithm is as follows:

Step 1. Search for the matching point.

We want to find good candidates in all the possible positions of the white columns in round 2 and round 6. Since there are 32 columns in two states, there are 2^{32} patterns in total.

For each of the pattern of white columns, we can calculate round 2 backward and round 6 forward and check if there are at least two byte of matching point. After the search for all the 2^{32} patterns, we found 1322 patterns with at least two bytes of matching point.

Step 2. Search for the initial structure.

Considering the mirror image and rotational similarity, there are only 120 distinct patterns in all the 1322 patterns of matching point. For each of the 120 patterns, we calculate forward from round 2 and backward from round 6.

If there is one white column in round 2, the number of possible patterns of the white bytes in the same column of round 3 is $2^8 - 1$, since there must be at least one white byte in this column. So size of the search space is $(2^8 - 1)^w$, where w is the number of white columns in both round 2 and round 6. In the 120 possible patterns, w is no more than 4, so the search space is at most $2^{32} \cdot 120 \approx 2^{39}$.

Using early-abort trick, we can directly skip some bad patterns in round 2 without knowing the pattern in round 6. Then the search space is reduced again and the search is practical.

5.2 Chosen Position Partial Preimage Attack on $P(H') \oplus H'$

For chosen position partial preimage, we use another chunk separation in Fig. 10.

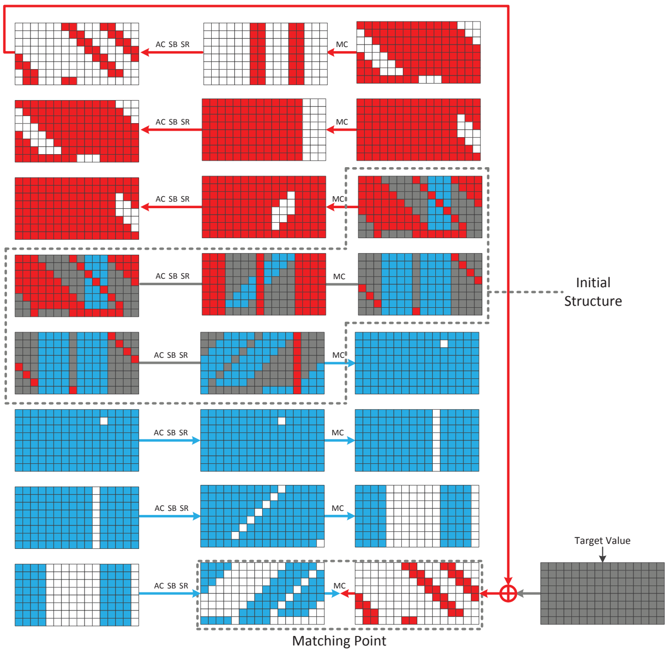


Fig. 10. Chunk separation of chosen position partial preimage attack on $P(H') \oplus H'$ for `Grøstl-512`

The freedom degrees for the MitM preimage attack are $D_1 = 24, D_2 = 8$. The distribution of the matching point bytes are shown in Table 3 and The graph of $m(b)$ is in Fig. 11. Then we can calculate the complexity of chosen position partial preimage:

$$2^{C_2(1024,b)} = 2^{-2} \cdot 2^b (2^{-d_1} + 2^{-d_2} + 2^{-m(b)})$$

where d_1 and d_2 are chosen according to equation (3), *i.e.*,

$$(d_1, d_2) = \begin{cases} (24, 8), & if \quad b \geq 32; \\ (b-8, 8), & if \quad 16 \leq b < 32; \\ (b/2, b/2), & if \quad b < 16. \end{cases}$$

The graphs for $m(b)$ and $C_2(1024, b)$ are in Fig. 11 and Fig. 12. The figures and tables are in Appendix B.

5.3 Minimizing the Overall Complexity

With the value and expression of $C_1(1024, 512)$ and $C_2(1024, b)$, we can deal with the overall complexity like we have done for Grøstl-256. The minimum overall complexity for different b is shown in Fig. 13.

When $b = 0, x_1 \approx 10.50, x_2 \approx 506.50$ and $x_3 \approx 507.00$, the overall complexity is the lowest: $2^{507.32}$. Memory requirement is $2^{507.00}$.

6 Conclusion

In this paper, we proposed pseudo preimage attacks on the hash functions of 5-round Grøstl-256 and 8-round Grøstl-512. This is the first pseudo preimage attack on round-reduced Grøstl hash function, which is a wide-pipe design.

In order to invert the wide-pipe hash function, we have to match $2n$-bit state value with less than 2^n computations. This is achieved by exploiting the special property of the Grøstl compression function. After collecting enough partial preimages on the component $P(X) \oplus X$, the double-sized state values are matched using a variant of the generalized birthday attack.

There is an interesting observation that this attack works with any function Q. Thus our attack can be applied to the Grøstl hash function with round-reduced permutation P and full-round permutation Q. However, our attacks do not threat any security claims of Grøstl.

Acknowledgement The authors would like to thank Kazumaro Aoki, Keting Jia, Mohammad Ali Orumiehchiha, Somitra Sanadhya, and Chunhua Su for their inspiring suggestions during the ASK 2011 workshop. The authors would also thank Lei Wang for useful discussions, and reviewers of FSE 2012 for helpful comments. This work is supported by the National Natural Science Foundation of China(No.60873259) and the Knowledge Innovation Project of The Chinese Academy of Sciences.

References

1. Kazumaro Aoki, Jian Guo, Krystian Matusiewicz, Yu Sasaki, and Lei Wang. Preimages for Step-Reduced SHA-2. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *LNCS*, pages 578–597. Springer, 2009.
2. Kazumaro Aoki and Yu Sasaki. Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography*, volume 5381 of *LNCS*, pages 103–119. Springer, 2008.
3. Kazumaro Aoki and Yu Sasaki. Meet-in-the-Middle Preimage Attacks Against Reduced SHA-0 and SHA-1. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *LNCS*, pages 70–89. Springer, 2009.
4. Paulo S.L.M. Barreto and Vincent Rijmen. The whirlpool hashing function. Submission to NESSIE, September 2000.
5. Florent Chabaud and Antoine Joux. Differential Collisions in SHA-0. In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *LNCS*, pages 56–71. Springer, 1998.
6. Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl – a SHA-3 candidate. Submission to NIST, 2008.
7. 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 Masayuki Abe, editor, *ASIACRYPT*, volume 6477 of *LNCS*, pages 56–75. Springer, 2010.
8. Deukjo Hong, Bonwook Koo, and Yu Sasaki. Improved Preimage Attack for 68-Step HAS-160. In Donghoon Lee and Seokhie Hong, editors, *ICISC*, volume 5984 of *LNCS*, pages 332–348. Springer, 2009.
9. Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *LNCS*, pages 306–316. Springer, 2004.
10. John Kelsey and Bruce Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 474–490. Springer, 2005.
11. Aggelos Kiayias, editor. *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, volume 6558 of *LNCS*. Springer, 2011.

12. Gaëtan Leurent. MD4 is Not One-Way. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *LNCS*, pages 412–428. Springer, 2008.
13. National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. Federal Register, 27(212):62212-62220, Nov. 2007. Available: `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf` (2008/10/17).
14. Yu Sasaki. Meet-in-the-Middle Preimage Attacks on AES Hashing Modes and an Application to Whirlpool. In Antoine Joux, editor, *FSE*, volume 6733 of *LNCS*, pages 378–396. Springer, 2011.
15. Yu Sasaki and Kazumaro Aoki. Preimage Attacks on 3, 4, and 5-Pass HAVAL. In Josef Pieprzyk, editor, *ASIACRYPT*, volume 5350 of *LNCS*, pages 253–271. Springer, 2008.
16. Yu Sasaki and Kazumaro Aoki. Finding Preimages in Full MD5 Faster Than Exhaustive Search. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *LNCS*, pages 134–152. Springer, 2009.
17. Yu Sasaki, Yang Li, Lei Wang, Kazuo Sakiyama, and Kazuo Ohta. New Non-Ideal Properties of AES-Based Permutations: Applications to ECHO and Grøstl. In *ASIACRYPT*, volume 6477 of *LNCS*, pages 38–55. Springer, 2010.
18. Martin Schläffer. Updated Differential Analysis of Grøstl. Grøstl website, January 2011.
19. Gautham Sekar, Nicky Mouha, Vesselin Velichkov, and Bart Preneel. Meet-in-the-Middle Attacks on Reduced-Round XTEA. In Kiayias [11], pages 250–267.
20. David Wagner. A Generalized Birthday Problem. In Moti Yung, editor, *CRYPTO*, volume 2442 of *LNCS*, pages 288–303. Springer, 2002.
21. Lei Wang, Yu Sasaki, Wataru Komatsubara, Kazuo Ohta, and Kazuo Sakiyama. (Second) Preimage Attacks on Step-Reduced RIPEMD/RIPEMD-128 with a New Local-Collision Approach. In Kiayias [11], pages 197–212.
22. Lei Wei, Christian Rechberger, Jian Guo, Hongjun Wu, Huaxiong Wang, and San Ling. Improved Meet-in-the-Middle Cryptanalysis of KTANTAN (Poster). In Udaya Parampalli and Philip Hawkes, editors, *ACISP*, volume 6812 of *LNCS*, pages 433–438. Springer, 2011.

A Calculation of Freedom Degrees and Size of the Matching Point

Calculating Freedom Degrees Compared to original attack, we have less constants in the initial structure. In Fig. 4, there is no constant in the initial structure. Before the MC operation that produces uncertain (white) bytes in the forward chunk, there are 24 red bytes. In order to maintain 18 constant (gray) bytes after the MC operation, it is equivalent to solve such a equation group with 24 variables and 18 equations:

$$C \cdot \begin{pmatrix} r_0 & r_8 & r_{16} \\ r_1 & r_9 & r_{17} \\ r_2 & r_{10} & r_{18} \\ r_3 & r_{11} & r_{19} \\ r_4 & r_{12} & r_{20} \\ r_5 & r_{13} & r_{21} \\ r_6 & r_{14} & r_{22} \\ r_7 & r_{15} & r_{23} \end{pmatrix} = \begin{pmatrix} c_0 & c_8 & c_{16} \\ c_1 & c_9 & c_{17} \\ c_2 & c_{10} & c_{18} \\ c_3 & c_{11} & c_{19} \\ * & c_{12} & c_{20} \\ * & * & c_{21} \\ c_6 & * & * \\ c_7 & c_{15} & * \end{pmatrix} \tag{5}$$

where C is the MDS matrix, $\{r_i\}$ are the values of red bytes, $\{c_i\}$ are constants and $*$ are arbitrary values we don't care. This equation group has $2^{8\cdot(24-18)} = 2^{48}$ solutions, which are the freedom degrees in the red bytes, `i.e.` the backward chunk. Similarly, by observing 40 variables and 32 equations, the freedom degrees in the blue bytes can be calculated as $2^{8\cdot(40-32)} = 2^{64}$.

Calculating Size of the Matching Point First, we need to explain how the partial matching through MR operation works. Use the first column of the matching point in Fig. 4 as an example, our target is to find proper values that satisfies the following equation:

$$C \cdot \begin{pmatrix} x_0 \; x_1 \; x_2 \; x_3 \; \underline{x_4} \; \underline{x_5} \; x_6 \; x_7 \end{pmatrix}^T = \begin{pmatrix} y_0 \; \underline{y_1} \; \underline{y_2} \; \underline{y_3} \; \underline{y_4} \; y_5 \; y_6 \; y_7 \end{pmatrix}^T \tag{6}$$

where $x_0, x_1, x_2, x_3, x_6, x_7, y_0, y_5, y_6, y_7$ are known bytes and $\underline{x_4}, \underline{x_5}, \underline{y_1}, \underline{y_2}, \underline{y_3}, \underline{y_4}$ are uncertain bytes. So only four equations of y_0, y_5, y_6 and y_7 are useful to us:

$$05\underline{x_4} + 03\underline{x_5} = 02x_0 + 02x_1 + 03x_2 + 04x_3 + 05x_6 + 07x_7 + y_0 \tag{7}$$

$$07\underline{x_4} + 02\underline{x_5} = 04x_0 + 05x_1 + 03x_2 + 05x_3 + 02x_6 + 03x_7 + y_5 \tag{8}$$

$$05\underline{x_4} + 07\underline{x_5} = 03x_0 + 04x_1 + 05x_2 + 03x_3 + 02x_6 + 02x_7 + y_6 \tag{9}$$

$$03\underline{x_4} + 05\underline{x_5} = 02x_0 + 03x_1 + 04x_2 + 05x_3 + 07x_6 + 02x_7 + y_7 \tag{10}$$

From equations (7)(8), obtain $\underline{x_4}, \underline{x_5}$ with linear combinations of the known bytes:

$$\underline{x_4} = F7x_0 + A5x_1 + 00x_2 + 52x_3 + A5x_6 + 53x_7 + 52y_0 + 52y_5$$

$$\underline{x_5} = F6x_0 + A4x_1 + 8Cx_2 + 50x_3 + 2Ax_6 + DDx_7 + DFy_0 + 52y_5$$

Then equation (10) can be rewritten as:

$$F1x_0 + 52x_1 + 8Cx_2 + A9x_3 + D3x_6 + 23x_7 = 2Ay_0 + A4y_5 + y_6 \tag{11}$$

$$03x_0 + F5x_1 + 8Ex_2 + F8x_3 + 71x_6 + 73x_7 = 78y_0 + F7y_5 + y_7 \tag{12}$$

Equations (11) and (12) are used as the matching point, since they provide equations of the known bytes that can be pre-computed, stored separately and then checked using table look-ups. Here the matching point is not directly truncated from the state value. In Fig. 4, two bytes of matching point can be derived from one column. At most 8 bytes (64 bits) of matching point can be found, since there are only four available columns.

Suppose there are b and r known bytes in one column of the input and output values of the MC at the matching point. $8 - b$ uncertain bytes are regarded as variables and r known red bytes can provide r equations. $8 - b$ of the equations are used to determine values of the variables. Then the remaining $r - (8 - b) = r + b - 8$ equations are on the known bytes, which are regarded as the matching point. Note that if $b + r \leq 8$, no matching point can be derived from this column. Otherwise, there are $b + r - 8$ bytes of matching point in this column.

If size of the known bits b' and r' are not exact multiples of 8, we can further split the linear equations on bytes into linear equations on bits. The bit size of matching point can be calculated as $b' + r' - 64$.

B Figures and Tables for Grøstl-512

Table 2. The matching point in Fig. 9

column index	9	10	11	12	13	14	15	16
blue bytes	4	3	3	4	5	5	4	3
red bytes	4	3	3	4	5	5	4	3
sum	8	6	6	8	10	10	8	6
matching point(bytes)	0	0	0	0	2	2	0	0

C Preimage Attack on round-reduced Whirlpool

C.1 Specification of Whirlpool

Whirlpool uses MD-strengthening structure, with narrow-pipe chaining value and no block counters. So it is vulnerable to generic attack, like the expandable messages [10] and multi-target pseudo preimage [12] attack. We will talk about the details later.

Table 3. matching points in Fig. 10

column index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
blue bytes	5	4	3	2	0	2	3	4	5	4	5	6	6	6	5	4
red bytes	2	3	2	0	0	0	2	3	2	2	2	3	4	3	2	2
sum	7	7	5	2	0	2	5	7	7	6	7	9	10	9	7	6
matching point(bytes)	0	0	0	0	0	0	0	0	0	0	0	1	2	1	0	0

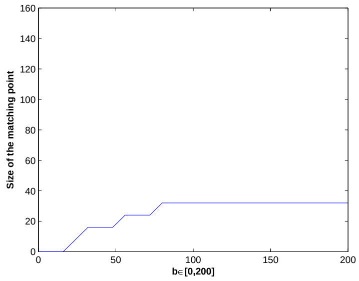


Fig. 11. Size of the matching point for chosen position truncations for `Grøstl-512`

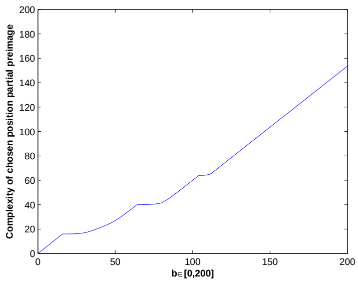


Fig. 12. Complexity of chosen position partial preimage of $P(H') \oplus H'$ for `Grøstl-512`

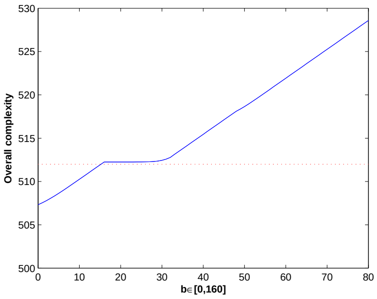


Fig. 13. Overall complexity of pseudo preimage attack on 8-round `Grøstl-512`

`Whirlpool` accepts any message with less than 2^{256} bits as input and the 256-bit binary expression of bit length is padded according to MD-strengthening, `i.e.` $M||1||0^*||length$. Size of the message block, the chaining value and the hash value is 512-bit.

Compression function of `Whirlpool` can be regarded as a block cipher called W in Miyaguchi-Preneel mode.

$$F(H, M) = W_H(M) \oplus M \oplus H$$

where block cipher W use `AES`-like iteration with 8×8 state of bytes and the $(8i+j)$-th input byte of the message block is placed at the i-th row and j-th column of the state. Each round consists of four operations:

- SubBytes(SB): applies the Substitution-Box to each byte.
- ShiftColumns(SC): cyclically shift the i-th column downwards for i positions.
- MixRows(MR): multiply each row of the state matrix by an MDS matrix

$$C = circ(01, 01, 04, 01, 08, 05, 02, 09)$$

- AddRoundKey(AK): XOR the round key to the state.

Since the key schedule is not important in our attack, the description is omitted.

C.2 Improved Second Preimage Attack on `Whirlpool`

In [14], Yu Sasaki proposed a second preimage attack on 5-round `Whirlpool` using the MitM approach. In their attack, there are only 2^8 freedom degrees in both chunks, but the size of matching point is much larger (40 bytes=320 bits). The comparison of the preimage attacks on `Whirlpool` is shown in Table 4.

Table 4. Comparison of the preimage attacks on `Whirlpool`

Attack Type	Rounds	Time	Memory	Source
Second Preimage	5	2^{504}	2^8	[14]
Second Preimage	5	2^{448}	2^{64}	this section
Preimage	5	$2^{481.5}$	2^{64}	this section

In this section, we propose an improved chunk separation with more freedom degrees and a smaller matching point in Fig. 14.

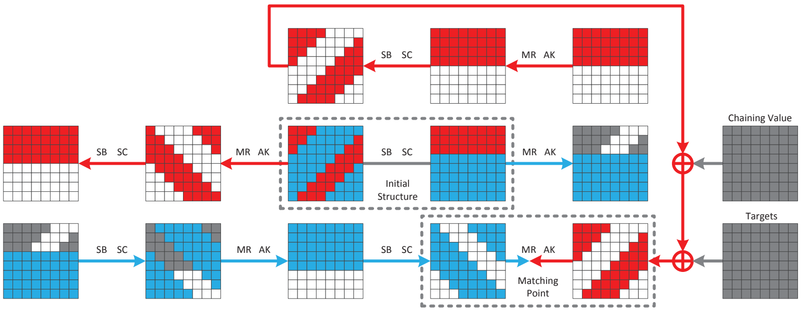


Fig. 14. Chunk separation for improved 2nd-preimage attack on 5-round `Whirlpool`

We use the same colors as in [14] to illustrate the chunk separations. The blue bytes in the forward chunk can be determined by the previously chosen blue bytes in the initial structure.

The white bytes in the forward chunk stands for the bytes whose value are affected by the red bytes from initial structure and can't be precomputed until the partial match is found. Similarly, in the backward chunk, red and white cells stand for the certain and uncertain bytes. The gray cells are constant bytes in the target value, the chaining value and the initial structure.

Since this is a second preimage attack, the second last chaining value and the last message block with proper padding are known. We choose random messages and get a random chaining value at the third last position. With this chaining value, apply MitM preimage attack of the compression function.

With chunk separation in Fig. 14, we have a MitM attack with $D_1 = 72, D_2 = 64, m = 64$ and $b = 512$. According to equation 4, the complexity can be computed as $2^{-1}2^{512}(2^{-72} + 2^{-64} + 2^{-64}) \approx 2^{448}$. Memory requirement is 2^{64}. Note that the complexity of computing the two chunks and checking the full match may be different. Here, we don't consider the difference between them and they are all regarded as the same cost of half compression function call. Methods for calculating freedom degrees of two chunks and the size of matching point are described in appendix A, which can also be found in [14].

C.3 First Preimage Attack on `Whirlpool`

This attack consists of three steps: First, find a preimage of the last block with proper padding. Second, construct an expandable message. At last, connect expandable message and the last block with MitM. The attack process is illustrated in Fig. 15.

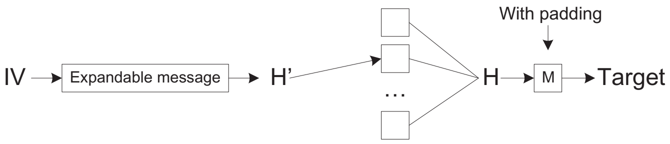


Fig. 15. Outline of the first preimage attack on 5-round `Whirlpool`

Dealing with Message Padding In order to apply the first preimage attack, the message padding must be dealt with properly. In our attack, the last message block consists of 255-bit message concatenated with one bit of "1" padding and 256-bit binary expression of the message length l. Since `Whirlpool` uses 512-bit message block, $l \equiv 255 \quad mod \quad 512$. Then the last 9 bits of l are fixed to 011111111.

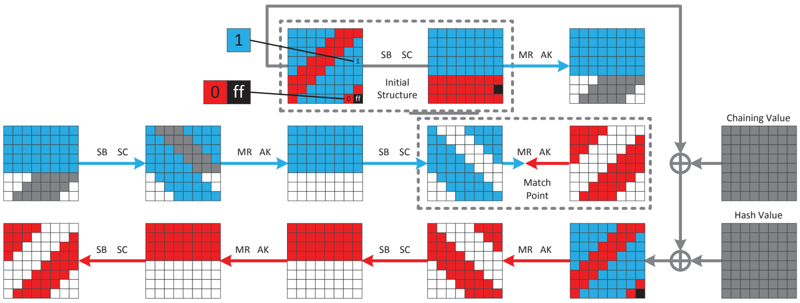


Fig. 16. Chunk separation for the last message block of `Whirlpool`

In Fig. 16, the initial structure is relocated at the beginning of the compression function for the convenience of the message padding, since in MP mode, the first state is the message block itself. Value of the black byte in the first state is fixed to $0xff$, because it is the last 8 bits of l. One red byte is marked with a "0", which means the last bit of it is fixed to zero due to the message length l. There is another blue byte marked with a "1", which comes from the "1-0" padding.

Parameters for this MitM attack are $D_1 = D_2 = 63, m = 64$ and $b = 512$. According to equation 4, the complexity is about 2^{449} computations and 2^{63} memory for the last block. When the attack on the last block is done, the remaining bits of the message length are fulfilled by expandable messages.

Expandable Messages Expandable messages [10] can be constructed using either Joux's multi-collision [9] or fix points of the compression function.

Expandable 2^k-collision can be constructed with $k \cdot 2^{n/2}$ computations and k memory. But its length can only be in the range of $[k, k + 2^k - 1]$ blocks. If the message length obtained from the last block is less than k (with a very small probability), we choose different random constants and repeat the attack.

Fix points of MP mode can be constructed by finding the zero preimages of the compression function in MMO mode, since

$$W_H(M) \oplus M \oplus H = H \Leftrightarrow W_H(M) \oplus M = 0.$$

This can be done using the same technique as in our 2nd-preimage attack, with complexity of $(2^{449}, 2^{64})$, which is an affordable cost for us. Note that for random H, the fix point exists with probability of $1 - e^{-1}$. If no fix point can be found for IV, we choose a random message block, compute the following chaining value and try to find fix point for this chaining value instead.

So, either way is fine to construct the expandable message here and has little influence on the overall complexity.

Turns Pseudo Preimage into Preimage After preparing preimage for the last message block $F(H, M) = T$ and the expandable message $IV \xrightarrow{M^*} H'$. Now we can connect them to form a first preimage.

Suppose it takes 2^c to find a pseudo preimage. A traditional MitM approach can convert preimage attack on the compression function into preimage attack on the hash function works like this. First, find and store 2^k pseudo preimages with 2^{k+c} computations and 2^k memory. Then choose 2^{n-k} random message , calculate from IV to find a chaining value appearing in one of the pseudo preimages. The complexity is $2^{n-k} + 2^{k+c}$. Take the optimal $k = \frac{n-c}{2}$, the minimum complexity is $2^{\frac{n+c}{2}+1}$. Using the pseudo preimage attack described in Sect. C.2, the preimage attack has a complexity of $(2^{481.5}, 2^{64})$.

D Improved MITM Attacks against AES Hashing Modes

It has been shown by Sasaki [14] that pseudo/second preimage of 7-round AES can be found in 2^{120} under hashing modes of Davies-Meyer (DM), Matyas-Meyer-Oseas (MMO), and Miyaguchi-Preneel (MP). In this section, we show that the pesudo-preimage can speed up with the help of the multi-target pseudo-preimage techniques proposed in [7], when the number of given targets is more than one. These improvements result in faster preimage and second preimages of 7-round AES hash modes. Details of the results are summarized in Table 5, comparing with those by Sasaki in [14].

Attack	Mode	Time	Memory	Message length	Reference
2nd Preimage	MMO, MP	2^{120}	2^8	-	[14]
	MMO, MP, DM	2^{128-k}	2^k	2^k blocks	[10]
	MMO, MP, DM	$2^{120-min(k,24)}$	$2^{8+min(k,24)}$	2^k blocks	this section
Preimage	DM	2^{125}	2^8	-	[14]
	DM	$2^{122.7}$	2^{16}	$> 2^8$ blocks	this section

Table 5. Results on (Second) Preimages of 7-round AES Hashing Modes.

We refer to Fig. 17 for the details of the attack. The states of AES are divided into two chunks, while the backward (red) chunk consists of states #8—#15 and forward (blue) chunk consists of states #20—#28 and #0—#7, the initial structure works for states #16—#19. These divisions are same as in [14]. However, when setting degree freedoms for both chunks, we find that we can have 2^{32} and 2^8 for the backward and forward directions, respectively. There is only one free byte in blue as in state #15 and the rest three bytes in the colunn are set to some constants. This byte is later propagated through the MixColumn into all four bytes of the first column in state #16. Similarly, we do not allow influence from red bytes in state #19 into the blue bytes in #20. Hence there is only (3-2)=1 free byte in each column of #19, which results in at most 2^{32} (4 bytes) for backward chunk. If one considers the situation that there are 2^k available targets T. Then the attack works in the follows.

1. Use $2^{8+min(k,24)}$ freedom degrees out of 2^{32} for the backward direction, and compute the values of the bytes in red and gray from state #15 back to #8, store them in a table.
2. For all 2^8 freedom degrees, compute the values of the bytes in blue and gray from state #15—#28, then for each of the 2^8 candidates, xor the $2^{min(k,24)}$ targets, so that $2^{8+min(k,24)}$ candidates will be available for state #0. Continue compute forward up to state #7.
3. Carry out the indirect partial matching between state #7 and #8.
4. Repeat until a full match is found.

It is easy to see that the overall complexity for this attack is $2^{120-min(k,24)}$, with memory requirement $2^{8+min(k,24)}$. While this can be directly applied when finding second preimages, we will use a tree-like construction as in [7] to find a first preimages for the AES hash in Davies-Meyer mode. Generally, given k targets with $1 < k < 2^{24}$, a pseudo preimage can be found in $2^{120}/k$. When finding the first preimage given one target T, one finds a pseudo preimage with chaining T_2 in 2^{120}, then finds the second pseudo preimage with the target set $\{T, T_2\}$ in time $2^{120}/2$, and so on. Hence finding Z pseudo preimages costs $\sum_{z=1}^{Z} 2^{120}/z \simeq 2^{120} \cdot \ln(Z)$. Finally, finding a message linking the IV to one of the targets costs $2^{128}/Z$. The overall time complexity is $2^{120} \cdot \ln(Z) + 2^{128}/Z$, which is $2^{122.7}$ when $Z = 2^8$ is chosen.

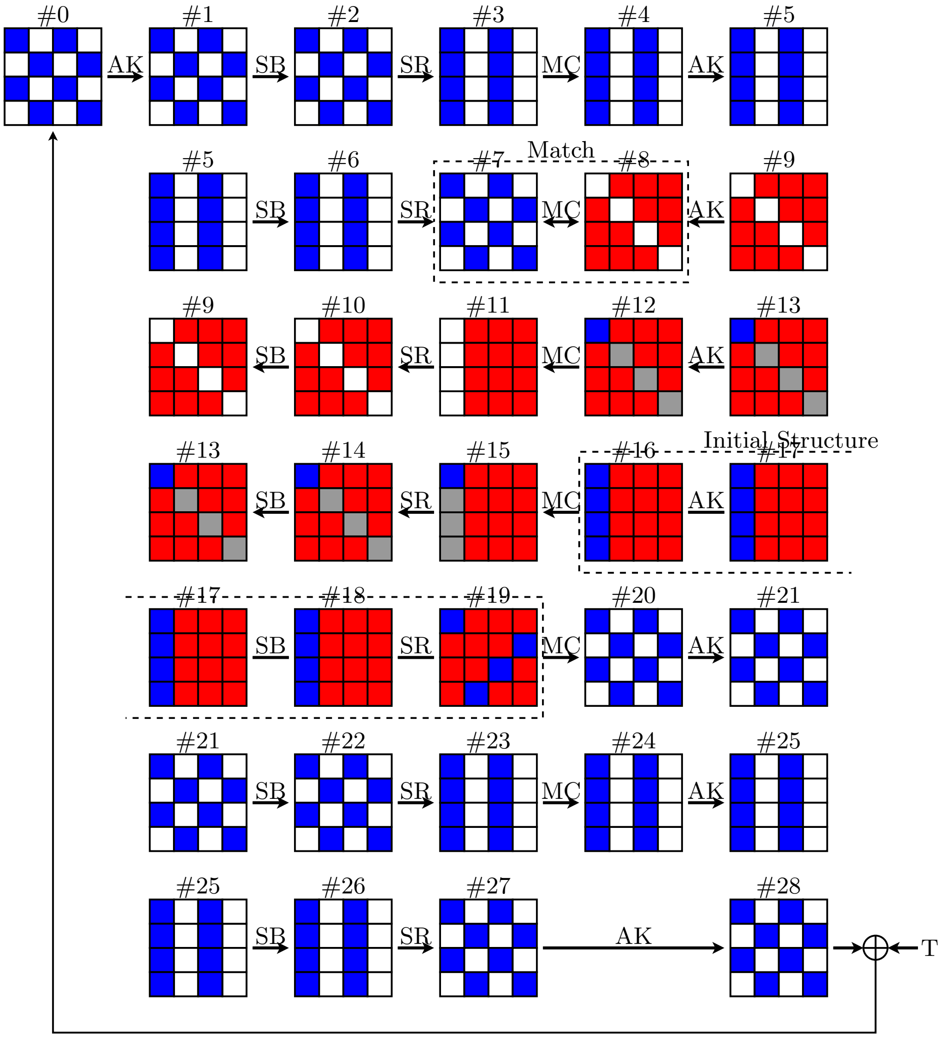


Fig. 17. MITM Attack against 7-Round `AES`

Practical Cryptanalysis of ARMADILLO2

María Naya-Plasencia[1,*] and Thomas Peyrin[2,**]

[1] University of Versailles, France
maria.naya-plasencia@prism.uvsq.fr
[2] Division of Mathematical Sciences, School of Physical and Mathematical Sciences,
Nanyang Technological University, Singapore
thomas.peyrin@gmail.com

Abstract. The ARMADILLO2 primitive is a very innovative hardware-oriented multi-purpose design published at CHES 2010 and based on data-dependent bit transpositions. In this paper, we first show a very unpleasant property of the internal permutation that allows for example to obtain a cheap distinguisher on ARMADILLO2 when instantiated as a stream-cipher. Then, we exploit the very weak diffusion properties of the internal permutation when the attacker can control the Hamming weight of the input values, leading to a practical free-start collision attack on the ARMADILLO2 compression function. Moreover, we describe a new attack so-called local-linearization that seems to be very efficient on data-dependent bit transpositions designs and we obtain a practical semi-free-start collision attack on the ARMADILLO2 hash function. Finally, we provide a related-key recovery attack when ARMADILLO2 is instantiated as a stream cipher. All collision attacks have been verified experimentally, they require negligible memory and a very small number of computations (less than one second on an average computer), even for the high security versions of the scheme.

Key words: ARMADILLO2, hash function, stream-cipher, MAC, cryptanalysis, collision

1 Introduction

Hash functions are among the most important and widely spread primitives in cryptography. Informally a hash function H is a function that takes an arbitrarily long message as input and outputs a fixed-length hash value of size n bits. The classical security requirements for such a function are collision resistance and (second)-preimage resistance. Namely, it should be impossible for an adversary to find a collision (two different messages that lead to the same hash value) in less than $2^{n/2}$ hash computations, or a (second)-preimage (a message hashing to a given challenge) in less than 2^n hash computations. In general, a hash function H is built from an iterative use of a n-bit output compression function h in a Merkle-Damgård-like operating mode [6, 4]. The compression function takes a chaining variable CV (fixed to an initial value IV at the beginning) and a message block M as inputs and in order to allow security proofs on the operating mode, one requires the same security properties as a hash function, namely collision and (second)-preimage resistance. However, the compression function allows several flavors of security properties depending on how well the attacker can control the chaining variable:

- free-start collision: the attacker fully controls the chaining variable, i.e. both its value and difference
- semi-free-start collision: the attacker control partially the chaining variable, i.e. only its value, and the difference is null
- collision: the attacker does not control the chaining variable, the value is defined by the IV and the difference is null

[*] Supported by the French Agence Nationale de la Recherche through the SAPHIR2 project under Contract ANR-08-VERS-014.

[**] The author is supported by the Lee Kuan Yew Postdoctoral Fellowship 2011 and the Singapore National Research Foundation Fellowship 2012.

For all three flavors, it should be impossible for an adversary to find a collision in less than $2^{n/2}$ compression function computations. Note that free-start collision is required as necessary assumption regarding the compression function in the Merkle-Damgård-like security proofs. Moreover, a semi-free-start collision means there exists initial values IV for which it is possible to find collisions for the hash function. Therefore, both these two notions are very important and should be verified for a secure compression function.

ARMADILLO2 [2] is a very novel primitive dedicated to hardware, defining a FIL-MAC, a stream cipher and a hash function. Originally, two versions were proposed, ARMADILLO and ARMADILLO2, the later being the recommended one. A key recovery attack on ARMADILLO was rapidly published by a subset of the designers [9]. ARMADILLO2 remained unbroken until Abdelraheem *et al.* [1] found a meet-in-the-middle technique that allows to invert the ARMADILLO2 main function. This cryptanalysis eventually led to a key recovery attack on the FIL-MAC and the stream cipher, and a (second)-preimage attack on the hash function. However, while being the first weakness published on ARMADILLO2, this work is an improved meet-in-the-middle technique, therefore requiring a lot of computations and memory, often close to the generic complexity. For example, the preimage attack on the 256-bit output hash function requires either 2^{208} computations and 2^{205} memory or 2^{249} computations and 2^{45} memory. With its data-dependent bit transpositions and original compression function construction, ARMADILLO2 is clearly not following the classical design trends for symmetric-key primitives (for example RC5 [7] and RC6 [8] use data-dependent rotations, while IDEA [5] use data-dependent multiplication). As a consequence, it would be interesting to look at this proposal without necessarily relying on known cryptanalysis techniques.

Our contributions. In this paper, we first observe the very unpleasant property that the parity bit is preserved through all ARMADILLO2 internal permutations. This allows us for example to derive a very cheap distinguisher for the stream-cipher. Then, we analyze the differential diffusion of the permutations and we provide practical free-start collision attacks for all versions of the compression function of ARMADILLO2. We extend our results by introducing a new technique, the *local linearization*, that seems very efficient against data-dependent bit transpositions. This method led us to practical semi-free-start collision attacks for all versions of ARMADILLO2. All attacks require very few computations (at most $2^{10.2}$ operations for 256-bit output version) and negligible memory. Moreover, our implementations validate our techniques and we provide collision examples. Finally, we provide a related-key recovery attack when ARMADILLO2 is instantiated as a stream cipher.

2 The ARMADILLO2 function

We let $X[i]$ denote the i-th bit of a word X. Let C be an initial vector of size c and U be a message block of size m. The size of the register $(C||U)$ is $k = c + m$, where $||$ denotes the concatenation operation. The internal ARMADILLO2 function transforms the vector (C, U) into (V_c, V_t) as described in Figure 1, $(V_c, V_t) = \texttt{ARMADILLO2}(C, U)$. The internal ARMADILLO2 function relies on a parameterized permutation on k bits Q, instantiated by Q_U and Q_X, where U is a m-bit parameter and X is a k-bit parameter.

Let σ_0 and σ_1 be two fixed bitwise permutations of size k. In [2], the permutations are not specifically defined but some criteria they should fulfill is given. We denote by cst a constant of size k defined by alternating 0?s and 1?s, i.e. : $cst = \texttt{1010} \cdots \texttt{10}$. Using these notations, we can specify Q which is used twice in the internal ARMADILLO2 function. Let A be the a-bit parameter and B be the k-bit input of Q, the parameterized permutation Q_A can be divided into $a = |A|$ simple steps. The i-th step of Q_A (reading A from its least significant bit to its most significant one) is defined by:

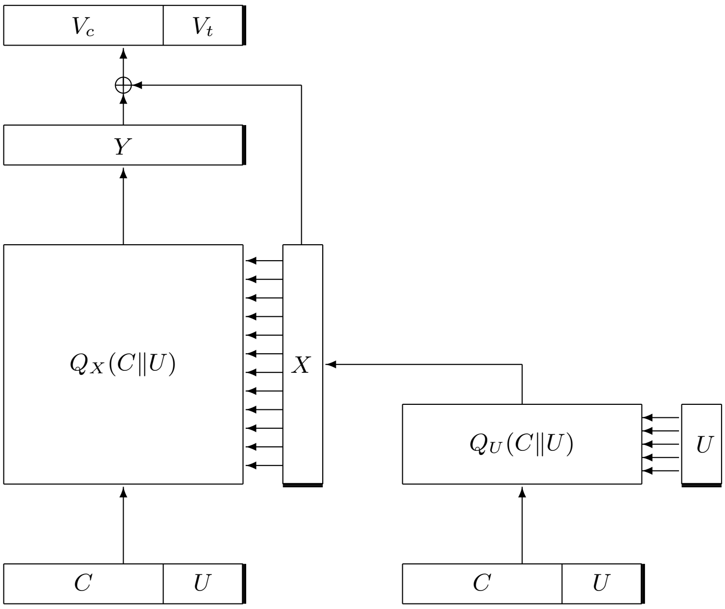


Fig. 1. The internal function of `ARMADILLO2`. The thick line at the side of a register represents the least significant bit.

- an elementary **bitwise permutation**: $B \leftarrow \sigma_{A[i]}(B)$, that is if the i-bit of A is `0` we apply σ_0 to B, otherwise we apply σ_1.
- a **constant addition** (bitwise XOR) of cst: $B \leftarrow B \oplus cst$.

The internal `ARMADILLO2` function first computes $X = Q_U(C||U)$, then $Y = Q_X(C||U)$, and finally outputs $(V_c, V_t) = Y \oplus X$.

Using this internal primitive, `ARMADILLO2` builds a FIL-MAC, a stream-cipher and a hash function:

- **Stream-cipher**: the secret key is inserted in the C register and the output sequence is obtained by taking the k bits of the output (V_c, V_t) after one iteration. The keystream is composed of k-bit frames indexed by U (which is a public value).
- **Hash function**: it uses a strengthened Merkle-Damgård construction, where V_c represents the output of the compression function (i.e. the next chaining value or the hash digest), U is the incoming message block and C is the incoming chaining variable.
- **FIL-MAC**: the secret key is inserted in the C register and the challenge, considered known by the attacker, is inserted in the U register. The response to the challenge is the m-bit output V_t.

Five different sets of register sizes (k, c, m) are provided, namely $(128, 80, 48)$, $(192, 128, 64)$, $(240, 160, 80)$, $(288, 192, 96)$ and $(384, 256, 128)$.

3 First tools

We denote `HAM`(X) the Hamming weight of the word X. We recall from [1] that for two random k-bit words A and B of Hamming weight a and b respectively, the probability that `HAM`$(A \wedge B) = i$

(where $\wedge$ stands for the bitwise AND function) is given by the formula

$$P_{\texttt{and}}(k,a,b,i) = \frac{\binom{a}{i}\binom{k-a}{b-i}}{\binom{k}{b}} = \frac{\binom{b}{i}\binom{k-b}{a-i}}{\binom{k}{a}}.$$

Moreover, we would like to deduce from it the probability that $\texttt{HAM}(A \oplus B) = i$ (where $\oplus$ stands for the bitwise XOR function) for two randomly chosen k-bit words A and B of Hamming weight a and b respectively. We remark that $\texttt{HAM}(A \oplus B) = a + b - 2 \cdot \texttt{HAM}(A \wedge B)$ and therefore the probability that $\texttt{HAM}(A \oplus B) = i$ is given by the formula

$$P_{\texttt{xor}}(k,a,b,i) = \begin{cases} P_{\texttt{and}}(k,a,b,\frac{a+b-i}{2}) & \text{for } (a+b-i) \text{ even} \\ 0 & \text{for } (a+b-i) \text{ odd} \end{cases}$$

Since they have not been specified in the original ARMADILLO2 document, in the following we assume that σ_0 and σ_1 are randomly chosen bit permutations.

4 Parity preservation

We call the parity bit of an a-bit word A the bit value $\bigoplus_{i=0}^{a-1} A[i]$. Regardless of the parameter A of the internal permutation Q_A, we have that **the parity of the input is always maintained through the permutation**. This can be easily verified by remarking that Q_A is composed of several identical rounds, all satisfying this property. Indeed, one round is composed of a bit permutation (which fully maintains the Hamming weight) and an XOR of the internal state with the constant cst = 1010...10. This constant being always the same during the whole ARMADILLO2 computation and its parity being even, the parity of the internal state remains the same after application of the XOR. Note that even if this constant was changed during the rounds, the attacker would only have to compute the parity of the XOR of all constants to be able to tell if the parity bit will be maintained or negated. This property is moreover maintained whatever number of rounds is applied in the permutations, thus the attack proposed in this section is independent of the number of rounds.

Distinguisher for the stream cipher mode. We can exploit the previous property to build a cheap distinguisher on ARMADILLO2 when used as a stream-cipher. In the attack model, the whole output of the function is assumed to be known as it is a frame of the keystream. This output is generated by a XOR of internal states X and Y. Since permutations Q_U and Q_X will maintain the parity, their respective outputs X and Y will both have the same parity as $(C||U)$. As a consequence, the output of the function $X \oplus Y$ always has an even parity. For a random sequence, this will only happen with probability 1/2, as for ARMADILLO2 this happens with probability 1. In other words, the entropy of the ARMADILLO2 function output is reduced by one bit.

5 Controlled diffusion: practical free-start collision attack

In this section, we show how an attacker can control the bit difference diffusion in ARMADILLO2 function by using the available inputs. This leads to a very cheap free-start collision attack against the compression function.

5.1 General description

Assume that we insert a single bit difference in C, that is $\texttt{HAM}(\Delta C) = 1$, and no difference in U that is $\Delta U = 0$. We can use c distinct ΔC, one for each active bit position. The attack is depicted in Figure 2.

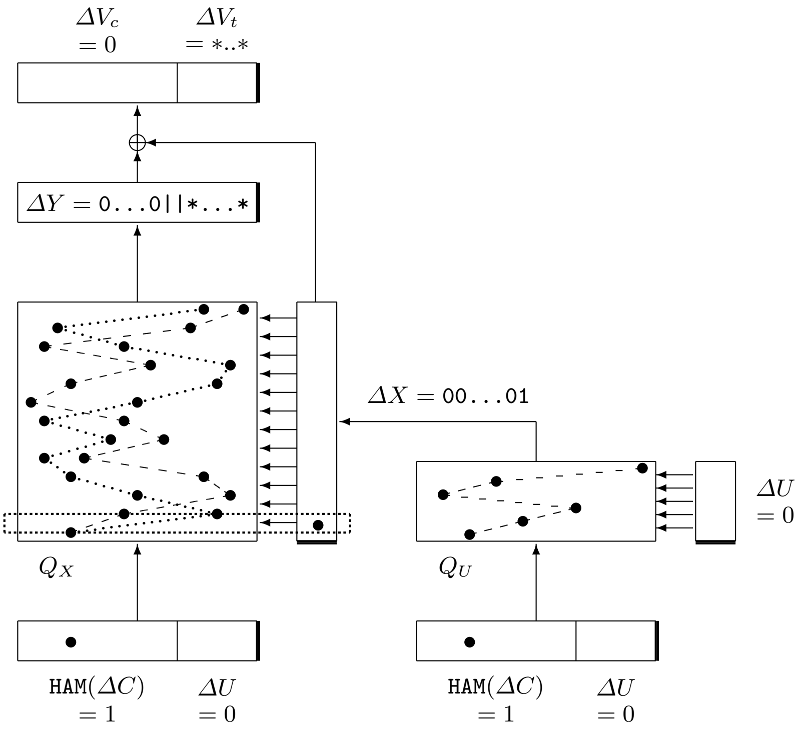


Fig. 2. A schematic view of the free-start collision attack on ARMADILLO2. The thick line at the side of a register represents the least significant bit and black circles stand for bit differences. The dashed box indicates the first round of Q_X, which contains a difference on its corresponding parameter input bit.

Difference propagation in Q_U. Since we have no difference in U, the permutation Q_U always remains the same. We only have to study the propagation of the bit difference in C through Q_U. Note that one round of the internal permutation Q_U provides no difference diffusion since it is only composed of a bit permutation and a constant addition. Therefore, the single bit difference in C will be just transfered to some random bit position in X at the end of Q_U and we have HAM(ΔX) $= 1$. We would like the single bit difference in X to be positioned in bit 0, i.e. $\Delta X =$ 00...01 (this will later allow us to use the freedom degrees efficiently). For a randomly chosen value of U and C, this happens with probability

$$P_X = \frac{1}{k}.$$

Difference propagation in Q_X. Since we have a single difference on the first bit of X (corresponding to the first step of Q_X), the permutation Q_X remains the same except for the first step where we switch from bit permutation σ_0 to σ_1 or from σ_1 to σ_0. We denote by $P_{step}(in, out)$ the probability that in active bits are mapped to out active bits through a step of data-dependent permutation with a difference (i.e. σ_0 and σ_1 are swapped). Assume for the moment that after this first step, only b bits are active in the internal state. This happens with probability $P_{step}(1, b)$. Since the next rounds of the internal permutation Q_X provide no difference diffusion, we end up in Y with b active bits randomly distributed. We need to ensure that all the b active bits remaining in Y will go to the m-bit V_t part of the k-bit output, so that all differences will be truncated and we eventually obtain a collision on the output of the compression function. For $b \leq m$, this happens with probability

$$P_{out}(b) = P_{\mathtt{and}}(k, m, b, b) = \frac{\binom{b}{b}\binom{k-b}{m-b}}{\binom{k}{m}} = \prod_{i=0}^{i=b-1} \frac{m-i}{k-i}.$$

During the feed-forward after Q_X the single active bit of X is already on the V_t part of the output. Overall the probability of obtaining a compression function collision for randomly chosen U and C values is:

$$P_{collision} = P_X \cdot \sum_{i=1}^{i=m} P_{step}(1,i) \cdot P_{out}(i).$$

the sum stopping at m because when $i > m$, we trivially have $P_{out}(i) = 0$. At this point our problem is that in order for the probability $P_{out}(i)$ to be high enough, we need the number i of active bits to be small. On the other side, if i is small, $P_{step}(1,i)$ will be very low (we do not explain how to compute $P_{step}(1,i)$ here as we will study a slightly more detailed problem in the next section). However, in this scenario we only considered an attacker that randomly chooses the value of U and C and the bit difference position in C, but we can do much better by using the available degrees of freedom efficiently.

5.2 Using the freedom degrees

First, note that the event related to the probability P_X only depends on the position of the bit difference in C and on the value of U. We can therefore attack Q_U in a first phase (by fixing the position of the bit difference in C and the value of U), and then independently attack Q_X by choosing the value of C.

Handling Q_U. We will see later that we would like C and U values to have an extremely low or extremely high Hamming weight. Therefore, we fix ΔX = `00...01` and test with the two values U = `00..00` and U = `11..11` how the bit difference will propagate through Q_U^{-1} (note that we are dealing with the inverse of Q_U, thus attacking backwards from ΔX). For each try, we have a probability $P_{\texttt{and}}(k, c, 1, 1) = c/k$ that the single bit difference is mapped to the C part of the input. Since for all `ARMADILLO2` versions we have $2c/k > 1$, we expect at least one of the two U candidates to satisfy ΔX = `00...01`, $\texttt{HAM}(\Delta C) = 1$ and $\texttt{HAM}(\Delta U) = 0$. Overall, this phase costs us only 2 operations. We assume without loss of generality that the selected candidate has value U = `00..00`.

Handling Q_X. At the present time, everything is fixed except the value of C and we have ΔX = `00...01` and U = `00..00`. We now describe a simple criteria in order to choose the values of C such that the first round probability $P_{step}(1,i)$ in Q_X is high, even for small i. As an example, let's assume that $C = 0$, that is $\texttt{HAM}(C||U) = 0$. In that case, we trivially have that $P_{step}(1,1) = 1$ (and $P_{step}(1,i) = 0$ for all other i) since changing the bit positions of the word `00..00` (switching from σ_0 to σ_1 or from σ_1 to σ_0) will not have any effect at all and the single bit difference in C will just be placed to some random bit position. Similarly, with a single one-bit in C, that is $\texttt{HAM}(C||U) = 1$, we have that $P_{step}(1,1) = \frac{1}{128} + \frac{2 \cdot 127}{128^2}$ and $P_{step}(1,3) = \frac{127 \cdot 126}{128^2}$ (and $P_{step}(i) = 0$ for all other i). More generally, we have to compute the probability $P_{step}(1,b,hw)$ which corresponds to the probability $P_{step}(1,b)$ knowing that the input word hamming weight is hw. This can be modeled as follows: choose two random k-bit words x and y both with Hamming weight hw (they represent $\sigma_0(C||U)$ and $\sigma_1(C||U)$) and compute $z = x \oplus y \oplus 1$ (the 1 represents the single bit difference in C). Then $P_{step}(1,b,hw)$ is the probability that $\texttt{HAM}(z) = b$ (note that $\texttt{HAM}(z)$ is always odd thus we have $P_{step}(1,2i,hw) = 0$ for all i) and we have:

$$P_{step}(1,b,hw) = \frac{hw}{c} \cdot P_{\texttt{xor}}(k, hw, hw-1, b) + \frac{c-hw}{c} \cdot P_{\texttt{xor}}(k, hw, hw+1, b).$$

The complexity for handling Q_X is finally

$$Comp = \frac{1}{\sum_{i=1}^{i=m} P_{step}(1, i, hw) \cdot P_{out}(i)}.$$

5.3 Complexity results

The number C of candidate values we can generate with Hamming weight hw is $\binom{c}{hw}$ and in order to have a good chance to find a collision after Q_X with this amount, we need to ensure that

$$\binom{c}{hw} \geq 1/\sum_{i=1}^{i=m} P_{step}(1, i, hw) \cdot P_{out}(i).$$

One can check that in order to minimize the complexity $Comp$, the dominant factor of the sum is when i is small. Then, for i small, $P_{step}(1, i, hw)$ is higher when hw is close to 0 or close to k, in other words the input should have very low or very high Hamming weight. Since we previously chose $U = 00..00$ our goal is to find for each ARMADILLO2 versions the smallest hw value hw_{min} that ensures enough C candidate values to handle the collision probability in Q_X (but the same reasoning is possible with $U = 11..11$ and the biggest hw value hw_{max}). Overall, the full attack runs in $2 + Comp$ operations (i.e. compression function calls) and negligible memory in order to find a free-start collision for the ARMADILLO2 compression function. We depict in Table 1 our results relative to all proposed versions of ARMADILLO2. This attack has been implemented and verified in practice for $k = 128$ and we give free-start collision examples in the Appendix.

Table 1. Summary of results for free-start collision attack on the different size variants of the ARMADILLO2 compression function. The number of C candidates must always be enough so as to handle the collision probability in Q_X.

scheme parameters				**attack parameters**			
k	c	m	generic complexity	hw_{min}	number of C candidates	collision probability in Q_X	attack complexity
128	80	48	2^{40}	1	$2^{6.3}$	$2^{-4.1}$	$2^{7.5}$
192	128	64	2^{64}	1	2^{7}	$2^{-4.6}$	$2^{7.8}$
240	160	80	2^{80}	1	$2^{7.3}$	$2^{-4.7}$	$2^{8.1}$
288	192	96	2^{96}	1	$2^{7.6}$	$2^{-4.7}$	$2^{8.3}$
384	256	128	2^{128}	1	2^{8}	$2^{-4.8}$	$2^{8.7}$

6 Local linearization: practical semi-free-start collision attack

In this section, we show how one can obtain a semi-free-start collision attack (no difference on the input chaining variable) with a very low computational complexity for the ARMADILLO2 compression function.

6.1 General description

The previous method only allows to add differences on the capacity part of the input, thus leading to free-start collision attacks. One can directly extend this technique to allow only differences in the message part of the input, but this only leads to semi-free-start collisions for randomly chosen bit permutations σ_0 and σ_1 with a not-so-high probability of success.

We would like to derive a semi-free-start collision attack that will output a result with very high probability. In order to achieve this goal we propose a new technique for data-dependent bit transposition ciphers, so-called **local linearization**: by guessing some part of the input we are able to render a few rounds of the internal permutation linear. Indeed, by knowing the g first bits of U we completely determine the permutations applied during the first g rounds of Q_U. Therefore, for those g rounds the primitive Q_U only consists of known bit permutations and known constant additions. With this method we neutralize for the first g rounds the only non-linearity source: the fact that we don't know which bit permutation σ_0 or σ_1 is applied each round.

On a high-level view, our semi-free-start collision attack will force a collision on the X value at the output of Q_U thanks to the local linearization technique. This collision on X will ensure that the Q_X permutation will be the same for both inputs. Therefore, the difference Hamming weight on the input of Q_X will remain the same in the output. We then hope that those bit differences will be mapped in the truncated part of the output in order to eventually obtain the semi-free-start collision (no difference is feed-forwarded from X since we forced a collision on it). The attack is depicted in Figure 3.

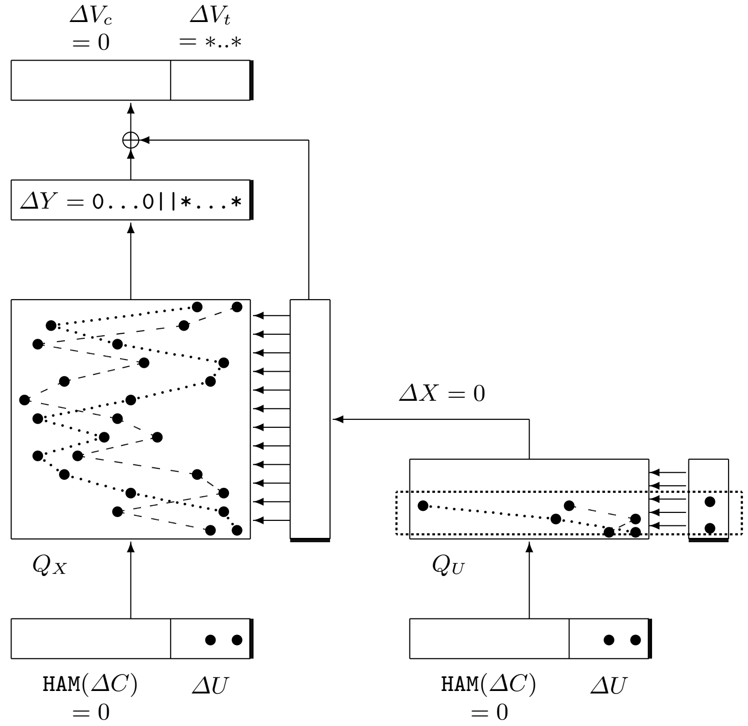


Fig. 3. A schematic view of the semi-free-start collision attack on ARMADILLO2. The thick line at the side of a register represents the least significant bit and black circles stand for bit differences. The dashed box indicates the linearized part.

During a first phase, the input will be divided into two parts: the fixed and the unfixed part. The fixed part $z \in \{0,1\}^g$ is composed of the g first bits of U and we choose random values for those g bits (so as to know the g first choices of σ_0 or σ_1). The unfixed part $w \in \{0,1\}^{k-g}$ is composed of the rest of the input bits and we will be set to a value later. We force the input difference to be contained in the fixed part and we denote it $\Delta z \in \{0,1\}^g$ (since we are looking for semi-free-start collisions we obviously have $g \leq m$, otherwise we would have a difference in the input chaining variable C). Let $I_1 = (C_1||U_1)$ (resp. $I_2 = (C_2||U_2)$) be the k-bit value of the first input (resp. second output), we have:

$$I_1 = (x||z) \text{ and } I_2 = (x||z \oplus \Delta z).$$

and our goal is to have the collision $X = Q_{U_1}(I_1) = Q_{U_2}(I_2)$.

Assume for the moment that this collision on X happens. Then the same permutation Q_X will be used for both inputs I_1 and I_2 on the right side of Figure 1. As a consequence, no additional bit difference will be introduced during the computation of Q_X, but the bit difference positions will be randomly moved. In order to obtain a semi-free-start collision on the output of the function, we need the $b = \mathtt{HAM}(\Delta z)$ active bits of the input to be mapped in the truncated part of the output through Q_X. As already explained in Section 5, this happens with probability

$$P_{out}(b) = P_{\mathtt{and}}(k, m, b, b) = \prod_{i=0}^{i=b-1} \frac{m-i}{k-i}.$$

6.2 Colliding on *X*

We need now to evaluate the probability of getting a collision on X. Note that for any round, if there is no difference on the bit choosing the permutation to apply σ_0 or σ_1, the bit differences at the input of this round will only have their position changed and cannot be erased. Therefore, if we want to obtain a collision on X, we need to obtain it at latest just after the last round of Q_U for which a difference is inserted on the side (in U). We consider from now on that the input difference Δz contains at least one active bit on its MSB, thus this last round is the g-th one.

We know the value of the g first bit of U, therefore we know exactly the permutation applied to I_1 and I_2 for the g first rounds of Q_U. For a collision after g rounds of Q_U, we want that

$$\begin{aligned} &\sigma_{U_1[g-1]}(\cdots(\sigma_{U_1[1]}(\sigma_{U_1[0]}(I_1) \oplus cst) \oplus cst)\cdots) \\ &= \sigma_{U_2[g-1]}(\cdots(\sigma_{U_2[1]}(\sigma_{U_2[0]}(I_2) \oplus cst) \oplus cst)\cdots) \end{aligned}$$

and since **all operations are linear**, this can be rewritten as

$$\rho(I_1) \oplus A = \rho'(I_2) \oplus B = \rho'(I_1 \oplus \Delta z) \oplus B = \rho'(I_1) \oplus \rho'(\Delta z) \oplus B$$

where

$$\begin{aligned} \rho &= \sigma_{U_1[g-1]} \circ \cdots \sigma_{U_1[1]} \circ \sigma_{U_1[0]} \qquad & A &= \sigma_{U_1[g-1]}(\cdots(\sigma_{U_1[1]}(cst) \oplus cst)\cdots) \\ \rho' &= \sigma_{U_2[g-1]} \circ \cdots \sigma_{U_2[1]} \circ \sigma_{U_2[0]} \qquad & B &= \sigma_{U_2[g-1]}(\cdots(\sigma_{U_2[1]}(cst) \oplus cst)\cdots). \end{aligned}$$

Finally, we end up with the equation

$$\rho(I_1) \oplus \rho'(I_1) = A \oplus B \oplus \rho'(\Delta z) \tag{1}$$

Since we know the value of the g first bit of U, we can compute the value of A and B. Moreover, assuming that we already chose a Δz, then the collision condition (1) can be rephrased as

$$I_1 \oplus \tau(I_1) = C$$

where $C = \rho^{-1}(A \oplus B \oplus \rho'(\Delta z))$ and $\tau = \rho^{-1} \circ \rho'$.

In order to study this system $\mathcal{S}$ of k bit equations, we model τ as a random bit permutation and C as a random k-bit word. Note that since this equation system is linear finding the potential solutions requires only a few operations, but we would like to know how many such systems we need to generate before finding a solution, i.e. a collision on X. Thus, our goal is now to deduce the probability that this system has at least one solution and what is the average number of expected solutions.

The structure of this equation system is very particular and the number of independent groups of bit equations is exactly the number of cycles of the bit permutation τ. More precisely, let $\texttt{CYCLE}(\tau)$ represent the number of cycles of the permutation τ and let S_i denote the set of bits belonging to the i-th cycle of τ.

Theorem 1. *The equation system $\mathcal{S} : I_1 \oplus \tau(I_1) = C$ admits a solution if and only if for every cycle set S_i of τ the parity of the sum of the corresponding C bit is null, that is*

$$\bigoplus_{p \in S_i} C[p] = 0.$$

If this system is solvable, then the number of solutions that can be generated is exactly equal to $2^{\texttt{CYCLE}(\tau)}$.

The idea of the theorem is that when we want to find a solution for the system, we can start by fixing one bit a_0 to a random value. This bit is involved into two binary equations from $\mathcal{S}$. All equations having only two terms, one of the two equations directly links bit a_0 with say bit a_1, and we can deduce the value of a_1. The bit a_1 is in turn linked with bit a_2 through his second equation and we directly deduce the value of a_2. This chain of dependency will eventually cycle (the new bit deduced will be a_0 again) and will be validated if and only if the sum of the C bits of the equations visited is null (otherwise we encounter a inconsistency). This check is then performed for all cycles.

Proof. Since τ is a bit permutation, the equation system $\mathcal{S}$ can be represented as a collection of cycles, each cycle depicting the direct cyclical dependencies between some set of bits: if bit x and bit y are linked by one of the k equations, then they belong to the same cycle. The vertex weight between two members x and y of the cycle is the value $C[x]$.

If we fix the bit value of a member of a cycle S_i, then this determines entirely all the other bits of that cycle (according to the vertices values). Then, if the XOR of all the vertex weights is different from zero, we have a direct contradiction. A solution can only exist if all cycles present no internal contradiction.

Each cycle can have either zero or two solutions (the two solutions being their mutual complement). If every cycle has no contradiction, then there exists exactly $2^{\texttt{CYCLE}(\tau)}$ distinct combinations of cycle solutions, each one leading to a distinct solution for the whole equation system $\mathcal{S}$. □

From Theorem 1, we directly deduce that the probability that the system admits a solution is equal to $2^{-\texttt{CYCLE}(\tau)}$. The expected number of cycles for a randomly chosen permutation on k elements is $\log(k)$. Therefore, we have to try at least $2^{\log(k)}$ different equation systems before finding one admitting a solution. When one system admits a solution, we directly get $2^{\log(k)}$ solutions for free. Overall, the cost for finding one solution of the system is 1 on average (the average cost is the meaningful one here since we will have to find several inputs colliding on X during the whole attack).

6.3 Complexity results

We now look for a solution such that the original guess of the g first bits of the input was right (with probability 2^{-g}) and such that the b bit differences in Q_X are mapped to the truncated part of the output (with probability $P_{out}(b)$). Overall, the total complexity of the semi-free-start collision attack is $2^g \cdot P_{out}^{-1}(b)$ with $b \leq g$. Minimizing g and b will minimize the overall complexity, but we need to ensure that we can go through enough equation systems in order to have a good chance to find a collision eventually. More precisely, we need

$$1/2 \cdot 2^g \cdot \binom{g}{b} \geq 2^g \cdot P_{out}^{-1}(b)$$

which can be rewritten as

$$\binom{g}{b} \geq 2 \cdot P_{out}^{-1}(b).$$

We depict in Table 2 our results relative to all proposed versions of ARMADILLO2. This attack has been implemented and verified in practice for $k = 128$ and we give semi-free-start collision examples in the Appendix.

Table 2. Summary of results for semi-free-start collision attack on the different size variants of the ARMADILLO2 compression function.

scheme parameters				attack parameters			
k	c	m	generic complexity	g	b	$P_{out}(b)$	time complexity
128	80	48	2^{40}	6	2	$2^{-2.9}$	$2^{8.9}$
192	128	64	2^{64}	7	2	$2^{-3.2}$	$2^{10.2}$
240	160	80	2^{80}	7	2	$2^{-3.2}$	$2^{10.2}$
288	192	96	2^{96}	7	2	$2^{-3.2}$	$2^{10.2}$
384	256	128	2^{128}	7	2	$2^{-3.2}$	$2^{10.2}$

7 Related-key recovery in stream cipher mode

In this section we will present a related key attack that will allow us to recover all key bits in practical time when using ARMADILLO2 in the stream cipher mode. We will first present the main idea of this attack, and afterwards, we will give a more detailed analysis of the probabilities and complexities.

7.1 Using Related-keys for Recovering the Key

First of all, we consider a pair of related keys (K_1, K_2) that have one only bit of difference, that is $\mathtt{HAM}(K_1 \oplus K_2) = \mathtt{HAM}(\Delta_K) = 1$. Our analysis will work for any bit difference position d amongst all the bits of the key. Note that we expect a pair of keys valid for performing the related-key attack to appear after using about $(2^k/k)^{1/2}$ keys.

Let us consider a value of U for generating k bits of key-stream with each of both keys K_1 and K_2. We use the index i for the intermediate states generated from the key K_i. We first make the following observations, important in order to understand the whole attack procedure:

- Since no difference is inserted in the U part (it is a public value) and since $\mathtt{HAM}(\Delta_K) = 1$, we have $\mathtt{HAM}(X_1 \oplus X_2) = 1$. Let e be the bit position of this difference in X.
- The first $(e-1)$ intermediate states of Q_X will also have a difference of Hamming weight 1.

We assume that the attacker can choose the values of U. In this case, we can make the bit difference in the key to go from position d to any wanted position e in X through Q_U. We expect $2^m/k$ distinct values of U that make the bit difference go from position d to e for $e \in [0, k-1]$. We denote by U_e each one of these k subgroups of U values.

The output of the function $(V_c, V_t) = X \oplus Y$ is known to the attacker, but concerning X he only knows the m bits of the U part (since U is known, he can deduce directly where the bits coming from U and C will be eventually located in X). Thus, he can recover m bits from the outputs of Q_X, Y_1 and Y_2. If he could compute backward from Y_1 and Y_2 until the beginning of the e-th step of Q_X, the colliding positions of the bits known from Y_1 and from Y_2 will have the same values with maybe the exception of one, which would be the original single bit difference before the step e.

Our attack basically consists in choosing several values for U from U_e, for decreasing e values (starting from $e = k-1$), that will gradually increase the number of key bits appearing in X after position e. Each time we will guess the value of the new key bits appearing and discard the guesses that will not lead to collisions on the bit values in the colliding positions just before step e when computing backward from Y_1 and Y_2 in Q_X. The complexity of this attack depends on the bit permutations σ_0 and σ_1, but in the next subsection we give a complexity analysis assuming that these permutations are randomly chosen.

7.2 Generic Complexity Estimation

We start at $e = k-1$. First, we choose the value of i (denoted i_{max}), that maximizes the probability $P_{\texttt{and}}(k, m, m, i)$ that we denote p_{max}. For instance, if we consider the smallest version of ARMADILLO2, where $k = 128$, $c = 80$ and $m = 48$, then we have $i_{max} = 18$ and the probability of obtaining 18 positions of known bits that collide is equal to $p_{max} = 2^{-2.72}$.

Amongst the values from U_{k-1}, we choose ${p_{max}}^{-1}$ random ones. Each of them is introduced in the ARMADILLO2 function parametrized with the keys K_1 and K_2. For each of the ${p_{max}}^{-1}$ pairs of values, we guess the bit at position $k-1$ of X_1 and of X_2 (for example 1 and 0 respectively since there is a difference on this bit position) and we end up with $2 \cdot {p_{max}}^{-1}$ pairs. Then, we can undo the last round of Q_X for the known bits from Y_1 and Y_2. We consider that a guess passes the test if it verifies the conditions on the number of colliding values on the colliding bit positions. For one of these $2 \cdot {p_{max}}^{-1}$ pairs (in our example $(Q_1^{-1}(Y_1), Q_0^{-1}(Y_2)))$, the number of colliding bit positions will be i_{max}. When this is the case, if the guess on the bit of X_1 and X_2 was incorrect, we have a probability of $2^{-i_{max}+1}$ to pass the test, while we will pass it with probability one if the guess was correct. Finally, we have determined one bit of each key K_1 and K_2 with a complexity of $2 \cdot {p_{max}}^{-1}$, which in our example would be $2^{3.72}$.

We can continue the process by considering $e = k-2$ and ${p_{max}}^{-1}$ values from U_{k-2} that have a key bit at position $k-1$. Following the same method as before, we will recover one key bit, i.e. the one at position $k-1$ in X when we have 18 colliding bits before the step $k-1$ of Q_X. Let us remark here that in practice we do not have to wait for having a collision on 18 bits, but most of the time collisions on a different number of bits will also be enough for determining if a guess passes the test or not. We can repeat this step in order to obtain the biggest possible number of key bits and determining each bit will add at most a complexity of ${p_{max}}^{-1}$.

The next steps depend on the number of bits that we have already determined. All in all, we conjecture that when both bit permutations behave like random ones, the complexity will not exceed $2 \cdot c \cdot {p_{max}}^{-1}$.

Conclusion

We have presented some new and practical analysis of ARMADILLO2. Notably a free-start and semi-free-start collision attacks for the full ARMADILLO2 hash functions. Extending this work to real collisions (i.e. with a predefined IV) might be possible but it is not very appealing because it is likely that several message blocks are required (all versions have $c > m$) and therefore the task of the cryptanalyst would be quite complex to handle. ARMADILLO2 should not be used in any security application since our attacks have a very low complexity. This work and the local-linearization method is a first step in order to evaluate the security of data-dependent bit transpositions cryptographic designs.

Acknowledgements

The authors would like to thank the anonymous referees and the ARMADILLO2 team for their helpful comments.

References

1. Mohamed Ahmed Abdelraheem, Céline Blondeau, María Naya-Plasencia, Marion Videau, and Erik Zenner. Cryptanalysis of ARMADILLO2. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 308–326. Springer, 2011.
2. Stéphane Badel, Nilay Dagtekin, Jorge Nakahara, Khaled Ouafi, Nicolas Reffé, Pouyan Sepehrdad, Petr Susil, and Serge Vaudenay. ARMADILLO: A Multi-purpose Cryptographic Primitive Dedicated to Hardware. In Stefan Mangard and François-Xavier Standaert, editors, *CHES*, volume 6225 of *LNCS*, pages 398–412. Springer, 2010.
3. Gilles Brassard, editor. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *LNCS*. Springer, 1990.
4. Ivan Damgård. A Design Principle for Hash Functions. In Brassard [3], pages 416–427.
5. Xuejia Lai and James L. Massey. A Proposal for a New Block Encryption Standard. In *EUROCRYPT*, pages 389–404, 1990.
6. Ralph C. Merkle. One Way Hash Functions and DES. In Brassard [3], pages 428–446.
7. Ronald L. Rivest. The RC5 Encryption Algorithm. pages 86–96. Springer-Verlag, 1995.
8. Ronald L. Rivest, M. J. B. Robshaw, and Yiqun Lisa Yin. RC6 as the AES, 2000.
9. Pouyan Sepehrdad, Petr Susil, and Serge Vaudenay. Fast Key Recovery Attack on ARMADILLO1 and Variants. In *CARDIS*, 2011.

A Implementation of the collision attacks for $k = 128$

We implemented all attacks for $k = 128$ and they require less than a second and negligible memory on an average computer (Intel Core2 Duo CPU @ 2.13 GHz) in order to find a collision. Since no specific σ_0 and σ_1 bit transpositions are defined for ARMADILLO2, we run the attack for many randomly chosen instances so as to ensure the soundness of our reasoning. We give here examples of (semi)-free-start collisions for ARMADILLO2 with a σ_0 and σ_1 bit transpositions instance that fulfill the criteria required in [2] for $k = 128$. Namely, we denote λ the second largest eigenvalue of the matrix $M = \frac{1}{4}(P_{\sigma_0} + P_{\sigma_0}^{128} + P_{\sigma_1} + P_{\sigma_1}^{128})$, then for the σ_0 and σ_1 instance found we have $\lambda = 0.87$. This means that there exists a distinguisher with advantage $\lambda^{256} = 2^{-51.4}$, while our attacks have much better advantage.

Free-start collision for ARMADILLO2 with $k = 128$, $c = 80$, $m = 48$:

$$\texttt{ARMADILLO2}(\texttt{ffffffffffffffffbfff}, \texttt{ffffffffffff}) = \texttt{dfb0d8f2b763ce97f785}$$

$$\texttt{ARMADILLO2}(\texttt{fffffdffffffffffbfff}, \texttt{ffffffffffff}) = \texttt{dfb0d8f2b763ce97f785}$$

Semi-free-start collision for ARMADILLO2 with $k = 128$, $c = 80$, $m = 48$:

$$\texttt{ARMADILLO2}(\texttt{6bc8c848de5ff533cd6f}, \texttt{0850b04b82e2}) = \texttt{26827e3d614d2fc75d64}$$
$$\texttt{ARMADILLO2}(\texttt{6bc8c848de5ff533cd6f}, \texttt{0850b04b82f0}) = \texttt{26827e3d614d2fc75d64}$$

Bit transpositions $\boldsymbol{\sigma_0}$ and $\boldsymbol{\sigma_1}$ used:

$$\begin{aligned}\sigma_0 = {} & 62, 98, 14, 114, 36, 77, 55, 3, 28, 88, 29, 122, 57, 90, 66, 52, 44, 22, 95, 118, 69, 86,\\ & 35, 56, 58, 82, 18, 97, 78, 21, 85, 101, 19, 65, 10, 6, 116, 121, 70, 99, 61, 102, 4, 91,\\ & 39, 119, 79, 16, 84, 50, 113, 45, 93, 104, 73, 112, 8, 5, 51, 9, 105, 46, 64, 94, 41, 54,\\ & 127, 67, 106, 23, 63, 49, 123, 15, 60, 81, 96, 72, 110, 37, 30, 89, 7, 92, 2, 68, 40, 32,\\ & 53, 11, 71, 26, 103, 59, 109, 111, 38, 74, 20, 48, 24, 43, 126, 117, 13, 124, 31, 33, 100,\\ & 125, 87, 27, 83, 128, 12, 42, 80, 107, 108, 17, 25, 120, 76, 75, 115, 47, 1, 34\end{aligned}$$

$$\begin{aligned}\sigma_1 = {} & 10, 60, 111, 78, 38, 57, 110, 75, 104, 56, 88, 79, 23, 99, 16, 22, 128, 94, 120, 24, 64, 3,\\ & 6, 55, 42, 51, 43, 82, 114, 89, 26, 35, 61, 73, 77, 36, 28, 21, 105, 15, 67, 70, 113, 65, 39,\\ & 80, 122, 31, 101, 100, 107, 124, 18, 46, 85, 19, 49, 14, 12, 71, 86, 68, 102, 91, 58, 95, 1,\\ & 53, 83, 125, 66, 98, 81, 44, 48, 59, 27, 9, 119, 40, 45, 74, 92, 112, 93, 69, 5, 108, 106,\\ & 115, 90, 13, 84, 126, 7, 109, 54, 127, 33, 121, 62, 87, 30, 29, 63, 2, 97, 116, 4, 47, 11,\\ & 8, 34, 96, 118, 72, 52, 103, 37, 25, 123, 50, 76, 17, 20, 41, 117, 32\end{aligned}$$

On the (In)Security of IDEA in Various Hashing Modes*

Lei Wei[1], Thomas Peyrin[1], Przemysław Sokołowski[2],
San Ling[1], Josef Pieprzyk[2], and Huaxiong Wang[1]

[1] Division of Mathematical Sciences, School of Physical and Mathematical Sciences,
Nanyang Technological University, Singapore
wl@pmail.ntu.edu.sg, thomas.peyrin@gmail.com
[2] Macquarie University, Australia

Abstract. In this article, we study the security of the IDEA block cipher when it is used in various simple-length or double-length hashing modes. Even though this cipher is still considered as secure, we show that one should avoid its use as internal primitive for block cipher based hashing. In particular, we are able to generate instantaneously free-start collisions for most modes, and even semi-free-start collisions, pseudo-preimages or hash collisions in practical complexity. This work shows a practical example of the gap that exists between secret-key and known or chosen-key security for block ciphers. Moreover, we also settle the 20-year-old standing open question concerning the security of the Abreast-DM and Tandem-DM double-length compression functions, originally invented to be instantiated with IDEA. Our attacks have been verified experimentally and work even for strengthened versions of IDEA with any number of rounds.

Key words: IDEA, block cipher, hash function, cryptanalysis, collision, preimage

1 Introduction

Hash functions are considered as a very important building block for many security and cryptography applications. Informally, a hash function H is a function that takes an arbitrarily long message as input and outputs a fixed-length hash value of size n bits. In cryptography, we want these functions to fulfill three security requirements, namely collision resistance and (second)-preimage resistance. It should be impossible for an adversary to find a collision (two different messages that lead to the same hash value) in less than $2^{n/2}$ hash computations, or a (second)-preimage (a message hashing to a given challenge) in less than 2^n hash computations. Most of nowadays hash functions divide the whole input message into blocks after padding it, and then process the blocks in an iterative way. A very known and utilised example is the Merkle-Damgaård algorithm [12, 33], which uses an n-bit compression function h in order to process the m message blocks M_i: $CV_{i+1} = h(CV_i, M_i)$, where CV_i is the n-bit internal state (or chaining variable) that is initialized by a fixed public value $CV_0 = IV$ and the final hash value is H_m. This algorithm is very interesting because it allows to reduce the collision/preimage security of the hash function to the collision/preimage security of the compression function. However, in order to guarantee the soundness of the construction, a designer must ensure that an attacker can not break the collision/preimage resistance of the compression function. One can identify different security properties for a compression function:

- *free-start collision*: in less than $2^{n/2}$ computations, find two different pairs $(CV, M) \neq (CV', M')$ such that they lead to the same compression function output value: $h(CV, M) = h(CV', M')$,

* The first, fourth and sixth authors are supported by the Singapore National Research Foundation under Research Grant NRF-CRP2-2007-03 and the first author is also supported by the Singapore Ministry of Education under Research Grant T206B2204 and by the NTU NAP Startup Grant M58110000. The second author is supported by the Lee Kuan Yew Postdoctoral Fellowship 2011 and the Singapore National Research Foundation Fellowship 2012.

- *semi-free-start collision*: in less than $2^{n/2}$ computations, find one chaining variable CV and two different message blocks $M \neq M'$ such that they lead to the same compression function output value: $h(CV, M) = h(CV, M')$,
- *preimage*: in less than 2^n computations, find one chaining variable CV and one message block M such that they lead to a given output challenge X: $h(CV, M) = X$.

Note that a semi-free-start collision for the compression function where the chaining variable CV is not chosen by the attacker directly leads to a collision for the whole hash function. In any case, a semi-free-start collision is very dangerous since it means that for some choices of IV, the attacker knows how to generate a collision. Even free-start collision are considered serious as they invalidate the collision resistance assumption on the compression function and we have seen many free-start collision attacks eventually turning into full hash collision attacks in the recent history (for example free-start collision attacks for MD5 were quickly identified [14], then upgraded to semi-free-start collision attacks [15] and eventually to full collision attacks [38]). As for preimage attacks on the compression function (also known as pseudo-preimages), they are very relevant since there exist a meet-in-the-middle algorithm that in most cases can turn them into a preimage attack for the full hash function.

The separation between a block cipher and a compression function has always been blurry. Constructions are known to turn the former into the latter [7, 36] or the latter into the former [31]. For example, the Davies-Meyer mode [1] converts a secure block cipher E into a secure compression function and is incorporated in a large majority of the currently known hash functions. While very satisfying solutions exist to transform a secure n-bit block cipher into an n-bit compression function (Davies-Meyer, Miyaguchi-Preneel, Matyas-Meyer-Oseas modes [1] or see [7, 36] for a systematic study of this problem), there is still a lot of research being actively conducted on double-block length compression functions (where the block cipher size is n bits and the compression function output size is $2n$), from simple-key block ciphers such as AES-128 or double-key such as AES-256 [11].

A major difference between the cryptanalysis of block ciphers and compression functions is that the attacker can fully control the inner behavior of the compression function. In other words, the attacker can use more efficiently the freedom degrees available on the input (i.e. the number of independent binary variables he has to determine). A new security model for block ciphers, the so-called *known-key model* [24], was recently proposed in order to fill the gap between these two situations. In this model, the secret key is known to the adversary and its goal is to distinguish the behavior of a random instance of the block cipher from the one of a random permutation by constructing a set of (plaintext, ciphertext) pairs satisfying an *evasive* property. Such a property is easy to check but impossible to achieve with the same complexity and a non-negligible probability using oracle accesses to a random permutation and its inverse. In general, these known-key attacks are not regarded as problematic when the block cipher is used in a classical “secret key” setting. Moreover, it is rare that such threats are extended to attacks on the compression function.

A potential candidate for hashing is the 64-bit block cipher IDEA [26, 39] that uses 128-bit keys. While a simple-length hashing mode would only provide a 64-bit hash output, insufficient for most of nowadays security applications, a double-block length construction (DBL) would allow 128-bit hash outputs which can be sufficient in some scenarios. As IDEA handles double-length keys, more freedom in the constructions is possible. In fact, the well known Abreast-DM and Tandem-DM modes were specifically created to perform hashing with IDEA (see page 2 and Section 6 of [39]). These modes were later studied in much details [16, 17, 28, 30], but the security they provide when instantiated with IDEA remains a 20-year-old standing open question. In classical “secret key” setting, IDEA has already been studied a lot [2–6, 9, 10, 13, 18] and is still considered as a secure cipher despite its age and despite the current best attack [5] that requires

2^{63} data (half the codebook) and 2^{114} computations to recover the secret key for IDEA reduced to 7.5 rounds over a total of 8.5 (the attack on the full cipher from [5] is very marginal with $2^{126.8}$ computations and the one from [22] requires 2^{126} computations and 2^{52} chosen plaintexts). One can also cite the work of [6], that exposes a weak key class of size 2^{64}. Note also that a first step towards analysis of IDEA in hashing mode was done in [21] where a 3-round chosen-key attack is described and in [9] where the authors show how to find a free-start near collision (only a subset of the output collides) when IDEA is plugged into the Hirose DBL mode [9] (and also a free-start collision if the internal constant c is controlled by the attacker).

Our contribution. In this paper, we study the security of the IDEA block cipher [26, 39] when plugged into various block cipher based compression function constructions, such as the classical Davies-Meyer mode [1], also DBL constructions such as Hirose [19, 20], Abreast-DM and Tandem-DM [27, 39], Peyrin *et al.* (II) [35] or MJH-Double [29]. Even if this cipher is still considered as secure in the classical "secret key" setting, its security remains an open problem in hashing mode. Depending on the IDEA-based hash construction, we show that an attacker can find free-start collisions instantaneously, preimages or semi-free-start collisions practically. For some modes, we even describe a method to compute collisions for the whole hash function. These attacks are based on weak keys utilisation, but in contrary to the "secret key" setting where the goal of the attacker is to exhibit the biggest weak key class possible, in hashing mode the goal is to find and exploit the weakest of all keys. We use the fact that the key 0 in IDEA is extremely weak, actually rendering the whole encryption process a T-function [23], already known as dangerous for building a hash function [34]. While weak-keys are already known to be dangerous for block cipher-based hash functions, our method use a novel and non-trivial almost half-involution property for IDEA. Even strengthened versions of the cipher with any number of rounds can be attacked with about the same complexities. This work is one more example that one has to be very careful when hashing with a block cipher that presents any weakness when the key is known or controlled by the attacker. In particular, one should strictly avoid the use of a block cipher for which weak keys exist, even if only a single weak key is known.

2 The IDEA block cipher

The International Data Encryption Algorithm (IDEA) is a 64-bit block cipher handling 128-bit keys and designed by Lai and Massey [26, 39] in 1990. While its use is reducing over the recent years, it remains deployed in practice and has not been broken yet despite its advanced age. It has a very simple design, performing 8.5 rounds composed of only 16-bit wide XOR, additions and multiplications. More precisely, one round is composed of three layers: first the key addition layer (denoted KA), a multiplication-addition layer (denoted MA) and a middle words switching layer (denoted S). For the eighth round, the switching is omitted.

Let X^i represent the 64-bit internal state of IDEA before application of the i-th round and we can view it as four 16-bit subwords $X^i = (X^i_1, X^i_2, X^i_3, X^i_4)$, with $1 \leq i \leq 9$. Also, $Y^i = (Y^i_1, Y^i_2, Y^i_3, Y^i_4)$ will stand for the intermediate internal state value of IDEA during the i-th round, right between the KA and the MA layers. We denote by $\oplus$ the bitwise XOR operation, by $\boxplus$ the addition modulo 2^{16} and by $\odot$ the multiplication modulo $2^{16}+1$, where the value 0 is considered as 2^{16} and vice-versa. Finally, $Z^i = (Z^i_1, Z^i_2, Z^i_3, Z^i_4, Z^i_5, Z^i_6)$ represents the six 16-bit subkeys used during the i-th round (only the first four subkeys for the last half round).

The KA layer simply incorporates four subkeys:

$$Y^i_1 = X^i_1 \odot Z^i_1, \qquad Y^i_2 = X^i_2 \boxplus Z^i_2, \qquad Y^i_3 = X^i_3 \boxplus Z^i_3, \qquad Y^i_4 = X^i_4 \odot Z^i_4.$$

The MA layer first computes $B = Z_6^i \odot ((Y_2^i \oplus Y_4^i) \boxplus (Z_5^i \odot (Y_1^i \oplus Y_3^i)))$ and $A = B \boxplus (Z_5^i \odot (Y_1^i \oplus Y_3^i))$. Then, after application of the S layer we have:

$$X_1^{i+1} = Y_1^i \oplus B, \qquad X_2^{i+1} = Y_3^i \oplus B, \qquad X_3^{i+1} = Y_2^i \oplus A, \qquad X_4^{i+1} = Y_4^i \oplus A.$$

All the subkeys are simply determined by choosing consecutive bits in the 128-bit master key according to the Table 2 given in Appendix A. Finally, ciphering the plaintext P with IDEA to obtain the ciphertext C is defined as: $C = \mathrm{KA} \circ \mathrm{S} \circ \{\mathrm{S} \circ \mathrm{MA} \circ \mathrm{KA}\}^8(P)$. Figure 1 provides a schematic view of one round of IDEA.

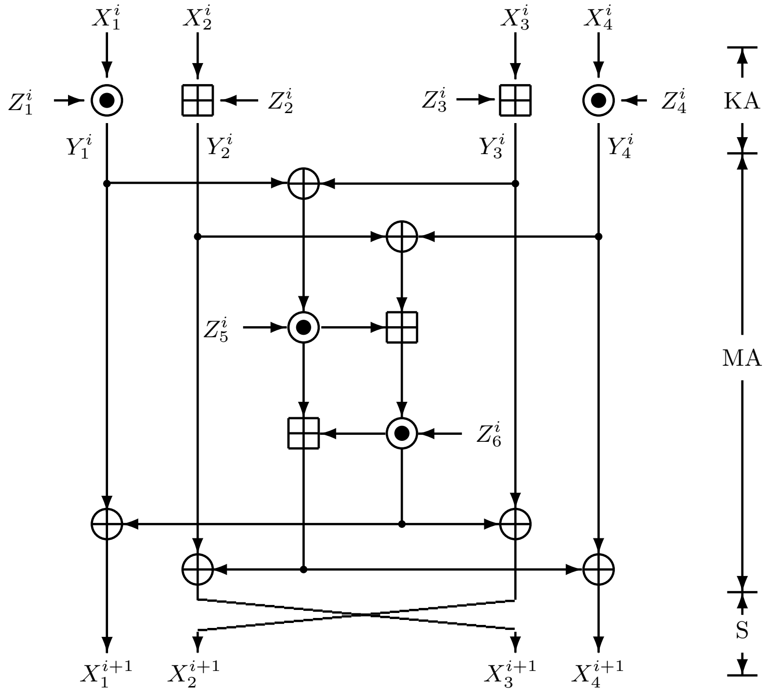


Fig. 1. One round of IDEA

Currently, the best cryptanalysis work published on IDEA [5] can reach 7.5 rounds with 2^{63} data (half the codebook) and 2^{114} computations. Concerning weak keys, the current biggest weak key class contains 2^{64} elements and has been published in [6].

3 Hashing with a double-length key block cipher

We will study the security of the various block cipher-based constructions that can use IDEA as the internal primitive. Therefore, we only consider the ones that use a double-key block cipher. More precisely, we denote $C = E_K(P)$ the process of ciphering the 64-bit plaintext P with IDEA using the 128-bit key K.

3.1 Simple-length compression function

A simple-length compression function construction with IDEA will provide a 64-bit output CV_{i+1}.

Davies-Meyer is the most usual simple-length mode [1] and it handles 128-bit message blocks: $CV_{i+1} = E_M(CV_i) \oplus CV_i$. Most standardized hash functions are actually implementing this mode, with an ad-hoc internal block cipher. While some weaknesses such as fixed-points are known, its security in terms of preimage and collision resistance have been studied and proved in the ideal cipher model [7]. Namely, we should expect at least 2^{32} and 2^{64} computations respectively to generate a (semi)-free-start collision or preimage for the compression function. Note that Miyaguchi-Preneel and Matyas-Meyer-Oseas simple-block length modes [1] are not considered in this article since they require the internal primitive to have the same block and key size, which is not the case for the `IDEA` block cipher.

3.2 Double-length compression function

A more interesting design strategy with `IDEA` would be to define double-block length constructions, in order to get 128-bit output, represented by two 64-bit words $CV1_i$ and $CV2_i$. This problem has already been studied a lot and remains a very active research domain, even when the internal primitive is a double-key block cipher.

Abreast-DM and Tandem-DM will of course be considered in this article since they both have been especially designed for `IDEA` [27, 39]. Tandem-DM handles a 64-bit message block M. We define $W = E_{CV1_i||M}(CV2_i)$ and then we have

$$\begin{aligned} CV1_{i+1} &= E_{M||W}(CV1_i) \oplus CV1_i, \\ CV2_{i+1} &= W \oplus CV2_i. \end{aligned}$$

Abreast-DM also handles a 64-bit message block M:

$$\begin{aligned} CV1_{i+1} &= E_{M||CV2_i}(\overline{CV1_i}) \oplus CV1_i, \\ CV2_{i+1} &= E_{CV1_i||M}(CV2_i) \oplus CV2_i, \end{aligned}$$

where $\overline{X}$ stands for the bitwise complement of X.

Hirose proposed a construction that contains two independent block cipher instances [19], later improved to only a single instance [20] by using a constant c to simulate the two independent ciphers:

$$\begin{aligned} CV1_{i+1} &= E_{CV2_i||M}(CV1_i) \oplus CV1_i, \\ CV2_{i+1} &= E_{CV2_i||M}(CV1_i \oplus c) \oplus CV1_i \oplus c. \end{aligned}$$

Peyrin *et al.* described in [35] a compression function (denoted Peyrin *et al.*(II)) that utilizes 5 calls to independent $3n$-to-n-bit compression functions, advising to be instantiated with double-key internal block ciphers such as `AES`-256 or `IDEA`. It handles two 64-bit message blocks $M1$ and $M2$:

$$\begin{aligned} CV1_{i+1} &= f_1(CV1_i, CV2_i, M1) \oplus f_2(CV1_i, CV2_i, M2) \oplus f_3(CV1_i, M1, M2), \\ CV2_{i+1} &= f_3(CV1_i, M1, M2) \oplus f_4(CV1_i, CV2_i, M1) \oplus f_5(CV2_i, M1, M2), \end{aligned}$$

where the functions f_i can be build for example by using the `IDEA` block cipher into a Davies-Meyer mode and we can simulate their independency by XORing distinct constants to the plaintext inputs, as it is done in [20]: $f_i(U, V, W) = E_{U||V}(W \oplus i) \oplus W$ (note that XORing the constants on the key input would be avoided in practice because it would lead to very frequent rekeying and therefore reduce the overall performance of the hash function). Since no real candidate was proposed by the authors, all possible position permutations of the three f_i inputs will be considered. Note that when cryptanalysing this scheme, we will attack the functions f_i independently. Thus, we will not use any weakness coming from potential dependencies between the functions f_i (apart of course that all 5 functions are based on IDEA).

MJH-Double is a rate 1 double-block length compression function recently published by Lee and Stam [29]. It uses a double-key block cipher and handles two 64-bit message blocks $M1$ and $M2$:

$$\begin{aligned}CV1_{i+1} &= E_{M2||CV2_i}(CV1_i \oplus M1) \oplus CV1_i \oplus M1,\\ CV2_{i+1} &= g \cdot (E_{M2||CV2_i}(f(CV1_i \oplus M1)) \oplus f(CV1_i \oplus M1)) \oplus CV1_i,\end{aligned}$$

where f is an involution with no fixed point and $g \neq 0, 1$ is a constant.

For all these double-block length proposals, the conjectured security is 2^{64} and 2^{128} computations respectively to generate a (semi)-free-start collision or preimage for the compression function or hash function. We summarize all of them in Appendix D.

4 Weak keys for IDEA

Weak keys for IDEA has already been studied in details [6, 10, 18], but what we are looking for is slightly different. Indeed, for block cipher cryptanalysis, since the attacker can not control the key input he looks for the biggest possible class of weak keys, so as to get the highest possible probability that a weak key will indeed be chosen. In the case of compression function cryptanalysis, the key input is fully known or even controlled by the attacker. The goal is therefore not to find the biggest possible class of weak keys, but to find the weakest possible key. As we will show for IDEA, even if only one weak key is found, its weakness might directly lead to successful attacks on the whole compression or hash function.

4.1 Analysis of the internal functions

When looking at the internal round function of IDEA, one might wonder what would be a weak key. In IDEA, the most annoying functions for the cryptanalyst are clearly the multiplications in $\mathbb{Z}_{2^{16}+1}$. Indeed, these operations are strongly non-linear and provide good diffusion between the different bit positions. On the contrary, XOR operations are linear and do not provide any diffusion between the bit positions, while the additions in $\mathbb{Z}_{2^{16}}$ can be easily approximated linearly and the diffusion between the bit positions only happens through the carry. Moreover, XOR and additions are even weaker in IDEA since no rotations are present, comparing with Addition-Rotation-XOR (ARX) designs. Here the rotation is done through the multiplications in $\mathbb{Z}_{2^{16}+1}$ and our goal is therefore to avoid them.

When adding $(a + b) \bmod 2^{16}$, we can avoid any diffusion by forcing one operand to 0. When multiplying $(a \odot b) = (a \cdot b) \bmod 2^{16} + 1$, the good diffusion will happen especially when $(a \cdot b) \geq 2^{16} + 1$. An easy way to avoid this is to fix one of the two operands to 1. In that case, we have $(a \odot 1) = (a \cdot 1) \bmod 2^{16} + 1 = a \bmod 2^{16}$. As already remarked in [10], a good choice is also 0, since

$$\begin{aligned}(a \odot 0) \bmod 2^{16} &= ((a \cdot 2^{16}) \bmod (2^{16} + 1)) \bmod 2^{16}\\ &= (((a \cdot 2^{16} + a) + (2^{16} + 1) - a) \bmod (2^{16} + 1)) \bmod 2^{16}\\ &= (0 + 2^{16} + 1 - a) \bmod 2^{16} = 1 - a \bmod 2^{16}\\ &= 2 + (2^{16} - 1 - a) \bmod 2^{16} = (2 + \overline{a}) \bmod 2^{16}\end{aligned}$$

and the multiplication is reduced to only a complement and an addition with a constant.

4.2 Weak keys classes

Based on the remark that the operand 0 is very weak for both multiplications and additions, Daemen *et al.* [10] generated a class of weak keys. A first obvious candidate is the null key (all bits set to zero), which will force all the subkeys to zero as well. As a consequence, all subkeys additions can be simply removed and all subkeys multiplications can be replaced by a complement (or XOR with `0xffff`) and an addition with value 2. At this point, all the operations in `IDEA` with null key are either XOR or additions. Therefore, by inserting differences only on the Most Significant Bit (MSB) of the four 16-bit plaintext input words, the attacker is ensured that only the MSB of the four output words will contain a difference. Even better, the mapping from an MSB input difference pattern to an MSB output difference pattern is completely deterministic (is it linear since on the MSB no carry is propagated). Such a property is largely sufficient to consider the null key as weak. This reasoning can be generalized by observing that the attacker does not necessarily need all subkeys to be null, but only the ones that are multiplied to an internal word which contains a MSB difference. Since the MSB differential paths are quite sparse, many of the null constraints on the subkeys are relaxed and one finally gets 2^{35} weak keys.

4.3 The null weak key

We show that the null key is particularly weak for hash function utilization. Even if other keys belong to a weak key class, they do not present the same special properties as the null key.

Almost half-involution When using the null key, we remark that all subkeys will be null as well. Then, all rounds layers will be the same and we write KA_0 and MA_0 the KA and MA layers with null subkeys. A nice practical feature of `IDEA` is that the decryption is done using the very same algorithm as encryption, but with different subkeys. The decryption subkeys for the MA layer are the same as the encryption ones since the MA layer is an involution (i.e. $\mathrm{MA}{=}\mathrm{MA}^{-1}$). The decryption subkeys for the KA layer are the respective multiplicative and additive inverses of the encryption subkeys. However, note that a null subkey is both its own multiplicative and additive inverse and the KA layer becomes an involution as well (i.e. $\mathrm{KA}_0{=}\mathrm{KA}_0^{-1}$). To summarize, using the null key, we are ensured that $\mathrm{KA}_0{=}\mathrm{KA}_0^{-1}$ and $\mathrm{MA}_0{=}\mathrm{MA}_0^{-1}$. Note that we trivially have $\mathrm{S}{=}\mathrm{S}^{-1}$.

Now, since the KA layer and S layer commute, `IDEA` with null key can be rewritten as

$$\begin{aligned} C &= \mathrm{KA}_0 \circ \mathrm{S} \circ \{\mathrm{S} \circ \mathrm{MA}_0 \circ \mathrm{KA}_0\}^8(P) \\ &= \mathrm{KA}_0 \circ \mathrm{S} \circ \{\mathrm{S} \circ \mathrm{MA}_0 \circ \mathrm{KA}_0\}^3 \circ \mathrm{S} \circ \mathrm{MA}_0 \circ \mathrm{KA}_0 \circ \{\mathrm{S} \circ \mathrm{MA}_0 \circ \mathrm{KA}_0\}^4(P) \\ &= \underbrace{\mathrm{KA}_0 \circ \mathrm{MA}_0 \circ \{\mathrm{S} \circ \mathrm{KA}_0 \circ \mathrm{MA}_0\}^3}_{\sigma^{-1}} \circ \underbrace{\mathrm{KA}_0 \circ \mathrm{S}}_{\theta} \circ \underbrace{\{\mathrm{MA}_0 \circ \mathrm{KA}_0 \circ \mathrm{S}\}^3 \circ \mathrm{MA}_0 \circ \mathrm{KA}_0}_{\sigma}(P) \end{aligned}$$

which eventually gives $C = \sigma^{-1} \circ \theta \circ \sigma(P)$. One can check that since KA_0, MA_0 and S are involutions, the operation denoted by σ^{-1} is indeed the inverse of the one denoted by σ. Thus, using the notation

$$P \xrightarrow{\sigma^{-1}} U \xrightarrow{\theta} V \xrightarrow{\sigma} C$$

where U and V are internal state values, we have

$$P \xleftarrow{\sigma} U \xrightarrow{\theta} V \xrightarrow{\sigma} C.$$

We will use this almost half-involution property in Section 6 to find free-start collisions and even hash function collisions for some `IDEA`-based constructions.

T-function: When using the null key, we have already described that all operations remaining are either XOR or additions. These operations are triangular functions [23] (or T-functions) in the sense that any output bit at position i only depends on the input bits located at a position i or lower. A composition of T-functions is itself a T-function, therefore the whole permutation defined by `IDEA` with the null key is a T-function. As shown in [34], this property might be very dangerous in a hash function design. We will explain in Section 7 how to exploit this weakness and compute preimages by guessing the input words bit layer by bit layer.

5 Simple collision attacks

As shown by Daemen *et al.* [10], when using the null key for the encryption process of `IDEA`, differences inserted uniquely on the MSB of the four 16-bit input plaintext words will lead to differences on the MSB of the four 16-bit output ciphertext words. Moreover, since this difference mapping is linear (the difference on the carry is not propagated further than the MSB), all possible differential characteristics have a differential probability 1. For example, we denote by $\delta_{MSB} = \texttt{0x8000}$ the 16-bit word with difference only on the MSB and by $\Delta_{MSB} = (\delta_{MSB}, \delta_{MSB}, \delta_{MSB}, \delta_{MSB})$ the 64-bit difference composed of 4 words with difference δ_{MSB}. Then, Δ_{MSB} propagates to itself with probability 1 through one round of `IDEA`, or through its last half-round. Therefore, we have with probability 1

$$\Delta_{MSB} \xrightarrow{\texttt{IDEA}_{K=0}} \Delta_{MSB}.$$

Note that instead of using δ_{MSB} only, one can generalize the input difference space and obtain other very good differential paths for the encryption of `IDEA` with the null key. However, we omit this generalization here since the methods described in later sections already provide much better attacks.

Davies-Meyer. Finding a free-start collision on Davies-Meyer mode instantiated with `IDEA` is very easy. Since the difference Δ_{MSB} is mapped to itself through the `IDEA` encryption process with the null key, the attacker only has to pick $M = 0$. Then, any value of CV with difference Δ_{MSB} applied to it will lead to a collision with probability 1. We give in Appendix C.1 examples of such a free-start collision.

Hirose. The same method as for Davies-Meyer mode can be applied to the Hirose mode in order to find free-start collisions. The attacker fixes $CV2 = 0$ and $M = 0$ so as to force the null key to both encryptions. Then, any value of $CV1$ with a difference Δ_{MSB} applied to it will lead to a collision with probability 1, since Δ_{MSB} will appear on the plaintext input of both encryptions with the null key. We give in Appendix C.3 examples of such a free-start collision.

Abreast-DM. This technique seems impossible to apply to the Abreast-DM mode since forcing a difference Δ_{MSB} on any of the two encryptions plaintext input will imply a difference inserted in the key input of the other encryption block. Therefore, one cannot use Δ_{MSB} difference on plaintext input with null key in both encryption blocks. Even if the attacker tries to attack only one encryption block with this method, the other block will not be controlled and he will have to deal with random differences on its output. These random differences cannot be dealt with some birthday technique because fixing all inputs of one encryption block will fix all inputs of the other one as well.

Tandem-DM. This technique seems impossible to apply to the Tandem-DM mode for the exact same reasons as for Abreast-DM.

Peyrin *et al.*(II). We have to separate in two groups the possible instances of this construction, obtained by permuting the position of the three inputs of each internal function f_i. If all compression function inputs $CV1$, $CV2$, $M1$ and $M2$ appear in at least one of the `IDEA` key inputs of any f_i internal function, then the attack will not apply. Indeed, since all inputs will be involved at least one time, the attacker will necessarily have to insert a difference in at least one `IDEA` key input and he will not be able to use the differential path with probability 1. Note that these instances would be avoided in practice because they would lead to more frequent re-keying and therefore reduce the overall performance of the hash function. If this condition is not met, then we can apply the following free-start collision attack. Let $X \in \{CV1, CV2, M1, M2\}$ denote the input that is missing in all the `IDEA` key inputs of the compression function. The attacker simply fixes the difference Δ_{MSB} on X (one can give any value to X) and all other inputs are set to 0 in order to get the null key in every internal `IDEA`. The attacker ends up with several Davies-Meyer in parallel, with either no difference at all or with null key and Δ_{MSB} as plaintext input difference. Thus, he obtains a collision with probability 1. If $X \notin \{CV1, CV2\}$, then this attack finds semi-free-start collisions.

MJH-Double. The MJH-Double mode prevents this simple attack since even if we fix $CV2 = 0$ and $M2 = 0$ in order to get the null key in both encryptions, it is hard to force the difference Δ_{MSB} on both their plaintext inputs. Indeed, the f operation will randomize the difference and in order for the attack to run, we would require $\Delta_{MSB} \xrightarrow{f} \Delta_{MSB}$ which is unlikely to happen.

6 Improved collision attacks

In this section, using the almost half-involution property with the null key, we will show how to get the same difference on the input and on the output of the `IDEA` ciphering process with good probability. Then, we will use this weakness to derive our collision attacks, for any number of rounds.

6.1 Exploiting the almost half-involution

We have already shown in Section 4 that when the key is null, `IDEA` encryption process can be rewritten as

$$P \xleftarrow{\sigma} U \xrightarrow{\theta} V \xrightarrow{\sigma} C$$

where

$$\sigma = \{\mathrm{MA}_0 \circ \mathrm{KA}_0 \circ \mathrm{S}\}^3 \circ \mathrm{MA}_0 \circ \mathrm{KA}_0 \qquad \text{and} \qquad \theta = \mathrm{KA}_0 \circ \mathrm{S}.$$

We denote ΔU the XOR difference between two 64-bit internal state values U and U', i.e $\Delta U = U \oplus U'$, and δU_i represents the 16-bit difference on the i-th word of ΔU, that is $\Delta U = (\delta U_1,\ \delta U_2,\ \delta U_3,\ \delta U_4)$. Let us consider two random 64-bit internal state values U and U' such that $\delta U_2 = \delta U_3$ and we denote this 16-bit difference δ_M. For truly random values U and U', this condition happens with probability 2^{-16}. One can check that applying θ on U and U' to obtain V and V' respectively will lead to $\delta V_2 = \delta V_3 = \delta_M$ since layer S only switches the two middle words and layer KA_0 has no effect on them (addition of null subkeys).

Let δ_L and δ_R represent the difference on δU_1 and δU_4 respectively, i.e. $\Delta U = (\delta_L,\ \delta_M,\ \delta_M,\ \delta_R)$. Applying function θ to U and U', we would like the same differences to appear on internal state V and V': $\Delta V = (\delta_L, \delta_M, \delta_M, \delta_R)$. The previous condition with probability 2^{-16}

already ensures the two middle differences being the same δ_M. Concerning differences δ_L and δ_R, they will both be unaffected by layer S, but they might be modified through layer KA_0 that applies a multiplication with a null subkey. Therefore, we need to study the probability that a random difference δ is mapped to itself through a multiplication by the null subkey. We show in Appendix B that this probability is equal to $2^{-1.585}$ and finally we have $\Pr[(\delta_L, \delta_M, \delta_M, \delta_R) \xrightarrow{\theta} (\delta_L, \delta_M, \delta_M, \delta_R)] = 2^{-3.17}$.

At this point, we proved that for randomly chosen internal state values U and U', we will observe with probability $2^{-19.17}$ the same difference on U and V, i.e. $\Delta U = \Delta V$.

One can see that computing backward from internal states U to P or forward from V to C, the function σ is applied. Our final goal is to have the same difference on P and C. However, this seems unlikely to happen since U and V have different values, the forward and backward computations of σ should be completely unrelated, even with the same input difference. Yet, this reasoning does not take in account the fact that while U and V have distinct values, they are far from being independent: $V = \theta(U)$ with θ being a very light function. Moreover, we remarked that almost each time that we got the same difference on P and C, the same differences were observed as well in all rounds of the forward and backward σ computations (the round success probability increasing with the number of rounds already processed). Because all the rounds are not independent and because U and V are strongly related, it is very difficult to compute theoretically the probability of observing the same difference on P and C and we leave this as an open problem. Therefore, we measured it by choosing random values of U, δ_L, δ_M, δ_R, computing $V = \theta(U)$, and checking for collisions on the difference of P and C. The probability obtained was $2^{-16.26}$ for about 2^{28} tests (note that this probability somehow contains the $2^{-3.17}$ probability computed previously, but we can not separate them because the two events are not independent).

To conclude, the probability that two randomly chosen internal state values U and U' give the same difference on P and C is equal to $2^{-16-16.26} = 2^{-32.26}$ (instead of 2^{-64} expected for a random function). In other words, using the birthday paradox, one can find such a pair with about $2^{16.13}$ computations.

Interestingly, we have observed that most of the pairs fulfilling the differential path for the full IDEA will also be valid for a strengthened version of the cipher with any number of additional rounds. Since the subkeys are always null, strengthening the cipher would mean that $\sigma = \{\mathrm{MA}_0 \circ \mathrm{KA}_0 \circ \mathrm{S}\}^t \circ \mathrm{MA}_0 \circ \mathrm{KA}_0$ for any $t > 3$. We checked that the probability that two randomly chosen internal state values U and U' give the same difference on P and C tends to $2^{-32.54}$ when t tends to infinite. Thus, similarily to the method presented in the previous section, the attacks using this almost half-involution property will work for any number of rounds.

6.2 Improving collision attacks

Davies-Meyer. A first obvious application of having the same difference in P and C is collision search on Davies-Mayer mode, where the feed-forward will cancel the two differences in the output. The attack finds collisions for the whole hash function and the procedure is very simple: we start from the IV and add random differences in the first message block M_0. This will cause random differences in the the first chaining variable CV_1. For the second message block M_1, we will set all its bits 0 ($M_1 = 0$), forcing the internal IDEA computation to use the null key. Since we estimated in the previous section that with the null key a random pair of inputs has a probability $2^{-32.26}$ to give the same input/output difference, one can use the birthday paradox to generate a collision on CV_2 with only $2^{16.13}$ distinct message blocks M_0. We give in Appendix C.2 examples of hash collisions for the Davies-Meyer mode. Note that finding semi-free-start collisions with this technique is impossible since we would have to insert differences in the message input, which forbids the use of the null key in the internal cipher.

Hirose. We already showed how to find free-start collisions for the Hirose mode. However, finding semi-free-start collisions with this technique is impossible since we would have to insert differences in the message input, which forbids the use of the null key in the internal cipher. Also, concerning hash collisions, it seems hard as well because forcing the null key during iteration i requires us to obtain a chaining variable $CV2_{i-1} = 0$ during the previous iteration. This half-preimage already costs the same complexity as a generic collision search on the entire compression function.

Abreast-DM. One can derive a free-start collision attack for the Abreast-DM compression function using this technique. The attacker first fixes $CV1 = 0$ and $M = 0$. Then, he builds a set of $2^{48.13}$ distinct values $CV2$ and checks if a pair of this set leads to a collision. The probability that a pair leads to a collision on the first (top) branch is $2^{-32.26}$ (since the internal cipher on this part has the null key), and 2^{-64} on the other half. Overall, using the birthday paradox on the set of $2^{48.13}$ values $CV2$ is sufficient to have a good chance to obtain a collision. Note that finding a semi-free-start collision for the compression function or a collision for the hash function seems impossible with this method, for the same reasons as the Hirose mode.

Tandem-DM. The situation of Tandem-DM is absolutely identical to the Abreast-DM one: one can find free-start collisions for compression function using this technique. The attacker first fixes $CV1 = 0$ and $M = 0$. Then, he builds a set of $2^{48.13}$ distinct values $CV2$ and checks if a pair of this set leads to a collision. The probability that a pair leads to a collision on the first (top) branch is $2^{-32.26}$ (since the internal cipher on this part has the null key), and 2^{-64} on the other half. Overall, using the birthday paradox on the set of $2^{48.13}$ values $CV2$ is sufficient to have a good chance to obtain a collision. Again, finding a semi-free-start collision for the compression function or a collision for the hash function seems impossible with this method, for the same reasons as the Hirose mode.

Peyrin *et al.*(II). We showed in previous section how to find (semi)-free-start collisions with probability 1 for a certain subset of Peyrin *et al.*(II) constructions, but here we provide attacks on a bigger subset. If all compression function inputs $CV1$, $CV2$, $M1$ and $M2$ appear in at least one of the IDEA key inputs of f_1, f_2, f_3 (left side) and in at least one of the IDEA key inputs of f_3, f_4, f_5 (right side), then the attack will not apply. Indeed, for both left side and right side of the compression function, the attacker will necessarily have to insert a difference in at least one key input (since all inputs will be involved) and he will not be able to use the null key completely. Note that these instances would be avoided in practice because they would lead to more frequent rekeying and therefore reduce the overall performance of the hash function. However, if this condition is not met, then we can apply the following free-start collision attack. Let $X \in \{CV1, CV2, M1, M2\}$ denote the input that is missing in all the IDEA key inputs of f_1, f_2, f_3 (wlog the reasoning is the same with f_3, f_4, f_5). The attacker first fixes all inputs but X to 0 in order to get the null key in every internal IDEA on the left side. Then he chooses $2^{48.13}$ random values for X and checks among them if any pair collides on the whole compression function output. Since he has a probability $2^{-32.26}$ to get a collision on the left side and 2^{-64} on the right side, using a birthday search the attacker finds a solution with complexity $2^{48.13}$. Again, if $X \notin \{CV1, CV2\}$, then this attack finds semi-free-start collisions. However, finding a collision for the hash function seems impossible with this method, because at least one of the chaining variable inputs $CV1$ and $CV2$ will be present as key input for one of the IDEA internal emcryption. Setting this word to 0 is equivalent to a half-preimage that already costs the same complexity as a generic collision search on the entire hash function.

MJH-Double. One can derive a semi-free-start collision attack on the MJH-Double compression function instantiated with IDEA. The attacker first fixes $CV2 = 0$ and $M2 = 0$ and this will force the null key in both encryptions. Now he chooses a random value for $CV1$ (note that actually this value could be fixed by the challenger) and builds a set of $2^{32.26}$ values $M1$. In this configuration, it is easy to see that one will have random differences on the plaintext inputs to both encryptions. Since the null key is used for both, we have a probability $2^{-64.52}$ that a pair of $M1$ leads to a collision after the feed-forward of both encryptions (on the output of the bottom block and just before the application of g on the top block). Therefore, with a birthday technique, one can find such a pair with only $2^{32.26}$ computations. Note that while this pair will directly lead to a collision on the bottom $CV1$ output, the difference on $M1$ is injected two times before computing the top $CV2$ output. Two times of the same difference will cancel themselves and we eventually get a full semi-free-start collision. Note that it seems hard to extend this attack to a hash collision since the attacker would require to force the incoming chaining variable $CV2$ to be equal to 0 and this half-preimage already costs the same complexity as a generic collision search on the entire hash function.

7 Preimage attacks

We showed in Section 4 that if used with the null key, the whole permutation defined by IDEA is a T-function. Since any output bit at position i only depends on the input bits located at a position i or lower, we reuse the idea of preimage attack for hash functions based on T-functions [34] where the preimage is computed bit layer by bit layer, starting from the LSB. However, here our situation is different than the functions studied in [34] since we do not have any truncation or reduction of the internal state at the end of the process.

We denote by p the probability that given a random challenge, our algorithm outputs a preimage for this challenge. We denote by s the average number of preimage solutions that the algorithm will output, given that at least one is found. The average number of solutions outputted by our algorithm is then $A = s \cdot p$. For an n-bit ideal compression function, a generic attack restricted to C computations can generate $A = C \cdot 2^{-n}$ solutions on average. Thus, we can consider that a preimage attack is found if we exhibit an algorithm that outperforms this generic complexity.

Davies-Meyer. Since the key is fixed to 0 and since the plaintext and ciphertext sizes are the same, we trivially have that $A = 1$. We measured[3] that $p = 2^{-17.50}$, thus we directly deduce that $s = A/p = 2^{17.5}$. A straightforward implementation is a recursive depth first search, attacking the T-function by bit layer from the LSB to the MSB of the 16-bit state words. Wrong candidates at lower layers are discarded thanks to an early-abort strategy. On average, the amount of IDEA encryptions required to find all the possible preimages (if at least one can be found) can be estimated as $C \simeq 16 \cdot 2^4 \cdot s = 2^{25.5}$, since we have 16 bit layers, each having 4 bits of input, and on average the number of candidates in one layer is s. This is a very conservative estimation since only $p = 2^{-17.50}$ of the challenges on average will eventually lead to a solution and the early-abort strategy will make the actual search of very low complexity. In the ideal case, with $C = 2^{25.5}$ computations allowed, an attacker should only be able to generate

[3] from 2^{31} random challenges, we measured that $p = 2^{-17.50}$ and $s = 2^{17.74}$.

$A = 2^{25.5-64} = 2^{-38.5}$ solutions on average for an ideal 64-bit compression function. We give an example of a preimage in Appendix C.4.

Hirose. We can reuse the attack on Davies-Meyer, but only one of the two branches will be controlled, with the other behaving randomly. We first find a preimage for the first branch (with probability $2^{-17.5}$) and then use the $2^{17.5}$ solutions on average to also match the second branch (with probability $2^{17.5-64} = 2^{-46.5}$). Therefore, our preimage search algorithm have parameters $p = 2^{-17.5-46.5} = 2^{-64}$ and $s = 1$, while the average number of preimage solutions found is $A = 2^{-64}$. The complexity of the search is equivalent to the Davies-Meyer case, $C = 2^{25.5}$. For an attacker using at most $2^{25.5}$ computations on an ideal 128-bit compression function, the average number of solutions he could find is only $2^{-102.5}$.

Abreast-DM. Similarly to Hirose, by setting for example $M = CV1 = 0$, one can attack one branch bit layer by bit layer while the other branch will behave randomly. The complexity analysis is identical to Hirose's case.

Tandem-DM. Similarly to Hirose, by setting $M = CV1 = 0$, one can attack one branch bit layer by bit layer while the other branch will behave randomly. The complexity analysis is identical to Hirose's case.

Peyrin *et al.*(II). If all compression function inputs $CV1$, $CV2$, $M1$ and $M2$ appear in at least one of the IDEA key inputs of f_1, f_2, f_3 (left side) and in at least one of the IDEA key inputs of f_3, f_4, f_5 (right side), then the attack will not apply (because the attacker will not be able to use the null key completely). Otherwise, similarly to Hirose, by setting all IDEA keys to 0 on one side, one can attack it bit layer by bit layer while the other side will behave randomly. The complexity analysis is identical to Hirose's case.

MJH-Double. The attacker first fixes $M2 = CV2 = 0$ so as to get the null key for both IDEA encryptions. Then, similarly to the Davies-Meyer case, he find a preimage with probability $p = 2^{-17.5}$ for one of the two sides and this defines the value of $M1 \oplus CV1$. In order to get the preimage on the second side as well, the attacker only has to modify the value of $M1$ accordingly. If a solution is found on the first side, the attacker therefore gets $s = 2^{17.5}$ preimages. On average, he finds $A = 1$ solutions and the complexity is again $2^{25.5}$ computations. For an attacker using at most $2^{25.5}$ computations on an ideal 128-bit compression function, the average number of solutions he should find is only $2^{-102.5}$.

8 Results and implementations

We depict in Table 1 our results for the block cipher to compression function modes considered in this article when instantiated with IDEA. We implemented all attacks of reasonable complexities and provide in Appendix C the collision/preimage examples obtained.

Conclusion

In this article, we showed collision and preimage attacks for several single and double-length block cipher based compression function constructions when instantiated with the block cipher IDEA. Namely, we analyzed all known double-key schemes such as Davies-Meyer, Hirose, Abreast-DM, Tandem-DM, Peyrin *et al.* (II) and MJH-Double. While most of these constructions are

Table 1. Summary of results for block cipher to compression function modes when instantiated with `IDEA` (we did not include MDC-2 as it does not provide ideal collision resistance). The preimage complexity results find s preimages on average with a certain probability p, for a total average of $A = s \cdot p$ solutions. The results for Peyrin *et al.*(II) construction, marked with a *, depend on the instance considered (see relevant parts of Sections 5, 6 and 7 for more details).

Mode	hash output size	compression function			hash function collision attack
		free-start collision attack	semi-free-start collision attack	preimage attack complexity (s, p)	
Davies-Meyer [1]	64	2^{1}		$2^{25.5}$ $(2^{17.5}, 2^{-17.5})$	$2^{16.13}$
Hirose [19, 20]	128	2^{1}		$2^{25.5}$ $(1, 2^{-64})$	
Abreast-DM [27, 39]	128	$2^{48.13}$		$2^{25.5}$ $(1, 2^{-64})$	
Tandem-DM [27, 39]	128	$2^{48.13}$		$2^{25.5}$ $(1, 2^{-64})$	
Peyrin *et al.*(II) [35]	128	2^{1} / $2^{48.13\star}$	2^{1} / $2^{48.13\star}$	$2^{25.5}$ $(1, 2^{-64})^{\star}$	
MJH-Double [29]	128	$2^{32.26}$	$2^{32.26}$	$2^{25.5}$ $(2^{17.5}, 2^{-17.5})$	

conjectured or proved to be secure in the ideal cipher model, we showed that their security is very weak when instantiated with the block cipher `IDEA`, which remains considered as secure in the secret key model. In particular, we answer in the negative for the 20-year-old standing open question concerning the security of the Abreast-DM and Tandem-DM instantiated with `IDEA`. All our practical attacks have been implemented and they can work even for any number of `IDEA` rounds. Our results indicate that one has to be very careful when hashing with a block cipher that presents any weakness when the key is known or controlled by the attacker. Also, since we extensively use the presence of weak-keys for `IDEA`, as a future work it would be interesting to look at the security of hash functions based on block ciphers for which some key sets are known to be weaker than others.

Acknowledgments

The authors would like to thank the anonymous referees for their helpful comments.

References

1. A. Menezes, P. van Oorschot, and S. Vanstone. CRC-Handbook of Applied Cryptography. CRC Press, 1996.
2. Eyüp Serdar Ayaz and Ali Aydin Selçuk. Improved DST Cryptanalysis of IDEA. In Eli Biham and Amr M. Youssef, editors, *Selected Areas in Cryptography*, volume 4356 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2006.
3. Eli Biham, Orr Dunkelman, and Nathan Keller. New Cryptanalytic Results on IDEA. In Lai and Chen [25], pages 412–427.
4. Eli Biham, Orr Dunkelman, and Nathan Keller. A New Attack on 6-Round IDEA. In Alex Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 211–224. Springer, 2007.
5. Eli Biham, Orr Dunkelman, Nathan Keller, and Adi Shamir. New Data-Efficient Attacks on Reduced-Round IDEA. Cryptology ePrint Archive, Report 2011/417, 2011.
6. Alex Biryukov, Jorge Nakahara Jr., Bart Preneel, and Joos Vandewalle. New Weak-Key Classes of IDEA. In Robert H. Deng, Sihan Qing, Feng Bao, and Jianying Zhou, editors, *ICICS*, volume 2513 of *Lecture Notes in Computer Science*, pages 315–326. Springer, 2002.
7. John Black, Phillip Rogaway, and Thomas Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2002.
8. Gilles Brassard, editor. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *LNCS*. Springer, 1990.

9. Donghoon Chang. Near-Collision Attack and Collision-Attack on Double Block Length Compression Functions based on the Block Cipher IDEA. Cryptology ePrint Archive, Report 2006/478, 2006. `http://eprint.iacr.org/`.
10. Joan Daemen, René Govaerts, and Joos Vandewalle. Weak Keys for IDEA. In Stinson [37], pages 224–231.
11. Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.
12. Ivan Damgård. A Design Principle for Hash Functions. In Brassard [8], pages 416–427.
13. Hüseyin Demirci, Ali Aydin Selçuk, and Erkan Türe. A New Meet-in-the-Middle Attack on the IDEA Block Cipher. In Matsui and Zuccherato [32], pages 117–129.
14. Bert den Boer and Antoon Bosselaers. Collisions for the Compressin Function of MD5. In *EUROCRYPT*, pages 293–304, 1993.
15. Hans Dobbertin. Cryptanalysis of MD5 compress. Presented at the rump session of EUROCRYPT 1996, 1996.
16. Ewan Fleischmann, Michael Gorski, and Stefan Lucks. On the Security of Tandem-DM. In Orr Dunkelman, editor, *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 84–103. Springer, 2009.
17. Ewan Fleischmann, Michael Gorski, and Stefan Lucks. Security of Cyclic Double Block Length Hash Functions including Abreast-DM. Cryptology ePrint Archive, Report 2009/261, 2009. `http://eprint.iacr.org/`.
18. Philip Hawkes. Differential-Linear Weak Key Classes of IDEA. In *EUROCRYPT*, pages 112–126, 1998.
19. Shoichi Hirose. Provably Secure Double-Block-Length Hash Functions in a Black-Box Model. In Choonsik Park and Seongtaek Chee, editors, *ICISC*, volume 3506 of *Lecture Notes in Computer Science*, pages 330–342. Springer, 2004.
20. Shoichi Hirose. Some Plausible Constructions of Double-Block-Length Hash Functions. In Matthew J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 210–225. Springer, 2006.
21. John Kelsey, Bruce Schneier, and David Wagner. Key-Schedule Cryptoanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 1996.
22. Dmitry Khovratovich, Gaetan Leurent, and Christian Rechberger. Narrow Bicliques: Cryptanalysis of Full IDEA. In *EUROCRYPT*, 2012, to appear.
23. Alexander Klimov and Adi Shamir. Cryptographic Applications of T-Functions. In Matsui and Zuccherato [32], pages 248–261.
24. Lars R. Knudsen and Vincent Rijmen. Known-Key Distinguishers for Some Block Ciphers. In Kaoru Kurosawa, editor, *ASIACRYPT*, volume 4833 of *LNCS*, pages 315–324. Springer, 2007.
25. Xuejia Lai and Kefei Chen, editors. *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*, volume 4284 of *Lecture Notes in Computer Science*. Springer, 2006.
26. Xuejia Lai and James L. Massey. A Proposal for a New Block Encryption Standard. In *EUROCRYPT*, pages 389–404, 1990.
27. Xuejia Lai and James L. Massey. Hash Function Based on Block Ciphers. In *EUROCRYPT*, pages 55–70, 1992.
28. Jooyoung Lee and Daesung Kwon. The Security of Abreast-DM in the Ideal Cipher Model. Cryptology ePrint Archive, Report 2009/225, 2009. `http://eprint.iacr.org/`.
29. Jooyoung Lee and Martijn Stam. MJH: A Faster Alternative to MDC-2. In Aggelos Kiayias, editor, *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 213–236. Springer, 2011.
30. Jooyoung Lee, Martijn Stam, and John P. Steinberger. The Collision Security of Tandem-DM in the Ideal Cipher Model. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 561–577. Springer, 2011.
31. Michael Luby and Charles Rackoff. How to Construct Pseudorandom Permutations from Pseudorandom Functions. *SIAM J. Comput.*, 17(2):373–386, 1988.
32. Mitsuru Matsui and Robert J. Zuccherato, editors. *Selected Areas in Cryptography, 10th Annual International Workshop, SAC 2003, Ottawa, Canada, August 14-15, 2003, Revised Papers*, volume 3006 of *Lecture Notes in Computer Science*. Springer, 2004.
33. Ralph C. Merkle. One Way Hash Functions and DES. In Brassard [8], pages 428–446.
34. Frédéric Muller and Thomas Peyrin. Cryptanalysis of T-Function-Based Hash Functions. In Min Surp Rhee and Byoungcheon Lee, editors, *ICISC*, volume 4296 of *Lecture Notes in Computer Science*, pages 267–285. Springer, 2006.
35. Thomas Peyrin, Henri Gilbert, Frédéric Muller, and Matthew J. B. Robshaw. Combining Compression Functions and Block Cipher-Based Hash Functions. In Lai and Chen [25], pages 315–331.
36. Bart Preneel, René Govaerts, and Joos Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In Stinson [37], pages 368–378.
37. Douglas R. Stinson, editor. *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*. Springer, 1994.

38. Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 19–35. Springer, 2005.
39. Xuejia Lai. On the Design and Security of Block Ciphers. Hartung-Gorre Verlag, Konstanz, 1992.

A The IDEA subkeys

i-th round	$Z_1^{(i)}$	$Z_2^{(i)}$	$Z_3^{(i)}$	$Z_4^{(i)}$	$Z_5^{(i)}$	$Z_6^{(i)}$
1	0-15	16-31	32-47	48-63	64-79	80-95
2	96-111	112-127	25-40	41-56	57-72	73-88
3	89-104	105-120	121-8	9-24	50-65	66-81
4	82-97	98-113	114-1	2-17	18-33	34-49
5	75-90	91-106	107-122	123-10	11-26	27-42
6	43-58	59-74	100-115	116-3	4-19	20-35
7	36-51	52-67	68-83	84-99	125-12	13-28
8	29-44	45-60	61-76	77-92	93-108	109-124
OT	22-37	38-53	54-69	70-85		

Table 2. Key bits used for subkeys $Z_j^{(i)}$ in the i-th round of IDEA

B Proof of difference preservation through multiplication with a null subkey

We prove in this section that for randomly chosen values a and a' with $a \oplus a' = \delta$, the probability that the difference δ is preserved after multiplication by the null subkey is equal to $2^{-1.585}$. The condition we expect can be translated into the following equation

$$\delta = a \oplus a' = (a \odot 0) \oplus (a' \odot 0).$$

Since the $\odot$ operation is equivalent to a complement (or XOR with `0xffff`) and an addition with value 2, we can rewrite

$$\begin{aligned}
\delta &= ((a \oplus \mathtt{0xffff}) + 2) \oplus ((a' \oplus \mathtt{0xffff}) + 2)\\
\delta &= ((a \oplus \mathtt{0xffff}) + 2) \oplus ((a \oplus \delta \oplus \mathtt{0xffff}) + 2)\\
\delta &= (b + 2) \oplus ((b \oplus \delta) + 2)\\
\delta \oplus (b + 2) &= (b \oplus \delta) + 2
\end{aligned}$$

where $b = a \oplus \mathtt{0xffff}$. One can check that the least significant bit condition of this equation is always fulfilled.

If the second least significant bit of b is 0 (probability 1/2), then $(b + 2) = b \oplus 2$ and the equation is fulfilled if and only if the second least significant bit of $(b \oplus \delta)$ is also 0 (probability 1/2). Overall, this situation happens with probability 1/4.

If the second least significant bit of b is 1 (probability 1/2), then we will have a carry propagating and we require the second least significant bit of $(b \oplus \delta)$ to be also 1 (probability 1/2). If the third least significant bit of b is 0 (probability 1/2), then $(b + 2) = b \oplus 6$ and the equation is fulfilled if and only if the third least significant bit of $(b \oplus \delta)$ is also 0 (probability 1/2). Overall, this situation happens with probability $(1/4)^2$.

Continuing this reasoning over all the bits layers, we obtain that the success probability is equal to

$$\sum_{i=1}^{14}(1/4)^i = 2^{-1.585}.$$

C Collision and preimage examples

C.1 Free-start collision for Davies-Meyer mode

CV_i : `0x9efc 0x14ef 0x85d6 0xc557`
CV_i' : `0x1efc 0x94ef 0x05d6 0x4557`

$M = M'$: `0`

$CV_{i+1} = H(CV_i, M)$: `0x7f11 0x83f1 0x7617 0x8af3`
$CV_{i+1}' = H(CV_i', M')$: `0x7f11 0x83f1 0x7617 0x8af3`

C.2 Hash function collision for Davies-Meyer mode

We use as initial value the first 64 output bits of the `SHA-2` computation of the string “IDEA”:

`SHA-2`(”$IDEA$”) = ”`9f8c7b26cde59ca3dacc74ec7afda737ac1d15aa5239206416f79019dbd7ec37`”

IV: $IV_1 =$ `0x9f8c`, $IV_2 =$ `0x7b26`, $IV_3 =$ `0xcde5`, $IV_4 =$ `0x9ca3`

M_1: `0xdacc 0xdacc 0xdacc 0xdacc 0xdacc 0xdacc 0xcadc 0x0282`
M_1': `0xdacc 0xdacc 0xdacc 0xdacc 0xdacc 0xdacc 0xcade 0x1a3f`

$CV_1 = H(IV, M_1)$: `0xb782 0x4583 0x83b6 0x0bef`
$CV_1' = H(IV, M_1')$: `0x1ce2 0x8553 0xe656 0x4387`

$CV_2 = H(CV_1, 0)$: `0xdffd 0x3ffd 0x8e7d 0x6e7d`
$CV_2' = H(CV_1', 0)$: `0xdffd 0x3ffd 0x8e7d 0x6e7d`

C.3 Free-start collision for Hirose mode

For Hirose mode, we used as constant c the first 64 output bits of the `SHA-2` computation of the string “IDEA”:

`SHA-2`(”$IDEA$”) = ”`9f8c7b26cde59ca3dacc74ec7afda737ac1d15aa5239206416f79019dbd7ec37`”

c : `0x9f8c 0x7b26 0xcde5 0x9ca3`

$CV1_i$: `0x93e8 0x4d86 0x45a5 0xa829`
$CV1_i'$: `0x13e8 0xcd86 0xc5a5 0x2829`
$CV2_i = CV2_i'$: `0`

$M = M'$: `0`

$CV1_{i+1}$: `0x2101 0x23c9 0xde42 0xdc96`
$CV1'_{i+1}$: `0x2101 0x23c9 0xde42 0xdc96`
$CV2_{i+1}$: `0x0009 0x0401 0x3d38 0x3934`
$CV2'_{i+1}$: `0x0009 0x0401 0x3d38 0x3934`

C.4 Preimage for Davies-Meyer mode

Since a random 64-bit challenge has preimage(s) with a probability p, we show the preimage of a challenge which we are sure at least one preimage exists (similar to a second-preimage search). In order to get the challenge, we use as input the first 64 output bits of the SHA-2 computation of the string "IDEA", and provide one of the preimages found:

SHA-2("$IDEA$") = "9f8c7b26cde59ca3dacc74ec7afda737ac1d15aa5239206416f79019dbd7ec37"

The challenge $H($`0x9f8c7b26cde59ca3`$, 0)$: `0x20ad1fc924e61ba2`

$CV_{i+1} = H(CV_i, M)$: `0x20ad 0x1fc9 0x24e6 0x1ba2`
M : `0`

CV_i : `0x1860 0x002e 0x2d82 0x0200`

CV_i is one preimage out of $2^{23.585}$ for CV_{i+1}, the search takes $2^{25.486}$ IDEA encryptions, and the average cost per preimage is around $2^{1.9}$.

D Double-key block cipher based compression functions

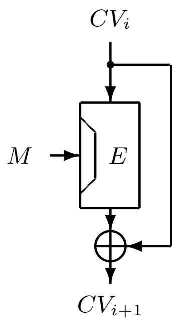


Fig. 2. Davies-Meyer

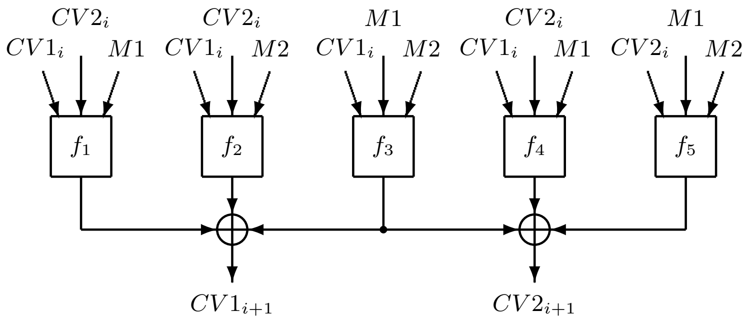


Fig. 3. Peyrin *et al.* (II)

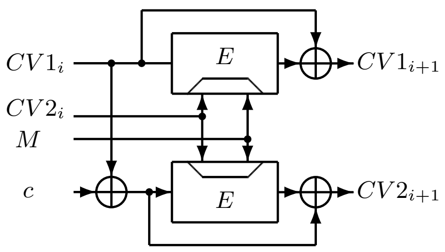


Fig. 4. Hirose

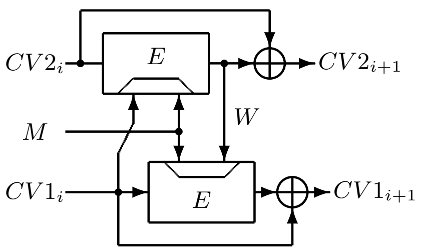


Fig. 5. Tandem-DM

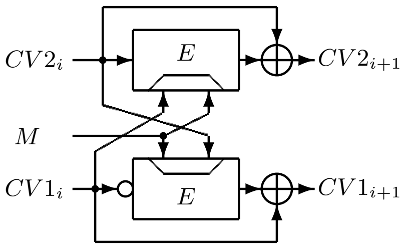


Fig. 6. Abreast-DM

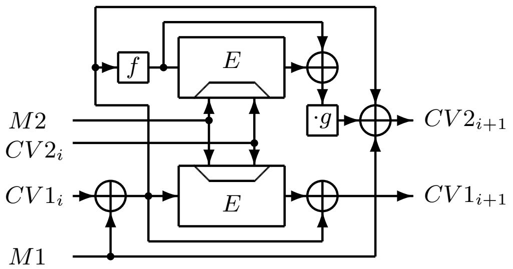


Fig. 7. MJH-Double

The Security of Ciphertext Stealing

Phillip Rogaway[1], Mark Wooding[2], and Haibin Zhang[1]

[1] Dept. of Computer Science, University of California, Davis, USA
[2] Thales e-Security Ltd, UK

Abstract. We prove the security of CBC encryption with ciphertext stealing. Our results cover all versions of ciphertext stealing recently recommended by NIST. The complexity assumption is that the underlying blockcipher is a good PRP, and the security notion achieved is the strongest one commonly considered for chosen-plaintext attacks, indistinguishability from random bits (ind\$-security). We go on to generalize these results to show that, when intermediate outputs are slightly delayed, one achieves ind\$-security in the sense of an online encryption scheme, a notion we formalize that focuses on what is delivered across an online API, generalizing prior notions of blockwise-adaptive attacks. Finally, we pair our positive results with the observation that the version of ciphertext stealing described in Meyer and Matyas's well-known book (1982) is not secure.

Keywords: blockwise-adaptive attacks, CBC, ciphertext stealing, cryptographic standards, modes of operation, provable security.

1 Introduction

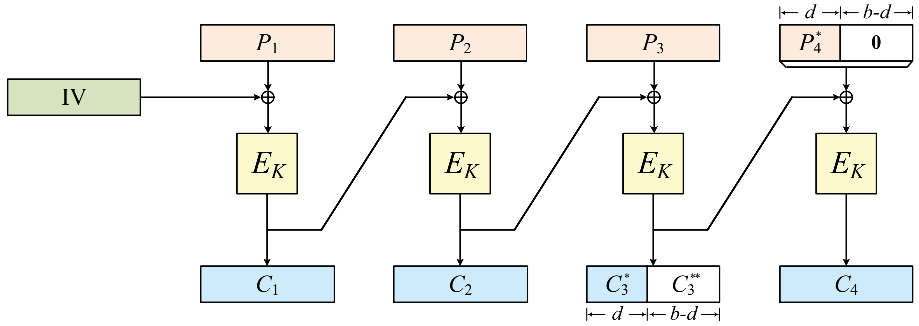
Ciphertext stealing. Many blockcipher modes require the input be a sequence of complete blocks, each having a number of bits that is the blockcipher's blocksize. One approach for dealing with inputs not of this form is *ciphertext stealing.* The classical combination is CBC encryption and ciphertext stealing, a mode going back to at least 1982 [14].

In 2010, NIST put out an addendum [8] to Special Publication 800-38A [7], the document that had defined blockcipher modes ECB, CBC, CFB, OFB, and CTR. The addendum defines three ways to enrich CBC with ciphertext stealing. The modes are named CBC-CS1, CBC-CS2, and CBC-CS3. See Fig. 1 for the definition of these modes, which differ only in the ordering of ciphertext bits.

Despite the classicism of ciphertext-stealing, its adoption in standards, and the strong preferences, these days, for proven-secure modes, there has, until now, been no proof offered for CBC with ciphertext stealing. This paper fills in this gap.

Our contributions. We begin by looking at the NIST ciphertext-stealing modes, which we collectively call CBC-CS. Assuming a random IV, we show that the CBC-CS schemes achieve the strongest conventional form of chosen-plaintext-attack (CPA) security: what we call ind\$, indistinguishability from random bits under an adaptive chosen-plaintext attack. The definition, easily shown to imply all conventional formulations of CPA-style semantic security, formalizes that a ciphertext C is indistinguishable from as many random bits.

Next we show that *delayed* versions of CBC-CS achieve an analogous IND\$ notion that we define for *online* security. The idea of delayed CBC is from Fouque, Martinet, and Poupard [11]. Our formulation for online security generalizes their and subsequent work (further history and credits coming shortly). In particular, prior definitional approaches were specific to blockcipher-based schemes of a specified form—restrictions not in keeping with identifying a general notion of security. We levy no such restrictions, but do imagine that the encryption scheme is written to an incremental API (application programming interface). Each time a user presents a piece of plaintext to encrypt she will get back a corresponding chunk of ciphertext. The length of both is arbitrary. One can understand our definition of online security as establishing that a



```
10    algorithm $\mathrm{CBC\text{-}CS}_K^{IV}(P)$
11    $n \leftarrow \lceil |P|/b \rceil$
12    $P_1 \cdots P_{n-1} P_n^* \leftarrow P$ where $|P_1| = \cdots = |P_{n-1}| = b$
13    $P_n \leftarrow P_n^*\, 0^{b-d}$ where $d \leftarrow |P_n^*|$
14    $C_0 \leftarrow IV$;
15    $C_1 \cdots C_n \leftarrow \mathrm{CBC}_K^{IV}(P_1 \cdots P_n)$ where $|C_1| = \cdots = |C_n| = b$
16    $C_{n-1}^* \leftarrow \mathrm{MSB}_d(C_{n-1})$
17-1  return $C_1 \cdots C_{n-2} C_{n-1}^* C_n$                                        ⇐ for CS1
17-2  if $d{=}b$ return $C_1 \cdots C_{n-2} C_{n-1}^* C_n$ else return $C_1 \cdots C_{n-2} C_n C_{n-1}^*$  ⇐ for CS2
17-3  return $C_1 \cdots C_{n-2} C_n C_{n-1}^*$                                        ⇐ for CS3
```

```
20    algorithm $\mathrm{CBC}_K^{IV}(P_1 \cdots P_n)$ where $|P_1| = \cdots = |P_n| = b$
21    $C_0 \leftarrow IV$
22    for $i \leftarrow 1$ to $n$ do $C_i \leftarrow E_K(C_{i-1} \oplus P_i)$
23    return $C_1 \cdots C_n$
```

Fig. 1. Encryption under NIST modes CBC-CS1, CBC-CS2, and CBC-CS3. The schemes differ only in which version of line 17 is used. The schemes depend on a blockcipher $E\colon \mathcal{K} \times \{0,1\}^b \to \{0,1\}^b$ that determines the key space $\mathcal{K}$, the IV space $\mathcal{IV}$, and the message space $\mathcal{P} = \{0,1\}^{\geq b}$. We insist that $K \in \mathcal{K}$, $IV \in \mathcal{IV}$, and $P \in \mathcal{P}$.

specified incremental API introduces no new security vulnerabilities. Technically, we reconceptualize an encryption scheme *as* the incremental interface. We regard a general definition for online security—a definition motivated by cryptographic APIs and not the characteristics of any particular encryption mode—as an important and independent contribution of this paper.

The workings of delayed CBC—the naturalness of this scheme and how much one must delay—are clarified by freeing the definition of online security from a demand on a scheme being blockcipher-based. Now it is the security analysis, not the syntax, that surfaces by just how much one must delay—an amount that is, in fact, slightly different for the CS1/CS2 and the CS3 versions of the scheme. Absent a careful treatment of such matters the author of an incremental API could well get these things wrong, buffering more than what is necessary or less than what is needed.

Finally, we point out that a 30-year-old version of ciphertext stealing described in the book of Meyer and Matyas [14] is essentially wrong: it will not achieve any desirable security notion we know. The apparently unnoticed observation highlights the importance of having proofs in this domain, and underscores NIST's wisdom in selecting the versions of ciphertext stealing that it did.

Additional history. The provable-security treatment of CBC, and of other blockcipher-based encryption modes, begins with Bellare, Desai, Jokipii, and Rogaway [3]. The stronger ind\$-definition that we adopt here is from Rogaway, Bellare, Black, and Krovetz [16]. For online security, the delayed-CBC scheme that we embellish with NIST's versions of ciphertext stealing is due to Fouque, Martinet, and Poupard [11].

Our definition of online security springs from the line of work on blockwise-adaptive attacks that starts with Bellare, Kohno, and Namprempre [4] and Joux, Martinet, and Valette [13] and continues with Fouque, Martinet, and Poupard [11], Fouque, Joux, and Poupard [10], and Bard [2]. As explained, our own security definitions take a different turn by divorcing the notion of online security from its former association with blockcipher-based schemes. We instead assume an arbitrary symmetric encryption scheme that is presented to the user by way of an incremental API. The user provides the plaintext as a sequence of chunks and the encryption algorithm, buffering what it needs, returns corresponding ciphertext chunks. The approach echos Gennaro and Rohatgi [12], which likewise transplants a primitive (digital signatures) from a setting that sees messages as atomic to one that sees messages as something produced and consumed across an expanse of time.

Discussion. A possible reaction to any discussion of ciphertext stealing is to say: forget it, use CTR mode instead. We are sympathetic to this point of view, knowing no convincing reason to favor CBC encryption over CTR mode, which natively handles plaintexts of arbitrary length. But the fact remains that CBC encryption is widely used, and that ciphertext stealing is a classical, standardized, and elegant way to extend it. This makes it worth attending to.

In justifying the use of ciphertext stealing in a mode that employed it, Matt Ball writes that "[d]espite lacking a formal security proof, ciphertext stealing still has general approval in the cryptographic community" [1, p. 5]. Probably this statement is at some level true, but "general approval" is hard to gauge and far removed from being a proof.

We think that security notions that attend to the vulnerabilities introduced by the specifics of an envisioned API comprise an interesting direction in narrowing the gap between conventional abstractions of cryptographic primitives and what cryptographic practice actually exports. It is not just that protocols may segment conceptually atomic messages (the original motivation for dealing with blockwise adaptivity); rather, it is that the segmentation is actually surfaced to users, and therefore desirable to directly model.

We do not discuss the security of CBC-CS when the IV fails to be unpredictable; it would seem that no interesting or desirable security notion is achieved in this case. NIST SP800-38A appropriately demands an unpredictable IV for CBC [7, Appendix C].

The CBC-CS schemes predate NIST's addendum [8]: CBC-CS2 goes back to at least 1996 [17], while older versions of ciphertext stealing go back to at least 1982 [14]. Looking at these schemes from a modern vantage is long overdue.

2 Preliminaries

Notation. Strings are assumed to be binary, elements of $\{0,1\}^*$. Both $A \,\|\, B$ and $A\,B$ denote the concatenation of strings A and B. If X is a string then $|X|$ is its length. The empty string is denoted ε. Throughout this paper we fix an integer $b \geq 1$ called the *blocksize*. For a string X and a number $d \leq |X|$ let $\text{MSB}_d(X)$ and $\text{LSB}_d(X)$ be the leftmost and rightmost d bits of X.

Blockciphers. A *blockcipher* is a map $E\colon \mathcal{K} \times \{0,1\}^b \to \{0,1\}^b$ where $\mathcal{K} \subseteq \{0,1\}^*$ is finite and $E_K(\cdot) = E(K,\cdot)$ is a permutation for each $K \in \mathcal{K}$. Let $\text{Perm}(b)$ be the set of all permutations on b bits. This may be regarded as a blockcipher with a $(2^b!)$-size key space. Let $\mathbf{Adv}_E^{\text{prp}}(A) = \Pr[\,A^{E_K(\cdot)} \Rightarrow 1\,] - \Pr[\,A^{\pi(\cdot)} \Rightarrow 1\,]$ with $K \overset{\$}{\leftarrow} \mathcal{K}$ and $\pi \overset{\$}{\leftarrow} \text{Perm}(b)$. Similarly define $\mathbf{Adv}_E^{\text{prf}}(A) = \Pr[\,A^{E_K(\cdot)} \Rightarrow 1\,] - \Pr[\,A^{\rho(\cdot)} \Rightarrow 1\,]$ with $K \overset{\$}{\leftarrow} \mathcal{K}$ and $\rho \overset{\$}{\leftarrow} \text{Func}(b)$ for $\text{Func}(b)$ the set of all functions from b bits to b bits. Here $E_K(\cdot)$ need not be a permutation.

Encryption schemes. It has become traditional to regard blockciphers as fixed functions but encryption schemes as tuples, as in $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. To simplify and unify matters we formalize an encryption scheme more like a blockcipher: an (IV-based, symmetric) *encryption scheme* is a function $\mathcal{E}\colon \mathcal{K}\times\mathcal{IV}\times\mathcal{P}\to\mathcal{P}$. We call $\mathcal{K}$, $\mathcal{IV}$, and $\mathcal{P}$ the *key space*, *IV space*, and *message space*. For simplicity we assume that $\mathcal{K}$ is finite and $\mathcal{IV}$ is the set of all strings of some one particular length. We write $\mathcal{E}_K^{IV}(P)$ instead of $\mathcal{E}(K, IV, P)$. To keep things simple we require that $\mathcal{E}_K^{IV}(\cdot)$ be a length-preserving permutation for all $K\in\mathcal{K}$ and $IV\in\mathcal{IV}$. The condition implies that $\mathcal{E}$ has a unique inverse, the map $\mathcal{D}$ where $\mathcal{D}_K^{IV}(C)=P$ when $\mathcal{E}_K^{IV}(P)=C$. Because there is no formal need to specify the decryption direction $\mathcal{D}$ of an encryption scheme $\mathcal{E}$, we never do so. Of course it is important in practice that $\mathcal{E}$ and $\mathcal{D}$ have efficient realizations, it is simply that this doesn't show up in the statement of definitions or security results.

Let $\mathcal{E}\colon \mathcal{K}\times\mathcal{IV}\times\mathcal{P}\to\mathcal{P}$ be an IV-based encryption scheme and let A be an adversary (algorithm) with one of two types of oracles. A *real* encryption oracle Real$(\cdot)$ chooses a random $K\xleftarrow{\$}\mathcal{K}$ and then, on input $P\in\mathcal{P}$, returns $C\leftarrow IV \parallel \mathcal{E}_K^{IV}(P)$ for a random $IV\xleftarrow{\$}\mathcal{IV}$. A *fake* encryption oracle Fake$(\cdot)$ takes an input $P\in\mathcal{P}$ and returns $C\xleftarrow{\$}\{0,1\}^c$ where $c=|IV|+|P|$ (for $IV\in\mathcal{IV}$). Define $\mathbf{Adv}_{\mathcal{E}}^{\mathrm{ind\$}}(A)=\Pr[\,A^{\mathrm{Real}(\cdot)}\Rightarrow 1\,]-\Pr[\,A^{\mathrm{Fake}(\cdot)}\Rightarrow 1\,]$. This "indistinguishability-from-random-bits" definition is easily shown to imply all conventional (CPA) formulations of indistinguishability and semantic security [3]; that we have selected a different syntax makes no difference in the proofs.

Note that even though the encryption function is formalized as taking, besides the key, an IV and a plaintext, the security definition does not allow the adversary to specify the IV; the adversary asks P and the IV is randomly generated, used, and and returned. Our security notion thus formalizes security for random IVs, not, for example, security for nonce IVs.

3 Conventional Security of the CBC-CS Schemes

We begin with a simple proposition about the security of conventional CBC encryption (no ciphertext stealing) with a random IV. The result is needed insofar as we deduce the security of CBC-CS from it. Recall that the mode was defined in Fig. 1 and was proven secure by Bellare *et al.* [3]. That proof, however, is for a somewhat weaker definition than the one we use here. The proof below is a simple application of the game-playing technique [5, 18].

Lemma 1. *Suppose A asks queries totaling at most σ blocks. Then we have $\mathbf{Adv}_{\mathrm{CBC[Perm}(b)]}^{\mathrm{ind\$}}(A)\le \sigma^2/2^b$.*

Proof. The difference between $\mathbf{Adv}_{\mathrm{CBC[Perm}(b)]}^{\mathrm{ind\$}}(A)$ and $r=\mathbf{Adv}_{\mathrm{CBC[Func}(b)]}^{\mathrm{ind\$}}(A)$ is at most $0.5\,\sigma^2/2^b$; this is a standard application of PRP/PRF switching [5]. It thus suffices to bound r by $r\le 0.5\,\sigma^2/2^b$. To that end, consider the games of Fig. 2. Observe that, with $\mathcal{E}=\mathrm{CBC[Func}(b)]$, $\Pr[A^{\mathrm{Real}(\cdot)}\Rightarrow 1]=\Pr[A^{G_1(\cdot)}\Rightarrow 1]$, while $\Pr[A^{\mathrm{Fake}(\cdot)}\Rightarrow 1]=\Pr[A^{G_0(\cdot)}\Rightarrow 1]$. As a consequence, we have that $r=\Pr[A^{\mathcal{G}_1(\cdot)}\Rightarrow 1]-\Pr[A^{\mathcal{G}_0(\cdot)}\Rightarrow 1]$ and, the two games being identical-until-*bad*, we know that $r\le\Pr[A^{G_0}$ sets *bad*$]$. Because in game G_0 all of the C_i values are uniform and independent of P_i, so too all of the X_i values are uniform and independent of one another, so the probability that *bad* gets set—the probability some two of the X_i's collide—is at most $(1+2+\cdots(\sigma-1))/2^b\le 0.5\,\sigma^2/2^b$. This completes the proof.

Turning now to the CBC-CS modes, we claim that these inherit CBC's security with no quantitative degradation. The needed observation is that $\text{CBC-CS1}_K^{IV}(P)$ is just $\mathrm{CBC}_K^{IV}(P0^*)$ (minimal padding to the next multiple of b bits) with some bits excised and some bits reordered. Which bits are excised and how bits are rearranged depends only on $|P|$. Thus if $\mathrm{CBC}_K^{IV}(\cdot)$ looks random, so too will look $\text{CBC-CS1}_K^{IV}(\cdot)$. The same comments hold for CBC-CS2 and CBC-CS3; these are just different rearrangements of the bits of $\mathrm{CBC}_K^{IV}(P\,0^*)$. The observation and proof are formalized by the proposition below.

```
algorithm Enc($P$)                                              Game $G_0$
$P_1 \cdots P_n \leftarrow P$ where $|P_1| = \cdots = |P_n| = b$                  $\boxed{\text{Game } G_1}$
$C_0, \ldots, C_n \xleftarrow{\$} \{0,1\}^b$
for $i \leftarrow 1$ to $n$ do
    $X_i \leftarrow P_i \oplus C_{i-1}$
    if $\rho(X_i)$ then $bad \leftarrow$ true, $\boxed{C_i \leftarrow \rho(X_i)}$
    $\rho(X_i) \leftarrow C_i$
return $C_0\, C_1 \cdots C_n$
```

Fig. 2. Proof of the ind\$-security of CBC encryption with a random IV. This application of game-playing is probably simple and well-known enough to be considered folklore. Game G_1 includes the boxed statement following the setting of *bad*; game G_0 omits it. Variable *bad* is initialized to false and ρ is initialized to everywhere undefined, a value treated as false if used as a boolean.

Theorem 1. *Let* $\mathcal{E}$ *be any of* CBC-CS1[Perm(b)], CBC-CS2[Perm(b)], *or* CBC-CS3[Perm(b)] *and suppose adversary* A *asks queries totaling at most* σ *blocks. Then* $\mathbf{Adv}_{\mathcal{E}}^{\text{ind\$}}(A) \le \sigma^2/2^b$.

Proof. Suppose that A, asking σ total blocks of queries, gets advantage δ at distinguishing oracles $\mathcal{E} = \text{CBC-CS1}(\cdot)$ and $\$(\cdot)$. The first of these oracles chooses a random permutation $\pi \xleftarrow{\$} \text{Perm}(n)$ and then, when asked a query $P \in \{0,1\}^{\ge b}$, returns $IV \parallel \text{CBC-CS1}_{\pi}^{IV}(P)$ for a random $IV \xleftarrow{\$} \{0,1\}^b$; the second oracle, when asked a query P, returns a random string of length $b+|P|$. We construct from A an adversary B that, also asking σ blocks worth of queries, also gets advantage δ, but now at distinguishing between CBC($\cdot$) and $\$(\cdot)$. The first of these oracle chooses a random permutation $\pi \xleftarrow{\$} \text{Perm}(n)$ and then, when asked a query $P \in (\{0,1\}^b)^+$, returns $IV \parallel \text{CBC}_{\pi}^{IV}(P)$ for a random $IV \xleftarrow{\$} \{0,1\}^b$. Adversary B now works as follows: it runs A and when A generates a query of $P \in \{0,1\}^{\ge b}$ adversary B queries its own oracle on $P' = P\,0^*$, meaning P padded on the right with the minimal number of zero-bits so that P' is a multiple of b bits. Suppose this returns a ciphertext $C = C_0 C_1 \cdots C_n$ where $|C_i| = n$. Then B returns to A the string $C^* = C_0 C_1 \cdots C_{n-1}^* C_n$ where $C_{n-1}^* = \text{MSB}_d(C_{n-1})$ and $d = b - (|P| \bmod b)$. We observe that $\Pr[B^{\text{CBC-CS1}(\cdot)} \Rightarrow 1] = \Pr[A^{\text{CBC}(\cdot)} \Rightarrow 1]$ (we have reordered bits exactly as required by CBC-CS1) and that $\Pr[B^{\$(\cdot)} \Rightarrow 1] = \Pr[A^{\$(\cdot)} \Rightarrow 1]$ (reordered and pruned uniform random bits are still uniform), and so $\delta = \mathbf{Adv}_{\text{CBC-CS1[Perm}(b)]}^{\text{ind\$}}(A) = \mathbf{Adv}_{\text{CBC[Perm}(b)]}^{\text{ind\$}}(B)$. By Proposition 1 we thus have $\delta \le 0.5\,\sigma^2/2^b$. This establishes the first of the three results. The analogous results for CBC-CS2 and CBC-CS3 are obtained simply by modifying the string C^* returned to A: for CBC-CS2 return $C^* = C_0 C_1 \cdots C_{n-2} C_{n-1}^* C_n$ when $|P|$ is a multiple of b and $C^* = C_0 C_1 \cdots C_{n-2} C_n C_{n-1}^*$ otherwise; for CBC-CS3 always return $C^* = C_0 C_1 \cdots C_{n-2} C_n C_{n-1}^*$. This completes the theorem.

The proof's simplicity stems from having unidentified a clean abstraction boundary: directly modifying the proof of Lemma 1 to attend to the ciphertext stealing would be much more complex.

Finally, one can pass from the information-theoretic result to its complexity-theoretic analog in the standard way, trading the family of random permutations for a conventional blockcipher. Stating the result for completeness, we have the following.

Corollary 1. *Let* $E\colon \mathcal{K} \times \{0,1\}^b \to \{0,1\}^b$ *be a blockcipher and let* $\mathcal{E}$ *be any of the encryption schemes* CBC-CS1[E], CBC-CS2[E], *or* CBC-CS3[E]. *Suppose* A *asks queries that total* σ *blocks, runs in time* t, *and achieves advantage* $\delta = \mathbf{Adv}_{\mathcal{E}}^{\text{ind\$}}(A)$. *Then there is an adversary* B, *explicitly known and constructed from* A *in a blackbox manner, that asks at most* σ *queries, runs in time at most* $t + \lambda b \sigma$, *and achieves advantage* $\mathbf{Adv}_{E}^{\text{prp}}(B) \ge \delta - \sigma^2/2^b$. *Here* λ *is an absolute constant depending only on details of the model of computation.*

4 Defining Online Security

Syntax. We adjust the syntax of an encryption scheme to accommodate the staged presentation of plaintexts and ciphertexts. Rather than messages being atomic objects that get encrypted all at once, messages may be arbitrarily partitioned into chunks, each of which gets fed into a stateful encryption engine. Breaking with former treatments, we do not assume that chunks are single blocks, nor multiples of blocks, where the length of a block is the blocksize of some underlying blockcipher. Instead, we provide a general definition where one assumes nothing about the structure of the underlying encryption scheme (in particular, there is no assumption that it is blockcipher-based). As each installment of plaintext is provided to the encryption interface, it is up to the algorithm to decide how much ciphertext to spit out. The algorithm will thus return not only a ciphertext chunk, but also an updated state.

Realizing the idea above, we choose to define an *online encryption scheme* as a function $\mathcal{E}\colon \mathcal{K} \times \mathcal{V} \times \{0,1\} \times \{0,1\}^* \to \{0,1\}^* \times \mathcal{V}$. We write $\mathcal{E}_K^{V,\delta}(P)$ for $\mathcal{E}(K,V,\delta,P)$. We call $\mathcal{K}$ and $\mathcal{V}$ the *key space* and *state space*, respectively. The key space is finite and the state space is a finite set of strings. The third argument to $\mathcal{E}$, a bit, is the *end-of-message indicator*. The final argument to $\mathcal{E}$ is the next chunk of *message*. An online encryption scheme $\mathcal{E}$ must have an associated *IV space* $\mathcal{IV} \subseteq \mathcal{V}$ and *message space* $\mathcal{P} \subseteq \{0,1\}^*$. The former contains strings of some one fixed length. Formally, an online encryption scheme is the tuple $(\mathcal{E}, \mathcal{IV}, \mathcal{P})$, but we will usually use the first component as shorthand for the whole.

We also impose a number of "syntactic" requirements on an online encryption scheme $(\mathcal{E}, \mathcal{IV}, \mathcal{P})$. First we define some additional notation. We write $(C_1, \ldots, C_n) \leftarrow \mathcal{E}_K^{IV}(P_1, \ldots, P_n)$ for the sequence:

```
    $V_0 \leftarrow IV$
    for $i \leftarrow 1$ to $n-1$ do $(C_i, V_i) \leftarrow \mathcal{E}_K^{V_{i-1},0}(P_i)$
    $(C_n, V_n) \leftarrow \mathcal{E}_K^{V_{n-1},1}(P_n)$
    return $(C_1, \ldots, C_n)$.
```

Alternatively, we can think of $\mathcal{E}_K^{IV}(P_1, \ldots, P_n)$ as returning a single string, setting $\mathcal{E}_K^{IV}(P_1, \ldots, P_n)$ to $C = C_1 \cdots C_n$ where $(C_1, \ldots, C_n) \leftarrow \mathcal{E}_K^{IV}(P_1, \ldots, P_n)$.

Now fix an online encryption scheme $(\mathcal{E}, \mathcal{IV}, \mathcal{P})$.

- The *consistency requirement* says that you get the same ciphertext regardless of how you split up the plaintext. More formally, if $P_1 \parallel \cdots \parallel P_n = P'_1 \parallel \cdots \parallel P'_{n'} = P \in \mathcal{P}$ then $\mathcal{E}_K^{IV}(P_1, \ldots, P_n) = \mathcal{E}_K^{IV}(P'_1, \ldots, P'_{n'})$. We can therefore write this as $\mathcal{E}_K^{IV}(P)$ without ambiguity.
- The *invertibility requirement* is that $\mathcal{E}_K^{IV}(\cdot)$ is injective on $\mathcal{P}$ (for all $K \in \mathcal{K}$ and $IV \in \mathcal{IV}$).
- The *length requirement* is that the length of the first and second components of $\mathcal{E}_K^{V,\delta}(P)$ depend only on $|V|$, $|P|$, and δ. This ensures that, when $(C_1, \cdots, C_m) \leftarrow \mathcal{E}_K^{IV}(P_1, \cdots, P_m)$, the lengths of $C_1, C_2, \ldots, C_m$ reveal nothing about $P = P_1 \cdots P_m$ beyond how it was partitioned up.

Indistinguishability. We define a very strong form of indistinguishability for an online encryption scheme: indistinguishability from random bits. Fix an online encryption scheme $(\mathcal{E}, \mathcal{IV}, \mathcal{P})$ and consider the following two $\mathcal{E}$-dependent oracles.

- Real(i, M, δ): At the beginning, set $K \xleftarrow{\$} \mathcal{K}$ and $V_i \xleftarrow{\$} \mathcal{IV}$ for all $i \in \mathbb{N}$. Then, on query $(i, P, \delta) \in \mathbb{N} \times \{0,1\}^* \times \{0,1\}$, compute $(C, V_i) \xleftarrow{\$} \mathcal{E}_K^{V_i,\delta}(P)$ and return C.
- Fake(i, P, δ): At the beginning, set $K \xleftarrow{\$} \mathcal{K}$ and $V_i \xleftarrow{\$} \mathcal{IV}$ for all $i \in \mathbb{N}$. Then, on query $(i, P, \delta) \in \mathbb{N} \times \{0,1\}^* \times \{0,1\}$, compute $(C, V_i) \xleftarrow{\$} \mathcal{E}_K^{V_i,\delta}(P)$ and return $|C|$ random bits.

We define $\mathbf{Adv}_{\mathcal{E}}^{\mathrm{IND\$}}(A) = \Pr[A^{\mathrm{Real}} \Rightarrow 1] - \Pr[A^{\mathrm{Fake}} \Rightarrow 1]$. Informally, an online encryption scheme is IND\$-secure if an adversary can't distinguish the ciphertexts it is receiving from random bits.

Discussion. Some of our high-level definitional choices differ for conventional and online encryption schemes. A conventional encryption scheme does not spit out its IV, while an online scheme does. The former is needed to match NIST's definitions for the CBC-CS schemes, but it works less well in the online setting, as here it is important that the algorithm can decide if and when to release the IV. Typically, the IV does get discharged, and as the first part of the ciphertext, so we say that an online encryption scheme $(\mathcal{E}, \mathcal{IV}, \mathcal{P})$ is *IV-prefixed* if $C = \mathcal{E}_K^{IV}(P)$ is always IV followed by some $|P|$ additional bits (assuming $K \in \mathcal{K}$ and $IV \in \mathcal{IV}$). An IV-prefixed online encryption scheme $(\mathcal{E}, \mathcal{IV}, \mathcal{P})$ determines a conventional encryption scheme $\hat{\mathcal{E}}$ in the natural way, setting $\hat{\mathcal{E}}_K^{IV}(P)$ to be $\mathcal{E}_K^{IV}(P)$ stripped of its initial $|IV|$ bits. Conversely, a conventional encryption scheme $\hat{\mathcal{E}}\colon \mathcal{K} \times \mathcal{IV} \times \mathcal{P} \to \mathcal{P}$ is realized by an IV-prefixed online encryptions scheme $(\mathcal{E}, \mathcal{IV}, \mathcal{P})$ if the latter determines the former in the manner just defined. In this way one can speak of an encryption scheme $\mathcal{E}\colon \mathcal{K} \times \mathcal{IV} \times \mathcal{P} \to \mathcal{P}$ as being online; the statement means that it has a secure online realization (the notion of security soon to be defined).

While our notions make sense regardless of whether or not $\mathcal{V}$ is finite, its being finite is the essence of what it means to be online: that one can encrypt (and decrypt) streaming messages without having to buffer more than a constant number of bits. Equivalently, that one can implement an incremental API with a fixed-size context. Our notions allow one to consider things in a more quantitative manner, using $|\mathcal{V}|$ as a measure of worth. We say that $\mathcal{E}\colon \mathcal{K} \times \mathcal{V} \times \{0,1\} \times \{0,1\}^* \to \{0,1\}^* \times \mathcal{V}$ *uses v-bits of state* if v is the smallest number such that $\mathcal{V} \subseteq \{0,1\}^{\leq v}$.

Since we concern ourselves only with chosen-plaintext security, we do not formalize syntax or security for the decryption direction of an online encryption scheme. Still, we comment that if an incremental encryption scheme is online then it has an online (that is, finite state-space) decryption.

We regard the initialization vector IV as the initial value of the saved state V. The embedding of the IV space into the state space doesn't prevent a scheme from performing "special" initialization; one can always distinguish the first chunk of a message from subsequent chunks of message by arranging that point in $\mathcal{IV}$ are never returned as a modified state.

An online encryption function has control over if and when the IV is revealed. This can be essential for security: in particular, the Delayed CBC scheme we will soon describe is insecure if the IV is revealed too soon.

Note that the IND\$-definition allows interleaved querying of multiple streams; this is the purpose of the index i. Fouque, Martinet, and Poupard earlier observed that, with respect to their definitions for online indistinguishability, this made for a stronger security notion [11]. The same is true for us; it is easy to see that if the adversary were restricted to asking a sequence of messages with nondecreasing indexes, a restriction that amounts to forbidding the interleaving of encryptions, the resulting security notion would be properly weaker.

We do not find it necessary to demand that, once an oracle query $(i, \cdot, 1)$ is made, there are no subsequent queries $(i, \cdot, \cdot)$. Nonetheless, this is the expected behavior, as the setting of $\delta = 1$ is meant to indicate that the message is complete.

5 Online Security of the CBC-CS Schemes

Delayed CBC. We now present an online version of CBC mode. For the moment, assume all messages have a multiple of b bits. The most obvious approach for defining an online version of CBC is to just spit out ciphertext blocks as they are formed. But this does not work: if an adversary knows C_{i-1} it can choose P_i such that $C_{i-1} \oplus P_i = P_j \oplus C_{j-1}$ for some $j < i$, whence C_i will be C_j if the adversary has a "real" encryption oracle, while this is unlikely if the adversary has a "fake" encryption oracle. We can defend against this attack and, more broadly,

```
algorithm $\mathrm{DCBC}_K^{V,\delta}(P)$
if $|V| < b$ then return error
$C_0\, P_0 \leftarrow V$ where $|C_0| = b$
$P \leftarrow P_0\, P$;   $n \leftarrow \lfloor\, |P|/b \,\rfloor$
$P_1 \cdots P_n\, P^* \leftarrow P$ where $|P_1| = \cdots = |P_n| = b$
if $\delta = 1$ and $P^* \neq \varepsilon$ then return error
for $i \leftarrow 1$ to $n$ do $C_i \leftarrow E_K(P_i \oplus C_{i-1})$
if $\delta = 0$ then $(C, V') \leftarrow (C_0 \cdots C_{n-1},\ C_n P^*)$
if $\delta = 1$ then $(C, V') \leftarrow (C_0 \cdots C_n,\ \varepsilon)$
return $(C, V')$
```

Fig. 3. Mode DCBC. An online encryption scheme, encryption now depends on the saved state $V \in \{0,1\}^*$. The first b bits of V comprise the *pending ciphertext*, C_0, while the remaining 0 to $b-1$ bits are *unprocessed plaintext*, P_0. Bit δ signals if the plaintext is over.

```
algorithm $\mathrm{Enc}(j, P, \delta)$                                        Game $G_0$
if $|V_j| < b$ then return error                                        $\boxed{\text{Game } G_1}$
$C_0\, P_0 \leftarrow V_j$ where $|C_0| = b$
$P \leftarrow P_0\, P$;   $n \leftarrow \lfloor\, |P|/b \,\rfloor$
$P_1 \cdots P_n\, P^* \leftarrow P$ where $|P_1| = \cdots = |P_n| = b$
if $\delta = 1$ and $P^* \neq \varepsilon$ then return error
for $i \leftarrow 1$ to $n$ do
    $X_i \leftarrow P_i \oplus C_{i-1}$
    $C_i \xleftarrow{\$} \{0,1\}^b$
    if $\rho(X_i) \neq$ undefined then $bad \leftarrow$ true, $\boxed{C_i \leftarrow \rho(X_i)}$
    else $\rho(X_i) \leftarrow C_i$
if $\delta = 0$ then $(C, V_j') \leftarrow (C_0 \cdots C_{n-1},\ C_n P^*)$
if $\delta = 1$ then $(C, V_j') \leftarrow (C_0 \cdots C_n,\ \varepsilon)$
return $C$
```

Fig. 4. Proof of the IND$-security of Delayed CBC. Game G_1 includes the boxed statement following the setting of *bad*; game G_0 omits it. The variable *bad* is initialized to false, V_j is initialized to a random b-bit string chosen uniformly at random for each $j \in \mathbb{N}$, and ρ is initialized to everywhere undefined.

get online-secure scheme, simply by *delaying* the last ciphertext block from each plaintext chunk, holding onto it until the relevant blockcipher has already been made. The idea is due to Fouque, Martinet, and Poupard [11]. The contents of this section are a strengthening and extension of that work, adding ciphertext stealing, employing less restrictive syntax, and establishing a stronger notion of security.

The algorithm, detailed in Fig. 3, is called *delayed CBC*, or DCBC. The state consists of two parts: a *pending ciphertext block*, which initially contains a randomly generated IV, and *unprocessed plaintext*, a partial block, possibly empty, carried over from the previous message chunk. If the blockcipher acts on b bits then the state will be at most $v = 2b - 1$ bits. In the pseudocode of Fig. 3, regard $C_i \cdots C_j$ as the empty string if $i > j$.

Informally, the algorithm of Fig. 3 proceeds as follows. The algorithm receives a key K, a state V, an end-of-message indicator δ, and a plaintext chunk P. It parses the state into a b-bit delayed ciphertext block C_0, and a partial plaintext block P_0 with $0 \leq |P_0| < b$. The algorithm then adjusts the incoming plaintext chunk P by prefixing it with P_0. Next it splits P into b-bit blocks $P_1, \ldots, P_n$, leaving a leftover and possibly empty partial block P^*. Since DCBC can only cope with messages that are an integral number of blocks long, the algorithm fails (it reports an error) if $P^* \neq \varepsilon$ when $\delta = 1$. The algorithm next performs the CBC encryption: for $1 \leq i \leq n$, set $C_i = E_K(P_i \oplus C_{i-1})$. Finally, if $\delta = 1$ the algorithm outputs all of $C = C_0 \cdots C_n$, and clears the state. If $\delta = 0$ then it outputs $C = C_0 \cdots C_{n-1}$, holding $V' = C_n \parallel P^*$ in the revised state.

Online security of DCBC. We now show that Delayed CBC achieves IND$ security.

Theorem 2. *Suppose that adversary A asks queries totaling at most σ blocks (each query P contributes $\lceil |P|/b \rceil$ blocks). Then $\mathbf{Adv}^{\mathrm{IND\$}}_{\mathrm{DCBC[Perm}(b)]}(A) \le \sigma^2/2^b$.*

Proof. Let $r = \mathbf{Adv}^{\mathrm{IND\$}}_{\mathrm{DCBC[Func}(b)]}(A)$. As in the proof of Lemma 1, the PRF/PRP switching gives us that $\left|\mathbf{Adv}^{\mathrm{IND\$}}_{\mathrm{DCBC[Perm}(b)]}(A) - r\right| \le \sigma^2/2^{b+1}$. It remains to show that $r \le \sigma^2/2^{b+1}$, for which we use the games in Fig. 4.

The games are constructed so that, with $\mathcal{E} = \mathrm{DCBC[Func}(b)]$, we have $\Pr[A^{\mathrm{Real}(\cdot,\cdot,\cdot)} \Rightarrow 1] = \Pr[A^{G_1(\cdot,\cdot,\cdot)} \Rightarrow 1]$ and $\Pr[A^{\mathrm{Fake}(\cdot,\cdot,\cdot)} \Rightarrow 1] = \Pr[A^{G_0(\cdot,\cdot,\cdot)} \Rightarrow 1]$. Furthermore, games G_0 and G_1 are identical-until-*bad*, and hence we get $r = \left|\mathbf{Adv}[A^{G_1(\cdot,\cdot,\cdot)} \Rightarrow 1] - \mathbf{Adv}[A^{G_0(\cdot,\cdot,\cdot)} \Rightarrow 1]\right| = \Pr[A^{G_0}$ sets *bad*$]$.

In game G_0, all of the C_i values are uniform and independent: all except C_0 in the initial call are generated explicitly by the oracle—and that C_0 is the initial state V_j, chosen uniformly as part of the initialization.

Since the P_i are determined solely by the adversary's inputs, we can think of them as being selected directly by the adversary. We claim that the adversary must choose each P_i before receiving any information about C_{i-1}. For $i > 1$ this is clear, since C_{i-1} is chosen uniformly at random after P_i has been determined. It remains to show that C_0 is uniformly distributed and independent of the adversary's view until P_1 is determined. (Ensuring this property is the reason for delaying the ciphertext block.) We do this inductively, and separately for each index $j \in \mathbb{N}$. The base case is the first encryption query with index j: then $C_0 = V_j$ is the randomly selected initialization vector. Here the adversary can't know anything about its value at this stage since it hasn't been used in any computations at all. The state is empty and we return an immediate error if the previous call's end-of-message indicator was set, so there is no C_0 to concern ourselves with. In the remaining case, the value of C_0 is equal to the value of C_n from the previous encryption query with the same index; the inductive step, therefore, is to show that C_n is uniform and independent of the adversary's view if C_0 is also and $\delta = 0$. But nothing dependent on C_n is part of the oracle's output if $\delta = 0$, and C_n is either freshly generated (if $n > 0$), or equal to C_0 and therefore uniform and independent of the adversary by the induction hypothesis (if $n = 0$).

It immediately follows that each P_i is independent of C_{i-1}, and therefore all of the X_i values are uniform and independent of one another. Hence the probability that two X_i collide—and *bad* is set—is at most $\sigma^2/2^{b+1}$, completing the proof.

As usual, it is easy to pass from the information-theoretic setting to complexity-theoretic one.

Delayed CBC with ciphertext stealing. The algorithms DCBC-CS1, DCBC-CS2, and DCBC-CS3 are defined in Fig. 5. Implicitly, the modes are all parameterized by a blockcipher $E\colon \mathcal{K} \times \{0,1\}^b \to \{0,1\}^b$. The state V once again maintains two portions: the *pending ciphertext* and the *unprocessed plaintext*. The pending ciphertext is a single block—or possibly two blocks in the cases of DCBC-CS3—that the algorithm retains until it is "safe" to spit this out. This is followed by 0 to $b-1$ bits of unprocessed plaintext. The dividing line between the two portions is always clear from the length of the string V. Note that for DCBC-CS3, the state has grown from $2b-1$ bits to $2b$ bits while, for DCBC-CS1 and DCBC-CS2 the state remains at $2b-1$ bits.

The IND\$ security of the DCBC-CS schemes can be inferred from the IND\$ security of the DCBC schemes. This is done in the proof below.

Theorem 3. *Let $\mathcal{E}$ be any of* DCBC-CS1[Perm(b)], DCBC-CS2[Perm(b)], *or* DCBC-CS3[Perm(b)], *and suppose A asks queries totaling at most σ blocks. Then $\mathbf{Adv}^{\mathrm{IND\$}}_{\mathcal{E}}(A) \le \sigma^2/2^b$.*

```
40    algorithm $\mathrm{DCBC\text{-}CS}_K^{V,\,\delta}(P)$
41    if $|V| < b$ then return error
42    $C_{-1}\, C_0\;\; P_0 \leftarrow V$ where $|C_{-1}| \in \{0, b\},\; |C_0| = b,\; |P_0| < b$
43    $P \leftarrow P_0\, P$
44    if $\delta = 0$ then $P_1 \cdots P_n\; P^* \leftarrow P$ where $n \leftarrow \lfloor |P|/b \rfloor,\, |P_1| = \cdots = |P_n| = b$
45    else $P_1 \cdots P_n \leftarrow P\, 0^{b-d}$ where $n \leftarrow \lceil |P|/b \rceil,\, d \leftarrow b + |P| - nb,\, |P_1| = \cdots = |P_n| = b$
46    for $i \leftarrow 1$ to $n$ do $C_i \leftarrow E_K(P_i \oplus C_{i-1})$
47    if $\delta = 0$ then
48-1      $(C, V') \leftarrow (C_0 \cdots C_{n-1},\;\; C_n P^*)$                                   $\Longleftarrow$ for CS1
48-2      $(C, V') \leftarrow (C_0 \cdots C_{n-1},\;\; C_n P^*)$                                   $\Longleftarrow$ for CS2
48-3      $(C, V') \leftarrow (P^* = \varepsilon)$? $(C_{-1} C_0 \cdots C_{n-2},\; C_{n-1} C_n)$ :        $\Longleftarrow$ for CS3
                                      $(C_{-1} C_0 \cdots C_{n-1},\; C_n\, P^*)$
49    if $\delta = 1$ then
50        if $n > 0$ then $C_{n-1} \leftarrow \mathrm{MSB}_d(C_{n-1})$
51-1      $(C, V') \leftarrow (C_0 \cdots C_{n-2} C_{n-1} C_n,\; \varepsilon)$                       $\Longleftarrow$ for CS1
51-2      $(C, V') \leftarrow (d = b)$? $(C_0 \cdots C_{n-2} C_{n-1} C_n,\; \varepsilon)$ :            $\Longleftarrow$ for CS2
                                      $(C_0 \cdots C_{n-2} C_n C_{n-1},\; \varepsilon)$
51-3      $(C, V') \leftarrow (C_{-1} C_0 \cdots C_{n-2} C_n C_{n-1},\; \varepsilon)$                 $\Longleftarrow$ for CS3
52    return $(C, V')$
```

Fig. 5. Delayed CBC with ciphertext stealing: DCBC-CS. Each online scheme depends on $E\colon \mathcal{K} \times \{0,1\}^b \to \{0,1\}^b$. String $C_{-1}C_0$ is pending ciphertext (with C_{-1} used only for DCBC-CS3). String P_0 is unprocessed plaintext from the prior call.

```
400   algorithm $\mathcal{DCBC\text{-}CS}_K^{V,\,\delta}(P)$
401   if $|V| \le b$ then $(W, C_{-1}, \ell) \leftarrow (V, \varepsilon, 0)$ else $[W, C_{-1}, \ell] \leftarrow V$
402   $m \leftarrow \ell + |P|,\;\; d \leftarrow m - b\lfloor (m-1)/b \rfloor$
403   if $\delta = 1$ then $P \leftarrow P\, 0^{b-d}$
404   $(C, W') \leftarrow \mathrm{DCBC}_K^{W,\,\delta}(P)$
405   $C_0 \cdots C_n \leftarrow C$ where $n \leftarrow |C|/b - 1$ and $|C_0| = \cdots = |C_n| = b$
406   if $\delta = 0$ then
407-1     $(C', C'_{-1}) \leftarrow (C_0 \cdots C_n, \varepsilon)$                                   $\Longleftarrow$ for CS1
407-2     $(C', C'_{-1}) \leftarrow (C_0 \cdots C_n, \varepsilon)$                                   $\Longleftarrow$ for CS2
407-3     $(C', C'_{-1}) \leftarrow (d = b)$? $(C_0 \cdots C_{n-1}, C_n)$ : $(C_0 \cdots C_n, \varepsilon)$   $\Longleftarrow$ for CS3
408       $\ell' \leftarrow (d = b)$? $0$ : $d$
409   if $\delta = 1$ then
410       if $n > 0$ then $C_{n-1} \leftarrow \mathrm{MSB}_d(C_{n-1})$
411-1     $C' \leftarrow C_0 \cdots C_{n-2} C_{n-1} C_n$                                            $\Longleftarrow$ for CS1
411-2     $C' \leftarrow (d = b)$? $C_0 \cdots C_{n-2} C_{n-1} C_n$ : $C_0 \cdots C_{n-2} C_n C_{n-1}$     $\Longleftarrow$ for CS2
411-3     $C' \leftarrow C_0 \cdots C_{n-2} C_n C_{n-1}$                                            $\Longleftarrow$ for CS3
412       $\ell' \leftarrow 0,\;\; C'_{-1} \leftarrow \varepsilon$
413   return $(C, [W', C'_{-1}, \ell'])$
```

Fig. 6. Defining DCBC-CS in terms of DCBC. The notation $[x_1, \ldots, x_n]$ denotes an unambiguous non-compressing encoding of the items $x_1, \ldots, x_n$; used on the left-hand side of an assignment, it implies a decoding operation.

Proof. We use a different description of DCBC-CS, shown in Fig. 6, now writing the algorithm in terms of DCBC. The state vector consists of three components: a state W for DCBC, which is not interpreted; an additional delayed ciphertext block C_{-1}, which corresponds to C_{-1} in Fig. 5; and a length $0 \le \ell < b$, which keeps track of the amount of unprocessed plaintext maintained in W, so that $\ell = |P_0|$.

The theorem will follow from three observations about this new description of DCBC. First, $\mathcal{DCBC\text{-}CS}$ is functionally identical to DCBC-CS. Second, if the call to function DCBC at line 404 were to instead call a function that returned a random strings of the appropriate length, then so too would $\mathcal{DCBC\text{-}CS}$. This observation is immediate, since the strings returned DCBC-CS$'$ are derived from those returned by $\mathrm{DCBC}_K^{V,\,\delta}$ by discarding and reordering particular fixed bits.

Third, $\mathcal{DCBC\text{-}CS}$ can be implemented using only oracle access to the DCBC function: it doesn't need to inspect or interpret the DCBC state vector W, nor examine the key K, and it uses the state only in the "single-threaded" way permitted by the online IND\$ oracle.

Consequently, for any adversary A attacking DCBC-CS, we can construct an adversary B attacking DCBC: B will run A against a simulated oracle built from B's (real or fake) DCBC oracle using $\mathcal{DCBC\text{-}CS}$ and, in the end, output A's guess as its own. We have

$$\begin{aligned}\mathbf{Adv}^{\mathrm{IND\$}}_{\mathrm{DCBC\text{-}CS}}(A) &= \Pr[A^{\mathrm{Real}} \Rightarrow 1] - \Pr[A^{\mathrm{Fake}} \Rightarrow 1] \\ &= \Pr[A^{\mathcal{DCBC\text{-}CS}[\mathrm{Real}]} \Rightarrow 1] - \Pr[A^{\mathcal{DCBC\text{-}CS}[\mathrm{Fake}]} \Rightarrow 1] \\ &= \Pr[B^{\mathrm{Real}} \Rightarrow 1] - \Pr[B^{\mathrm{Fake}} \Rightarrow 1] \\ &= \mathbf{Adv}^{\mathrm{IND\$}}_{\mathcal{E}}(B) \le \sigma^2/2^b\end{aligned}$$

appealing to Theorem 2 for the final inequality.

As before, one can immediately conclude the corresponding complexity-theoretic statement, which would read as follows.

Corollary 2. *Let $E\colon \mathcal{K} \times \{0,1\}^b \to \{0,1\}^b$ be a blockcipher and let $\mathcal{E}$ be any of the encryption schemes* DCBC-CS1$[E]$, DCBC-CS2$[E]$, *or* DCBC-CS3$[E]$. *Suppose A asks queries that total σ blocks, runs in time t, and achieves advantage $\delta = \mathbf{Adv}^{\mathrm{IND\$}}_{\mathcal{E}}(A)$. Then there is an adversary B, explicitly known and constructed from A in a blackbox manner, that asks at most σ queries, runs in time $t + \lambda b\sigma$, and achieves advantage $\mathbf{Adv}^{\mathrm{prp}}_{E}(B) \ge \delta - \sigma^2/2^b$. Here λ is an absolute constant depending only on details of the model of computation.*

6 Insecurity of the Meyer-Matyas CBC-CS

The CBC ciphertext-stealing construction by Meyer and Matyas, what we will call CBC-CSX, is defined in Fig. 7. This well-known scheme—it has been used since the early 1980's under the IBM CUSP architecture—is susceptible to a simple chosen-plaintext attack, a fact that appears not to have been pointed out before. Thus NIST did well in choosing not to standardize this form of ciphertext stealing, but the alternative, "correct" variant.

Here is an attack on the ind\$-security of CBC-CSX. The adversary makes two encryption queries: $M = 1^b 0^{b-1}$ and $M' = 1^b 0^{b-1}$. As the IV is randomized, asking the same plaintext twice is not without purpose. The oracle returns $C = C_0 C_1 C_2$ and $C' = C'_0 C'_1 C'_2$ where C_0 and C'_0 are randomly chosen IVs and $|C_1| = |C'_1| = b - 1$. If $C_2 = C'_2$ the adversary returns 1; otherwise, it returns 0. Now if the adversary is given a CBC-CSX oracle, the probability that $C_2 = C'_2$ is at least 1/2; otherwise, it's about $1/2^b$. Thus we have a trivial but effective ind\$-attack.

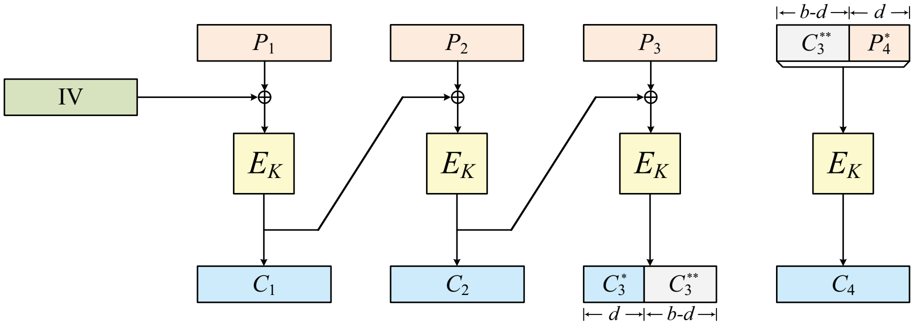
We remark that, not surprisingly, CBC-CSX is not secure under conventional, weaker notions of security, like left-or-right indistinguishability [3]; a similar attack can easily be described. It is not that the definition is too strong; from a modern point of view, the scheme is simply wrong.

Acknowledgments

Authors Rogaway and Zhang received support for this project under NSF grant CNS 0904380. Many thanks to the NSF for their support.

References

1. M. Ball. Follow-up to NIST's consideration of XTS-AES as standardized by IEEE Std 1619-2007. http://tinyurl.com/nist-ball-xts, Public comments to NIST, 2008.


$\leftarrow b\text{-}d \rightarrow$
$\leftarrow d \rightarrow$
P_1
P_2
P_3
C_3^{**}
P_4^*
IV
E_K
E_K
E_K
E_K
C_1
C_2
C_3^*
C_3^{**}
C_4
$\leftarrow d \rightarrow$
$\leftarrow b\text{-}d \rightarrow$

```
algorithm CBC-CSX$^{IV}_K(P)$
$n \leftarrow \lceil |P|/b \rceil$
$P_1 \cdots P_{n-1} P_n^* \leftarrow P$ where $|P_1| = \cdots = |P_{n-1}| = b$ and $|P_n^*| = d$
$C_0 \leftarrow IV$
for $i \leftarrow 1$ to $n-1$ do $C_i \leftarrow E_K(P_i \oplus C_{i-1})$
$C_n \leftarrow E_K((\mathrm{LSB}_{b-d}(C_{n-1}) \parallel P_n^*)$
$C_{n-1}^* \leftarrow \mathrm{MSB}_d(C_{n-1})$
return $C_0\, C_1 \cdots C_{n-2}\, C_{n-1}^*\, C_n$
```

Fig. 7. Mode CBC-CSX. The mode is insecure and should not be used. This version of ciphertext stealing is from Meyer and Matyas [14]. The mode depends on a blockcipher $E\colon \mathcal{K} \times \{0,1\}^b \to \{0,1\}^b$. That can be ideal, and the IV random, and still the mode will fail to achieve standard (CPA) privacy definitions.

2. G. Bard. Blockwise-adaptive chosen-plaintext attack and online modes of encryption. *Cryptography and Coding 2007*, LNCS 4887, Springer, pp. 129–151, 2007.
3. M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption: analysis of the DES modes of operation. *FOCS 97*, IEEE Press, pp. 394–403, 1997.
4. M. Bellare, T. Kohno and C. Namprempre. Breaking and provably repairing the SSH authenticated encryption scheme: a case study of the encode-then-encrypt-and-MAC paradigm. *ACM Transactions on Information and System Security* (TISSEC), 7:2, pp. 206–241, 2004. Earlier version from *CCS 2002*.
5. M. Bellare and P. Rogaway. Code-based game-playing proofs and the security of triple encryption. *EUROCRYPT 2006*, LNCS vol. 4004, Springer, pp. 409–426, 2006.
6. A. Boldyreva and N. Taesombut. Online encryption schemes: new security notions and constructions. *CT-RSA 2004*, LNCS vol. 2964, Springer, pp. 1–14, 2004.
7. M. Dworkin. Recommendation for block cipher modes of operation: method and techniques. NIST Special Publication 800-38A, 2001 Edition. December 2001.
8. M. Dworkin. Recommendation for block cipher modes of operation: three variants of ciphertext stealing for CBC mode. Addendum to NIST Special Publication 800–38A. October 2010.
9. P. Fouque, A. Joux, G. Martinet, and F. Valette. Authenticated on-line encryption. *SAC 2003*, LNCS vol. 3006, Springer, pp. 145–159, 2003.
10. P. Fouque, A. Joux and G. Poupard. Blockwise adversarial model for on-line ciphers and symmetric encryption schemes. *SAC 2004*, LNCS vol. 3357, Springer, pp. 212–226, 2004.
11. P. Fouque, G. Martinet, G. Poupard. Practical symmetric on-line encryption. *FSE 2003*, LNCS vol. 2887, Springer, pp. 362–375, 2003.
12. R. Gennaro and P. Rohatgi. How to sign digital streams. *CRYPTO 97*, LNCS vol. 1294, Springer, pp. 180–197, 1997.
13. A. Joux, G. Martinet, and F. Valette. Blockwise-adaptive attackers: revisiting the (in)security of some provably secure encryption models: CBC, GEM, IACBC. *CRYPTO 2002*, LNCS vol. 2442, Springer, pp. 17–30, 2002.
14. C. Meyer and M. Matyas. Cryptography: a new dimension in data security. *John Wiley & Sons*, New York, 1982.
15. NIST. Proposal to extend CBC mode by "ciphertext stealing." Anonymous draft, May 6, 2007. Available from NIST's website.
16. P. Rogaway, M. Bellare, and J. Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. In *ACM Transactions on Information and System Security*, 6:3, pp. 365–403, 2003. Earlier version, with T. Krovetz, in *ACM CCS 01*.

17. B. Schneier. *Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C*. New York, Wiley, 1996.
18. V. Shoup. Sequences of games: a tool for taming complexity. ePrint archive 2004/332. Revised 2006.
19. S. Vaudenay. Security flaws induced by CBC padding — applications to SSL, IPSEC, WTLS .... *EUROCRYPT 2002*, LNCS vol. 2332, Springer, pp. 534–545, 2002.

McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes

Ewan Fleischmann, Christian Forler, and Stefan Lucks

Bauhaus-University Weimar, Germany
{Ewan.Fleischmann, Christian.Forler, Stefan.Lucks}@uni-weimar.de

Abstract. On-Line Authenticated Encryption (OAE) combines privacy with data integrity and is on-line computable. Most block cipher-based schemes for Authenticated Encryption can be run on-line and are provably secure against *nonce-respecting* adversaries. But they fail badly for more general adversaries. This is not a theoretical observation only – in practice, the reuse of nonces is a frequent issue[1].

In recent years, cryptographers developed *misuse-resistant* schemes for Authenticated Encryption. These guarantee excellent security even against general adversaries which are allowed to reuse nonces. Their disadvantage is that encryption can be performed in an off-line way, only.

This paper considers OAE schemes dealing both with nonce-respecting *and* with general adversaries. It introduces McOE, an efficient design for OAE schemes. For this we present in detail one of the family members, McOE-X, which is a design solely based on a standard block cipher. As all the other member of the McOE family, it provably guarantees reasonable security against general adversaries as well as standard security against nonce-respecting adversaries.

Keywords: authenticated encryption, on-line encryption, provable security, misuse resistant

1 Introduction

On-Line Authenticated Encryption (OAE). Application software often requires a network channel that guarantees the privacy and authenticity of data being communicated between two parties. Cryptographic schemes able to meet both of these goals are commonly referred to as Authenticated Encryption (AE) schemes. The ISO/IEC 19772:2009 standard for AE [21] defines generic composition (Encrypt-then-MAC [4]) and five dedicated AE schemes: OCB2 [38], SIV [41] (denoted as “Key Wrap” in [21]), CCM [13], EAX [6], and GCM [34]. To integrate an AE-secure channel most seamlessly into a typical software architecture, application developers expect it to encrypt in an *on-line* manner meaning that the i-th ciphertext block can be written before the $(i+1)$-th plaintext block has to be read. A restriction to off-line encryption, where usually the entire plaintext must be known in advance (or read more than once) is an encumbrance to software architects.

Nonces and their reuse. Goldwasser and Micali [18] formalized encryption schemes as stateful or probabilistic, because otherwise important security properties are lost. Rogaway [37, 39, 40] proposed an unified point of view, by always defining a cryptographic scheme as a deterministic algorithm that takes an user supplied nonce (a *number used once*). So the application programmer – and not the encryption scheme – is responsible for flipping coins or maintaining state. This reflects cryptographic practice since the algorithm itself is often implemented by a multi-purpose cryptographic library which is more or less application-agnostic.

In theory, the concept of a nonce is simple. In practice, it is challenging to ensure that a nonce is *never* reused. Flawed implementations of nonces are ubiquitous [9, 20, 28, 44, 45]. Apart from implementation failures, there are fundamental reasons why software developers can't always

[1] A prominent example is the PlayStation 3 'jailbreak' [20], where application developers used a constant that was actually supposed to be a nonce for a digital signature scheme.

secure ...	against nonce-respecting adversaries	ag. nonce-reusing adversaries
on-line	CCFB[33] CHM[22] CIP[23] CWC[29] EAX[6] GCM[34] IACBC[26] IAPM[26] **McOE** OCB1-3[40, 38, 30] RPC[10] TAE[31] XCBC[17]	**McOE** (this paper)
off-line	BTM[24] CCM[13] HBS[25] SIV[41] SSH-CTR[36]	BTM[24] HBS[25] SIV[41]

Table 1. Classification of provably secure block cipher-based AE Schemes. CCM and SSH-CTR are considered off-line because encryption requires prior knowledge of the message length. Note that the family of McOE schemes, because of being on-line, satisfies a slightly weaker security definition against nonce-reusing adversaries than SIV, HBS, and BTM.

prevent nonce reuse. A persistently stored counter, which is increased and written back each time a new nonce is needed, may be reseted by a backup – usually after some previous data loss. Similarly, the internal and persistent state of an application may be duplicated when a virtual machine is cloned, etc.

Related Work and Our Contribution. We aim to achieve *both simultaneously*: security against nonce-reusing adversaries (sometimes also called nonce-misusing adversaries) *and* support for on-line-encryption in terms of an AE scheme. Apart from generic composition (Encrypt-then-Mac, EtM), none of the ISO/IEC 19772:2009 schemes – in fact, no previously published AE scheme at all – achieves both of these goals, cf. Table 1. In this table, we classify a vast variety of provably secure block cipher-based AE scheme with respect to their on-line-ability and against which adversaries (nonce-respecting versus -reusing) they are proven secure.

Since EtM is not a concrete scheme but merely a generic construction technique, there are some challenges left in order to make it full on-line secure: First, an appropriate on-line cipher has to be chosen. Second, a suitable, on-line computable, secure deterministic MAC must be selected. And, third, the EtM scheme requires at least two *independent* keys to be secure. Since two schemes are used in parallel, is likely to squander resources in terms of run time and – important for hardware designers – in terms of space. Since EtM first has to be turned into an OAE scheme by making the appropriate choices, we don't include it in our analysis.

As it turned out, we actually found nonce-reuse attacks for *all* of those schemes, cf. Table 2 and Appendix A. In this paper we present a new construction method for efficient AE schemes, called McOE-X, that is actually able to fill the apparent gap in the upper-right. It belongs to the family of McOE schemes [14]. We argue that closing this gap is both practically relevant and theoretically interesting.

Initial Value (IV) based AE schemes maximally forgiving of repeated IV's have been addressed in [41], coining the notion of "misuse resistance" and proposing SIV as a solution. SIV and related schemes (HBS [25] and BTM [24]) actually provide excellent security against nonce-reusing adversaries, though there are other potential misuse cases, cf. Appendix A.2. Their main disadvantage is that they are inherently off-line: For encryption, one must either keep the entire plaintext in memory, or read the plaintext twice.

Ideally, an adversary seeing the encryptions of two (equal-length) plaintexts P_1 and P_2 can't even decide if $P_1 = P_2$ or not. When using a nonce more than once, deciding about $P_1 = P_2$ is easy. SIV and its relatives ensure that nothing else is feasible for nonce-reusing adversaries. In the case of on-line encryption, where the first few bits of the encryption of a lengthy message must not depend on the last few bits of that message, there is unavoidably something beyond $P_1 = P_2$. The adversary can compare any two ciphertexts for their longest common prefix, and then conclude about common prefixes of the secret plaintexts. Our notion of *misuse resistance* means that this is all the adversary can gain. Even in the case of a nonce-reuse, the adversary

	privacy attack workload	authenticity attack workload
CCFB [33]	$O(1)$	$O(1)$
CCM [13]	$O(1)$	$\ll 2^{(n/2)}$ [15]
CHM [22]	$O(1)$	$O(1)$
CIP [23]	$O(1)$	$O(1)$
CWC [29]	$O(1)$	$O(1)$
EAX [6]	$O(1)$	$O(1)$
GCM [34]	$O(1)$	$O(1)$
IACBC [26]	$O(1)$	$O(1)$
IAPM [26]	$O(1)$	$O(1)$
OCB1 [40]	$O(1)$	$O(1)$
OCB2 [38]	$O(1)$	$O(1)$
OCB3 [30]	$O(1)$	$O(1)$
RPC [10]	$O(1)$	$O(1)$
TAE [31]	$O(1)$	$O(1)$
XCBC [17]	$O(2^{n/4})$	?

Table 2. Overview of our **nonce-reuse** attacks on published AE schemes, excluding SIV, HBS and BTM, which have been explicitly designed to resist nonce-reuse. Almost all attacks achieve an advantage close to 1. An "attack workload" of X means that the adversary is restricted to at most X units of time and at most X chosen texts. Details are given in Appendix A.

1. can't do anything beyond determining the length of common plaintext prefixes and
2. the scheme still provides the usual level of authenticity for AE (INT-CTXT).

The first property is common for on-line ciphers/permutations (OPRP) [1]. Recently, [43] studied the design of on-line ciphers from tweakable block ciphers bearing some similarities to our approach, especially to TC3. In contrast to the McOE family, the constructions from [43] provide no authentication. The McOE schemes are, *e.g.*, based on a normal block cipher *or* a tweakable block cipher.

Design Principles for AE Schemes. The question how to provide authenticated encryption (without stating that name) when given a secure on-line cipher is studied in [3], the revised and full version of [1]. The first idea in [3] only provides security if all messages are of the same length. The second idea repairs that by prepending the message's length to the message, at the cost of being off-line, since the message length must be known at the beginning of the encryption process. The third idea is to prepend and append a random W to a message M and then to perform the on-line encryption of $(W||M||W)$. This looks promising, but the same W is used for two different purposes, putting different constraints on the generation of W. For privacy, it suffices that W behaves like a nonce, not requiring secrecy or unpredictability. Even if W is not a nonce, but the same W is used for the encryption of several messages, all the adversary can determine are the lengths of common plaintexts prefixes, as we required for nonce-reuse. On the other hand, authenticity actually assumes a *secret or unpredictable* W, rather than a nonce. If the adversary can guess W before choosing a message, she asks for the authenticated encryption of $(M||W)$. Then she can predict the authenticated encryption of M without actually asking for it.

The McOE family replaces the "random" W by a proper nonce and a value τ which is *key-dependent*, performing a nonce-dependent on-line encryption of $(M||\tau)$. The encryption can also depend on some associated data, which turns McOE into a family of schemes for OAEAD *(On-Line Authenticated Encryption with Associated Data)*.

Roadmap. In this paper we focus on one member of the McOE [14] family of schemes called McOE-X. In Section 2 we describe a concrete block cipher based OAE scheme – called McOE-X– and provide performance data when McOE-X is instantiated with either AES-128 or Threefish-512 as the underlying block cipher. Section 3 deals with general notions and definitions, and Section 4 defines the security of OAE. The main result of the paper, the full McOE-X scheme and its analysis, is presented in Section 5. The discussion in Section 6 con-

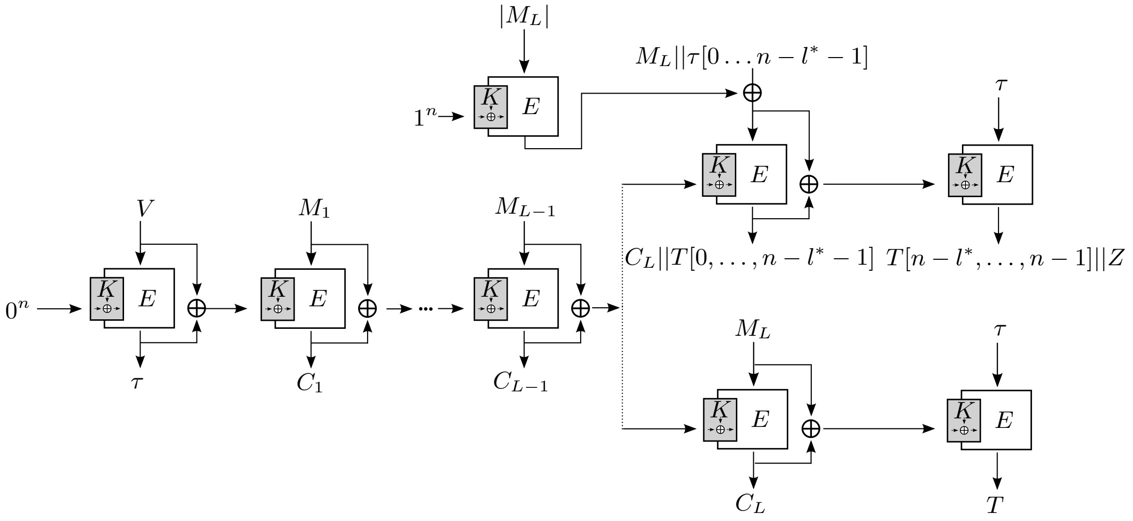


Fig. 1. The McOE-X-AES/McOE-X-Threefish encryption process. If, after the last complete message block has been encrypted, there is some incomplete block left, McOE-X performs tag-splitting (upper variant), Else, the tag can be computed without splitting (lower variant). The key used for the block cipher E is computed by the injective function $K \oplus W$ which is given the secret key K and the chaining value input W. The tag returned is the n-bit value T. The $n-l$-bit value Z is discarded. The decryption process works in a similar way from 'left to right' only the block cipher component E is replaced by its counterpart E^{-1} apart from one exception: the first call computing τ.

cludes the paper. The appendix deals with misuse attacks against published AE schemes, and provides some proof supplements.

2 Practical On-Line Authenticated Encryption using AES and Threefish

We start with the fruits of our analysis by giving two concrete instances of OAE schemes (illustrated in Figure 1) including performance data and reference source code[2]. One instance, McOE-X-AES uses AES-128 as the core component while McOE-X-Threefish uses the block cipher Threefish-512, a cipher with 512-bit block size and key size, which is the core working component inside the SHA-3 finalist Skein[35].

We also introduce the *tag-splitting* (TS) method for processing messages whose length is not a multiple of the block length. Without TS, we would have to pad such messages and then encrypt the padded messages – resulting in an expanded ciphertext. The effect of TS is similar to the well-known length preserving method called *ciphertext stealing* (CTS), *e.g.* [12]. But the technique itself is quite different since CTS requires to process the last block before the last but one, which is not possible for McOE-X.

Let E_K be a block cipher taking a k-bit key K and a plaintext/ciphertext of size n-bit. Note that for our chosen instances, AES-128 and Threefish-512, we have $n = k$. The pseudo code for these two McOE-X instances is given in Table 3 – on the upper side without TS, on the lower side with TS.

The algorithms without TS, **EncryptAuthenticate** and **DecryptAuthenticate**, are simplified algorithms for messages that are aligned on n-bit boundaries, *i.e.* $M = (M_1, \ldots, M_L) \in (\{0,1\}^n)^L$ for some integer L. The TS-variants **EncryptAuthenticateSplitTag** and **DecryptAuthenticateSplitTag**, can handle arbitrarily sized messages, *i.e.*, $M = (M_1, \ldots, M_L) \in (\{0,1\}^n)^{L-1}||\{0,1\}^{l^*}$ where L and l^* are integers with $0 < l^* < n$ and $'||'$ denotes the string concatenation operator. See Figure 1 and Table 3.

[2] The reference source code is available on request; it will be published as open source.

```
**EncryptAuthenticate**$(V, M)$
$\tau \leftarrow E_K(V)$
$U \leftarrow V \oplus \tau \oplus K$
**for** $i = 1, \ldots, L$ **loop**
    $C_i \leftarrow E_U(M_i)$
    $U \leftarrow M_i \oplus C_i \oplus K$
$T \leftarrow E_U(\tau)$
**return** $(C_1, \ldots, C_L, T)$
```

```
**DecryptAuthenticate**$(V, C, T)$
$\tau \leftarrow E_K(V)$
$U \leftarrow V \oplus \tau \oplus K$
**for** $i = 1, \ldots, L$ **loop**
    $M_i \leftarrow E_U^{-1}(C_i)$
    $U \leftarrow M_i \oplus C_i \oplus K$
**if** $T = E_U(\tau)$ **then**
    **return** $(M_1, \ldots, M_L)$
    **else return** $\perp$
```

```
**EncryptAuthenticateSplitTag**$(V, M)$
$\tau \leftarrow E_K(V)$
$U \leftarrow V \oplus \tau \oplus K$
**for** $i = 1, \ldots, L-1$ **loop**
    $C_i \leftarrow E_U(M_i)$
    $U \leftarrow M_i \oplus C_i \oplus K$
$M^* \leftarrow (M_L || \tau[0 \ldots n - l^* - 1])$
$M^* \leftarrow M^* \oplus E_{K \oplus 1^n}(|M_L|)$
$C^* \leftarrow E_U(M^*)$
Parse $C_L || T[0 \ldots n - l^* - 1] \leftarrow C^*$
$U \leftarrow M^* \oplus C^* \oplus K$
$C^{**} \leftarrow E_U(\tau)$
$T[n - l^* \ldots n - 1] \leftarrow C^{**}[0 \ldots l^* - 1]$
**return** $(C_1, \ldots, C_{L-1}, C_L^*, T)$
```

```
**DecryptAuthenticateSplitTag**$(V, C, T)$
$\tau \leftarrow E_K(V)$
$U \leftarrow V \oplus \tau \oplus K$
**for** $i = 1, \ldots, L-1$ **loop**
    $M_i \leftarrow E_U^{-1}(C_i)$
    $U \leftarrow M_i \oplus C_i \oplus K$
$C^* \leftarrow C_L || T[0 \ldots n - l^* - 1]$
$M^* \leftarrow E_U^{-1}(C^*)$
$U \leftarrow M^* \oplus C^* \oplus K$
$M^* \leftarrow M^* \oplus E_{K \oplus 1^n}(|C_L|)$
Parse $M_L || \tau'[0 \ldots n - l^* - 1] \leftarrow M^*$
$T' \leftarrow E_U(\tau)$
**if** $\tau'[0 \ldots n - l^* - 1] = \tau[0 \ldots n - l^* - 1]$
    **and** $T'[0 \ldots l^* - 1] = T[n - l^* \ldots n - 1]$
    **then return** $(M_1, \ldots, M_L)$ **else return** $\perp$
```

Table 3. Instances of McOE-X: upper side is for messages whose size is evenly divisible by the block size n; Lower side is for arbitrarily sized messages (TS-variant); see text for details

In addition to McOE-X, we introduce two further authenticated encryption schemes following the McOE design principles. The first one is called McOE-D and is based on the THC-CBC construction [7]. The ratio of this scheme is 2-1, *i.e.* the block cipher is invoked twice to encipher resp. decipher one message block. The second one is called McOE-G and is based on the HCBC-2 construction [2]. This scheme updates the chaining value by invoking a universal hash function, *i.e.*, a n-bit Galois-Field multiplication.

Remarks. For McOE-X we actually do need related key resistance for the block cipher E since the adversary can 'partially control' some relations among keys used in the computation. This is not true for the other mentioned constructions.

All McOE schemes are easily extended to smoothly handle associated data, *i.e.* data that is not encrypted but only authenticated. This is discussed in more detail in Section 5.

3 On-Line Authenticated Encryption and Related Notions

3.1 Definitions

Length of Longest Common Prefix (LLCP$_n$). The length of a string $x \in \{0,1\}^n$ is denoted by $|x| := n$. For integers $n, \ell, d \geq 1$, set $D_n^d = (\{0,1\}^n)^d$, and $D_n^* := \bigcup_{d \geq 0} D_n^d$, and $D_{\ell,n} = \bigcup_{0 \leq d \leq \ell} D_n^d$. Note that D_n^0 only contains the empty string. For $M \in D_n^d$, we write $M = (M_1, \ldots, M_d)$ with $M_1, \ldots, M_d \in D_n$. For $P, R \in D_n^*$, say, $P \in D_n^p$ and $R \in D_n^r$, we define the *length of the longest common n-prefix* of P and R as

$$\mathrm{LLCP}_n(P, R) = \max_i \{P_1 = R_1, \ldots, P_i = R_i\}.$$

Block cipher	Impl.	Message length in Bytes									
		64	128	256	512	1024	2048	4096	8192	16384	32768
McOE-X-AES	software	31.2	26.3	23.9	22.7	22	21.7	21.6	21.5	21.5	21.5
McOE-X-AES	AES-NI	14.2	12.2	11.2	10.7	10.5	10.4	10.4	10.3	10.3	10.3
McOE-X-Threefish	software	19.5	13.1	9.9	8.3	7.5	7.1	6.9	6.8	6.8	6.7
McOE-D-AES	software	40.1	33	29.4	27.6	26.7	26.3	26.1	25.9	25.9	25.9
McOE-D-AES	AES-NI	11.6	9.9	8.3	7.2	6.7	6.4	6.3	6.3	6.2	6.2
McOE-G-AES	software	33	27.9	25.4	24.1	23.5	23.2	23	22.9	22.8	22.8
McOE-G-AES	GF-NI/AES-NI	12.5	10.6	9.7	9.3	9	8.9	8.9	8.8	8.8	8.8
AES-CBC encryption	software	38.3	35.9	13.5	13.3	13.2	13.2	13.1	13.1	13.1	13.1
AES-CBC encryption	AES-NI	4	3.7	3.6	3.5	3.5	3.5	3.5	3.5	3.5	3.5

Table 4. Performance values (cycles-per-byte, single core), measured on an Core i5 540M for AES-128 and Threefish-512. McOE-X is the main contribution in the current paper, McOE-D invokes the underlying block cipher twice and McOE-G uses Galois field arithmetic. For a comparsion, we also provide the performance of unauthenticated AES-CBC. The AES software implementation is based on Gladman [16], whereas the hardware implementation is based on the Intel AES-NI Sample Library[11]. The Threefish implementation is based on the NIST/SHA-3 reference source as provided by the Skein authors [35]. Finally, the implementation of Galois field NI multiplication (GF-NI) is based on the example-code from [19].

For a non-empty set $\mathcal{Q}$ of strings in D_n^* we define $\mathrm{LLCP}_n(\mathcal{Q}, P)$ as $\max_{q \in \mathcal{Q}}\{\mathrm{LLCP}_n(q, P)\}$. For example, if $P \in \mathcal{Q}$, then $\mathrm{LLCP}_n(\mathcal{Q}, P) = |P|/n$.

For convenience, we introduce a notation for a *restriction on a set.* If $\mathcal{Q} = \{0,1\}^a \times \{0,1\}^b \times \{0,1\}^c$, we write $\mathcal{Q}_{|b,c} = \{(B, C) \,|\, \exists A : (A, B, C) \in \mathcal{Q}\}$. This generalizes in the obvious way.

3.2 Block Ciphers and On-Line Permutations

Block Ciphers. An (k, n) block cipher is a keyed family of permutations consisting of two paired algorithms $E : \{0,1\}^k \times D_n \to D_n$ and $E^{-1} : \{0,1\}^k \times D_n \to D_n$, accepting a k-bit key and an input from D_n for some $k, n > 0$. For $n > 0$, $Block(k, n)$ is the set of all (k, n) block ciphers. For any $E \in Block(k, n)$ and a fixed key $K \in \{0,1\}^k$, the decryption $E_K^{-1}(Y) := E^{-1}(K, Y)$ is the inverse function of encryption $E_K(X) := E(K, X)$, so that $E_K^{-1}(E_K(X)) = X$ holds for any $X \in D_n$.

We follow the usual convention to write oracles, that are provided to an algorithm, as superscripts. We define the related key PRP-security of a block cipher E by the success probability of an adversary trying to differentiate between the block cipher and a random permutation.

Definition 1. *Let $E \in Block(k, n)$ and denote by E^{-1} the corresponding inverse. Let $\varphi : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^k$. A fixed related key adversary A has access to an E oracle with two parameters such that she can query either $E_{\varphi(K,\cdot)}(\cdot)$ or its inverse. Let* $\mathrm{PERM}(n, n)$ *be the set of n-bit permutations such that the first parameter models the permutation and the second parameter the value that is to be permuted,* i.e. *for $\pi \in \mathrm{PERM}(n, n)$ it holds that $\pi(Z, \cdot)$ is a random permutation for any given value of Z. The related-key (RK) advantage [32] of A in breaking E is then defined as*

$$\mathbf{Adv}_E^{\text{RK-CPA-PRP}}(A) = \left|\Pr[K \xleftarrow{\$} \{0,1\}^k : A^{E_{\varphi(K,\cdot)}(\cdot)} \Rightarrow 1] - \Pr[\pi \xleftarrow{\$} \mathit{Perm(n,n)} : A^{\pi(\cdot,\cdot)} \Rightarrow 1]\right|$$

$$\mathbf{Adv}_{E,E^{-1}}^{\text{RK-CCA-PRP}}(A) = \Big|\Pr[K \xleftarrow{\$} \{0,1\}^k : A^{E_{\varphi(K,\cdot)}(\cdot), E^{-1}_{\varphi(K,\cdot)}(\cdot)} \Rightarrow 1]$$
$$- \Pr[\pi \xleftarrow{\$} \mathit{Perm(n,n)} : A^{\pi(\cdot,\cdot), \pi^{-1}(\cdot,\cdot)} \Rightarrow 1]\Big|.$$

On-Line Permutations. We aim for larger permutations that not only permute single blocks but can handle multiple/variable block messages. Such a permutation, from D_n^* to D_n^*, is (n-)on-line if the i-th block of the output is determined completely by the first i blocks of the input.

Definition 2. *Let $n, k \geq 0$, $K \in \{0,1\}^k, V \in D_n$. A function $\Pi : \{0,1\}^k \times D_n^* \to D_n^*$ is an (n-)on-line permutation if for any fixed K, V the function $\Pi(K, V, \cdot)$ is a permutation and there exists for any message $M = (M_1, M_2, \ldots, M_m)$ a family of functions $\widetilde{\pi}^i : \{0,1\}^k \times \{0,1\}^n \times D_n^i \to D_n$, $i = 1, \ldots, m$ such that*

$$\begin{aligned}\Pi(K,V,M) = \ & \widetilde{\pi}_K^1(V, M_1) || \widetilde{\pi}_K^2(V, M[1..2]) \\ & || \ldots || \\ & \widetilde{\pi}_K^{m-1}(V, M[1..m-1]) || \widetilde{\pi}_K^m(V, M[1..m]),\end{aligned}$$

where $M[a \ldots b] := M_a || M_{a+1} || \ldots || M_b$ with "$||$" being the concatenation of strings, holds.

An encryption scheme is (n-)on-line if the encryption function is (n-)on-line. A thorough discussion of on-line encryption and its properties can be found in [1].

3.3 Authenticated Encryption (With Associated Data)

An authenticated encryption scheme is a tuple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. Its aim is to provide privacy and data integrity. The key generation function $\mathcal{K}$ takes no input and returns a randomly chosen key K from the key space, *e.g.* from $\{0,1\}^k$. The encryption algorithm $\mathcal{E}$ and the decryption algorithm $\mathcal{D}$ are deterministic algorithms that map values from $\{0,1\}^k \times \mathcal{H} \times D_n^*$ to a string or – if the input is invalid – the value $\perp$. The header $\mathcal{H}$ consists either only of the initial value/nonce $V \in D_n$ (if no data is to be authenticated/checked in the encryption/decryption process) or is a combination of V and a value from D_n^*. So $\mathcal{H} \subset D_n^+$ in either case. For sake of convenience, we usually write $\mathcal{E}_K^H(M)$ for $\mathcal{E}(K, H, M)$ and $\mathcal{D}_K^H(M)$ for $\mathcal{D}(K, H, M)$, where the message M is chosen from D_n^*, $H \in \mathcal{H}$ and a key from the key space. We require $\mathcal{D}_K^{\mathcal{H}}(\mathcal{E}_K^{\mathcal{H}}(M)) = M$ for any possible K, M, H, and define the tag size for a message $M \in D_n^*$ and header $H \in \mathcal{H}$ as $\textsc{tag}(H, M) := |\mathcal{E}_K^H(M)| - |M|$. We denote an authenticated encryption scheme with the requirement that the initial vector V is only used once in a *nonce based* scheme. Otherwise, we call such a scheme *deterministic*. Similarly, we call an adversary *nonce-respecting* (NR) if no nonce is used twice for any query. Otherwise, the adversary is called *nonce-ignoring* (NI).

4 Security Notions for On-Line Authenticated Encryption

Authenticated (On-Line) Encryption tries to achieve privacy and authenticity at the same time. Therefore we need security notions to handle this twofold goal. For AE, there have been notions and their relations introduced for deterministic [42] and nonce based [4, 5, 27, 37, 40] AE schemes. In order to have one convenient toolset of notions, we adopt the notion of CCA3 security suggested in [42] as a *natural strengthening* of CCA2 security.

We parameterize our definition in order to define different – but closely related – notions by explicitly stating whether we mean an on-line or off-line scheme, $\omega \in \{\textsc{ae}, \textsc{oae}\}$ and stating the adversary behavior as either nonce-respecting or nonce-ignoring, $\nu \in \{\textsc{nr}, \textsc{ni}\}$.

Definition 3 (CCA3(ω, ν)). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an authenticated encryption scheme with header space $\mathcal{H}$ and message space D_n^*, and fix an adversary A. The advantage of A breaking Π is defined as*

$$\mathbf{Adv}_{\Pi}^{\mathrm{CCA3}(\omega,\nu)}(A) = \left| \Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{E}_K(\cdot,\cdot), \mathcal{D}_K(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[A^{\$^{\omega}(\cdot,\cdot), \perp(\cdot,\cdot)} \Rightarrow 1\right] \right|.$$

```
Game $G_{\text{CPA}}$, $\boxed{G_{\text{CCA3}}}$

Initialize($\omega, \nu$)
    b $\xleftarrow{\$}$ {0,1};
    if(b=1) then
        K $\leftarrow \mathcal{K}()$;

Finalize($d$)
    return ($b = d$);

Encrypt($H, M$)
    if ($\nu = \text{NR}$ and $V \in B$) then
        return $\perp$;
    if(b=1) then
        $C \leftarrow \mathcal{E}_K$(H,M);
    else
        $C \leftarrow \$^{\omega}(H, M)$;
    $B \leftarrow B \cup \{V\}$;
    $\boxed{\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(H, C)\};}$
    return C;

Decrypt(H,C)
    if (($H, C) \in Q$) then
        return $\perp$;
    if(b=1) then
        M $\leftarrow \mathcal{D}_K$(H,C);
    else
        M $\leftarrow \perp$(H,C);
    return M;
```

Fig. 2. $G_{\text{CPA}}(\omega, \nu)$ is the $\text{CPA}_{\Pi}^{(\omega,\nu)}$-Game and $G_{\text{CCA3}}(\omega, \nu)$ the $\text{CCA3}_{\Pi}^{(\omega,\nu)}$-Game where $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. Game G_{CCA3} contains the code in the box while G_{CPA} does not. The oracle $\$^{\text{AE}}(H, M)$ returns a string of length $|M| + \text{TAG}(H, M)$, this string is on-line compatible if $\omega = \text{OAE}$. V denotes the last block of the header representing the nonce/initial value.

The adversary's random-bits oracle, $\$^{\text{AE}}(\cdot, \cdot)$ or $\$^{\text{OAE}}(\cdot, \cdot)$, returns on a query with header $H \in \mathcal{H}$ and plaintext $X \in D_n^*$ a random string of length $|\mathcal{E}_K(M)|$ which is either on-line or not, depending on the variable ω. The $\perp(\cdot, \cdot)$ oracle returns $\perp$ on every input. We assume *wlog.* that the adversary A never ask a query which answer is already known. It is easy to see that we can rewrite the term given in Definition 3 as

$$\mathbf{Adv}_{\Pi}^{\text{CCA3}(\omega,\nu)}(A) = |\Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{E}_K(\cdot,\cdot),\perp(\cdot,\cdot)} \Rightarrow 1\right] \quad (1)$$

$$+ \Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{E}_K(\cdot,\cdot),\perp(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[A^{\$^{\omega}(\cdot,\cdot),\perp(\cdot,\cdot)} \Rightarrow 1\right]|. \quad (2)$$

One can interpret (1) as the advantage that an adversary has on the integrity of the ciphertext and (2) as the advantage that an CPA adversary has on the privacy. Using this decomposition as a motivational starting point, we now define ciphertext integrity and what we mean by a CPA adversary on authenticated encryption schemes. From now on, our definitions are based on the game playing methodology. For example, we can restate Definition 3 using the game G_{CCA3} given in Figure 2 as

$$\mathbf{Adv}_{\Pi}^{\text{CCA3}(\omega,\nu)}(A) = 2|\Pr[A^{G_{\text{CCA3}(\omega,\nu)}} \Rightarrow 1] - 0.5|.$$

We denote $\mathbf{Adv}_{\Pi}^{\text{CCA3}(\omega,\nu)}(q, t, \ell)$ as the maximum advantage over all $\text{CCA3}(\omega, \nu)$ adversaries run in time at most t, ask a total maximum of q queries to $\mathcal{E}$ and $\mathcal{D}$, and whose total query length is not more than ℓ blocks.

4.1 Privacy and Integrity Notions for Authenticated Encryption Schemes.

Similarly, we define the privacy and integrity of an authenticated (on-line) encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with header space D_n^+, message space D_n^* and tag-size function $\text{TAG}(H, M)$ as follows.

Definition 4. *Let $G_{CPA}(\omega, \nu)$ be the $CPA_{\Pi}^{\omega,\nu}$ game given in Figure 2. Fix an adversary A. The advantage of A breaking Π is defined as*

$$\mathbf{Adv}_{\Pi}^{CPA(\omega,\nu)}(A) \leq 2|\Pr[A^{G_{CPA}(\omega,\nu)} \Rightarrow 1] - 0.5|.$$

Definition 5. *Let $G_{\text{INT-CTXT}}(\nu)$ be the $\text{INT-CTXT}_{\Pi}^{\nu}$ game given in Figure 3. Fix an adversary A. The advantage of A breaking Π is defined as*

$$\mathbf{Adv}_{\Pi}^{\text{INT-CTXT}(\nu)}(A) \leq \Pr[A^{G_{\text{INT-CTXT}}(\nu)} \Rightarrow 1].$$

We denote $\mathbf{Adv}_{\Pi}^{CPA(\omega,\nu)}(q, t, \ell)$ and $\mathbf{Adv}_{\Pi}^{\text{INT-CTXT}(\nu)}(q, t, \ell)$ as the maximum advantage over all $CPA(\omega, \nu)$ resp. $\text{INT-CTXT}(\nu)$ adversaries run in time at most t, ask a total maximum of q queries to $\mathcal{E}$ and $\mathcal{D}$, and whose total query length is not more than ℓ blocks.

```
Game $G_{INT-CTXT}$

Initialize($\nu$)
    $K \leftarrow \mathcal{K}()$;

Finalize()
    return win;

Encrypt(H,M)
    if ($\nu$ = NR and $V \in B$) then
        return $\bot$;
    $C \leftarrow \mathcal{E}_K$(H,M);
    $B \leftarrow B \cup \{V\}$;
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(H, C)\}$;
    return $C$;

Verify(H,C)
    M $\leftarrow \mathcal{D}_K$(H,C);
    if $((H,C) \notin Q$ and $M \neq \bot)$ then
        win $\leftarrow$ true;
    return $(M \neq \bot)$;
```

Fig. 3. Game $G_{INT-CTXT}(\nu)$ is the INT-CTXT$_{\Pi}^{\omega,\nu}$ game where $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. V denotes the last block of the header representing the nonce/initial value.

4.2 CCA3 is equal to INT-CTXT plus CPA.

We now give a generalization of Theorem 3.2 from Bellare and Namprempre [4]. It simply states the equivalence of a scheme being CCA3 secure and both INT-CTXT and CPA secure. These statements hold in the on-line and offline case.

Theorem 1. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an authenticated encryption scheme. Fix $\omega \in \{\text{AE}, \text{OAE}\}$ and $\nu \in \{\text{NR}, \text{NI}\}$. Let A be an* CCA3$(\omega, \nu)_\Pi$*-adversary running in time t, making q queries with a total length of at most ℓ blocks. Then there are a* CPA(ω, ν)*-adversary A_p and an* INT-CTXT(ω, ν)*-adversary A_c such that*

$$\mathbf{Adv}_{\Pi}^{\text{CCA3}(\omega,\nu)}(A) \leq \mathbf{Adv}_{\Pi}^{\text{CPA}(\omega,\nu)}(A_p) + \mathbf{Adv}_{\Pi}^{\text{INT-CTXT}(\omega,\nu)}(A_c).$$

Furthermore, A_c and A_p run in time $O(t)$ and both make at most q queries in each case.

The proof is given in Appendix B.

5 The On-Line Authenticated Encryption Scheme McOE-X

In this section, we present McOE-X, a construction for an OAE scheme. We prove that McOE-X achieves our two-fold goal. First, it guarantees a certain minimum, well defined, security against a nonce-ignoring adversary. And, second, we show – in the full version of the paper [14] – that the complete McOE family of OAE schemes (including McOE-X) is fully secure against a nonce-respecting adversary.

Since we already have presented two McOE-X instances in Section 2, we proceed by formally defining McOE-X and giving its pseudocode. Indeed this is very similar to the results presented in Section 2, but here our definitions are slightly more general. Instead of fixing the key computation function to $K \oplus V$, where R is the chaining value and K the secret key, we here use a key derivation function $\varphi(K, R)$. By this we make sure that our proof also works for tweakable block ciphers - with K as key and R as tweak - leading to more efficient design.

Definition 6 (McOE-X). *Let $k, n \in \mathbb{N}$ with $k \geq n$, $E \in$ Block(k, n), and $\varphi : \{0,1\}^k \times \{0,1\}^v \to \{0,1\}^k$ such that $\varphi(K, \cdot)$ is injective. The encryption function takes a header $H \in D_n^{L_H}$, a message M and returns a ciphertext C and a tag $T \in D_n$. The decryption function takes a header $H \in D_n^{L_H}$, a ciphertext C and a tag $T \in D_n$ and returns either a plaintext M or the fail symbol $\bot$.*

(i) 'Non-TS'. *Let $M, C \in D_N^L$ for some integer L, then* McOE-X *is defined by the algorithms* ***EncryptAuthenticate*** *and* ***DecryptAuthenticate*** *given in Table 5.*

(ii) 'TS'. *Let $M, C \in D_N^L || \{0,1\}^{l^*}$ for some integers L and l^*, $0 < l^* < n$, then* McOE-X*/TS is defined by the algorithms* ***EncryptAuthenticateSplitTag*** *and* ***DecryptAuthenticate-SplitTag*** *given in Table 5.*

```
**EncryptAuthenticate**$(H, M)$
$U \leftarrow \varphi(K, 0^n)$
for $i = 1, \ldots, L_H - 1$ do
    $U \leftarrow \varphi(K, H_i \oplus E_U(H_i))$
$\tau \leftarrow E_U(H_{L_H})$
$U \leftarrow \varphi(K, H_{L_H} \oplus \tau)$
for $i = 1, \ldots, L$ do
    $C_i \leftarrow E_U(M_i)$
    $U \leftarrow \varphi(K, M_i \oplus C_i)$
$T \leftarrow E_U(\tau)$
return $(C_1, \ldots, C_L, T)$
```

```
**DecryptAuthenticate**$(H, C, T)$
$U \leftarrow \varphi(K, 0^n)$
for $i = 1, \ldots, L_H - 1$ do
    $U \leftarrow \varphi(K, H_i \oplus E_U(H_i))$
$\tau \leftarrow E_U(H_{L_H})$
$U \leftarrow \varphi(K, H_{L_H} \oplus \tau)$
for $i = 1, \ldots, L$ do
    $M_i \leftarrow E_U^{-1}(C_i)$
    $U \leftarrow \varphi(K, M_i \oplus C_i)$
if $T = E_U(\tau)$ then
return $(M_1, \ldots, M_L)$ else return $\perp$
```

```
**EncryptAuthenticate**$(H, C, T)$
$U \leftarrow \varphi(K, 0^n)$
for $i = 1, \ldots, L_H - 1$ do
    $U \leftarrow \varphi(K, H_i \oplus E_U(H_i))$
$\tau \leftarrow E_U(H_{L_H})$
$U \leftarrow \varphi(K, H_{L_H} \oplus \tau)$
for $i = 1, \ldots, L - 1$ do
    $C_i \leftarrow E_U(M_i)$
    $U \leftarrow \varphi(K, M_i \oplus C_i)$
$M^* \leftarrow M_L || \tau[0 \ldots n - l^* - 1]$
$M^* \leftarrow M^* \oplus E_{K \oplus 1^n}(|M_L|)$
$C^* \leftarrow E_U(M^*)$
Parse $C_L || T[0 \ldots n - l^* - 1] \leftarrow C^*$
$U \leftarrow \varphi(K, M^* \oplus C^*)$
$C^{**} \leftarrow E_U(\tau)$
$T[n - l^* \ldots n - 1] \leftarrow C^{**}[0 \ldots l^* - 1]$
return $(C_1, \ldots, C_L, T)$
```

```
**DecryptAuthenticateSplitTag**$(H, C, T)$
$U \leftarrow \varphi(K, 0^n)$
for $i = 1, \ldots, L_H - 1$ do
    $U \leftarrow \varphi(K, H_i \oplus E_U(H_i))$
$\tau \leftarrow E_U(H_{L_H})$
$U \leftarrow \varphi(K, H_{L_H} \oplus \tau)$
for $i = 1, \ldots, L$ do
    $M_i \leftarrow E_U^{-1}(C_i)$
    $U \leftarrow \varphi(K, M_i \oplus C_i)$
$C^* \leftarrow C_{L+1} || T[0 \ldots n - l^* - 1]$
$M^* \leftarrow E_U^{-1}(C^*)$
$U \leftarrow \varphi(K, M^* \oplus C^*)$
$M^* \leftarrow M^* \oplus E_{K \oplus 1^n}(|C_L|)$
Parse $M_L || \tau'[0 \ldots n - l^* - 1] \leftarrow M^*$
$T' \leftarrow E_U(\tau)$
if $\tau'[0 \ldots n - l^* - 1] = \tau[0 \ldots n - l^* - 1]$
    and $T'[0 \ldots l^* - 1] = T[n - l^* \ldots n - 1]$
    then return $(M_1, ..., M_L)$ else return $\perp$
```

Table 5. Instances of McOE-X: Left side is for messages whose size is evenly divisible by the block size n; Right side is for arbitrarily sized messages (TS-variant); see text for details

We now proceed to show the security of McOE-X. For this we use the results of Theorem 1 and show the INT-CTXT and RK-CPA-PRP security separately.

Theorem 2.

(i) *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a* McOE-X *scheme as in Definition 6 (i),* i.e. *$\mathcal{K}$ is the key derivation function, $\mathcal{E}$ =* **EncryptAuthenticate** *and $\mathcal{D}$ =* **DecryptAuthenticate**. *We further assume that the block cipher E is secure against related key attacks. Then*

$$\mathbf{Adv}_{\Pi}^{\mathrm{CCA3(OAE,NI)}}(q, \ell, t) \leq \frac{2(q+\ell)(q+\ell+1) + 3q + 2\ell}{2^n - (q+\ell)} + 3\mathbf{Adv}_{E,E^{-1}}^{\text{RK-CCA-PRP}}(q+\ell).$$

(ii) *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a* McOE-X *scheme as in Definition 6 (ii),* i.e. *$\mathcal{K}$ is the key derivation function, $\mathcal{E}$ =* **EncryptAuthenticateSplitTag** *and $\mathcal{D}$ =* **DecryptAuthenticateSplitTag**. *We further assume that the block cipher E is secure against related key attacks. Then*

$$\mathbf{Adv}_{\Pi}^{\mathrm{CCA3(OAE,NI)}}(q, \ell, t) \leq \frac{4(q+\ell+2)(q+\ell+3) + 6(2q+\ell)}{2^n - (q+\ell)} + \frac{3q(q+1)}{2^n - q} + \frac{q}{2^{n/2} - q} + 3\mathbf{Adv}_{E,E^{-1}}^{\text{RK-CCA-PRP}}(2q+\ell).$$

Proof. The proof of (i) follows from Theorem 1 together with Lemmas 1 and 2. Due to the lack of of space the proof of (ii) it is skipped here and is available in the full version of the paper [14].

Lemma 1. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a* McOE-X *scheme as in Definition 6 (i). Let q be the number of total queries an adversary A is allowed to ask and ℓ be an integer representing the total length in blocks of the queries to $\mathcal{E}$ and $\mathcal{D}$. Then,*

$$\mathbf{Adv}_{\Pi}^{\text{INT-CTXT(NI)}}(q, \ell, t) \leq \frac{(q+\ell)(q+\ell+1)}{2^n - (q+\ell)} + \frac{2q+\ell}{2^n - (q+\ell)} + \mathbf{Adv}_{E,E^{-1}}^{\text{RK-CCA-PRP}}(q+\ell).$$

```
Initialize()
  $K \xleftarrow{\$} \mathcal{K}()$;
  $B \leftarrow \{\varphi(K, 0^n\}$;

Encrypt(H, M)  Game $G_1$
  $L_H \leftarrow |H|/n$;  $L \leftarrow |M|/n$;
  $U \leftarrow \varphi(K, 0^n)$;
  for $i = 1, ..., L_H$ do
    $\tau \leftarrow E_U(H_i)$;
    $U \leftarrow \varphi(K, H_i \oplus \tau)$;
  for $i = 1, ..., L$ do
    $C_i \leftarrow E_U(M_i)$;
    $U \leftarrow \varphi(K, C_i \oplus M_i)$;
  $T \leftarrow E_U(\tau)$;
  $\mathcal{Q} \leftarrow (H, M, C, T)$;
  return $(C_1, \ldots, C_L, T)$;

Encrypt(H, M)  Game $G_2$, $\boxed{G_3}$
  $L_H \leftarrow |H|/n$;  $L \leftarrow |M|/n$;
  $A \leftarrow A \cup H$;
  $p \leftarrow \text{LLCP}_n(\mathcal{Q}_{|H,M}, (H, M))$;
  $U \leftarrow \varphi(K, 0^n)$;
  for $i = 1, \ldots, L_H$ do
    $\tau \leftarrow E_U(H_i)$;
    $U \leftarrow \varphi(K, H_i \oplus \tau)$;
    if $(U \in B$ and $i > p)$ then
      bad $\leftarrow$ true;  $\boxed{U \xleftarrow{\$} \{0,1\}^n \setminus B;}$
    $B \leftarrow B \cup U$;
  for $i = 1, \ldots, L$ do
    $C_i \leftarrow E_U(M_i)$;
    $U \leftarrow \varphi(K, C_i \oplus M_i)$;
    if $(U \in B$ and $i + L_H > p)$ then
      bad $\leftarrow$ true;  $\boxed{U \xleftarrow{\$} \{0,1\}^n \setminus B;}$
    $B \leftarrow B \cup U$;
  $T \leftarrow E_U(\tau)$;
  $\mathcal{Q} \leftarrow (H, M, C, T)$;
  return $(C_1, \ldots, C_L, T)$;
```

```
Finalize()
  return win;

Verify(H, C, T)  Game $G_1$
  $L_H \leftarrow |H|/n$;  $L \leftarrow |C|/n$;
  $U \leftarrow \varphi(K, 0^n)$;
  for $i = 1, ..., L_H$ do
    $\tau \leftarrow E_U(H_i)$;
    $U \leftarrow \varphi(K, H_i \oplus \tau)$;
  for $i = 1, ..., L$ do
    $M_i \leftarrow E_U^{-1}(C_i)$;
    $U \leftarrow \varphi(K, C_i \oplus M_i)$;
  if $(T = E_U(\tau)$ and $(H, C) \notin \mathcal{Q}_{|H,C})$ then
    win $\leftarrow$ true;
  $\mathcal{Q} \leftarrow (H, \perp, C, \perp)$;
  return $(T = E_U(\tau))$

Verify(H, C, T)  Game $G_2$, $\boxed{G_3}$
  $L_H \leftarrow |H|/n$;  $L \leftarrow |C|/n$;
  $p \leftarrow \text{LLCP}_n(\mathcal{Q}_{|H,M}, (H, M))$;
  $U \leftarrow \varphi(K, 0^n)$;
  for $i = 1, \ldots, L_H$ do
    $\tau \leftarrow E_U(H_i)$;
    $U \leftarrow \varphi(K, H_i \oplus \tau)$;
    if $(U \in B$ and $i > p)$ then
      bad $\leftarrow$ true;  $\boxed{U \xleftarrow{\$} \{0,1\}^n \setminus B;}$
    $B \leftarrow B \cup U$;
  for $i = 1, \ldots, L - 1$ do
    $M_i \leftarrow E_U^{-1}(C_i)$;
    $U \leftarrow \varphi(K, C_i \oplus M_i)$;
    if $(U \in B$ and $i + L_H > p)$ then
      bad $\leftarrow$ true;  $\boxed{U \xleftarrow{\$} \{0,1\}^n \setminus B;}$
    $B \leftarrow B \cup U$;
  $M_L \leftarrow E_U^{-1}(C_L)$;
  $U \leftarrow \varphi(K, C_L \oplus M_L)$;
  if $(U \in B$ and $H \notin A)$ then
    bad $\leftarrow$ true;  $\boxed{U \xleftarrow{\$} \{0,1\}^n \setminus B;}$
  if $(T = E_U(\tau)$ and $(H, C, T) \notin \mathcal{Q}_{|H,C,T})$ then
    win $\leftarrow$ true;
  $\mathcal{Q} \leftarrow (H, \perp, C, \perp)$;
  $B \leftarrow B \cup U$;
  return $(T = E_U(\tau))$;
```

Fig. 4. Games G_1-G_3 for the proof of Lemma 1. Game G_3 contains the code in the box while G_2 does not.

Proof (Lemma 1). Our bound is derived by game playing arguments. Consider games G_1-G_3 of Figure 4 and a fixed adversary A asking at most q queries with a total length of at most ℓ blocks.

The functions **Initialize** and **Finalize** are identical for all games in this proof. Lets denote G_0 as the Game INT-CTXT(NI) as defined in Figure 3. Definition 5 states that

$$\mathbf{Adv}_{\Pi}^{\text{INT-CTXT(NI)}}(A) \leq \Pr[A^{G_0} \Rightarrow 1].$$

In G_1, the encryption and verify placeholders are replaced by their specific McOE-X counterparts as of Definition 6. Clearly, $\Pr[A^{G_0} \Rightarrow 1] = \Pr[A^{G_1} \Rightarrow 1]$. We now discuss the differences between G_1 and G_2. The set B is initialized to $\{\varphi(K, 0^n)\}$ and then collects new key-input values U which are computed during the encryption or verification process (in lines 204, 207, 213, 223, 226, 232 and 237). We note that, since φ is injective, a collision for the chaining values follows if there is a collision in the U values.

In lines 203 and 222, the LLCP_n oracle is inquired. Finally, the variable `bad` is set to `true` if one of the if-conditions in lines 208, 214, 227, 233, or 238 is `true`. *None* of these modifications affect the values returned to the adversary and therefore

$$\Pr[A^{G_1} \Rightarrow 1] = \Pr[A^{G_2} \Rightarrow 1].$$

For our further discussion we require another game G_4 which is explained in more detail later in this proof[3]. It follows that

$$\begin{aligned}\Pr[A^{G_2} \Rightarrow 1] &= \Pr[A^{G_3} \Rightarrow 1] + |\Pr[A^{G_2} \Rightarrow 1] - \Pr[A^{G_3} \Rightarrow 1| \\ &\leq \Pr[A^{G_3} \Rightarrow 1] + \Pr[A^{G_3} \mathit{sets}\ \texttt{bad}] \\ &\leq \Pr[A^{G_4} \Rightarrow 1] + |\Pr[A^{G_3} \Rightarrow 1] - \Pr[A^{G_4} \Rightarrow 1]| + \Pr[A^{G_3} \mathit{sets}\ \texttt{bad}]. \qquad (3)\end{aligned}$$

We now proceed to upper bound any of the three terms contained in (3) – in right to left order. The success probability of game G_3 does not differ from the success probability of G_2 unless a chaining value U occurs twice. In this case, the adversary must (i) either have 'found' a collision for $E_{\varphi(K,X)}(Y) \oplus Y$, *i.e.* she stumbles over (X, Y) and (X', Y') such that $E_{\varphi(K,X)}(Y) \oplus Y = E_{\varphi(K,X')}(Y') \oplus Y'$ or, (ii), must have found a preimage of $\varphi(K, 0^n)$, which is always the starting point of our chain. Note that that value $\varphi(K, 0^n)$ is initially stored in the set B. In both cases, the variable `bad` would have been set to `true`, and it follows [8] that

$$\Pr[A^{G_3} \mathit{sets}\ \texttt{bad}] \leq \frac{(q+\ell)(q+\ell+1)}{2^n - (q+\ell)} + \frac{q+\ell}{2^n - (q+\ell)}.$$

We now describe the new game G_4. It is equal to G_3 *except* that the block cipher E and its inverse E^{-1} are replaced by randomly chosen functions **EncryptBlock** and **DecryptBlock**, which are modeled as pseudo random permutations. We assume that they are implemented via lazy sampling. More precisely, the call $E_K(A)$ is replaced by an invocation of **EncryptBlock**$_K(A)$ and the call $E_K^{-1}(A)$ is replaced by an invocation of **DecryptBlock**$_K(A)$. We now upper bound the difference between G_3 and G_4.

So, by definition of G_4, we have

$$|\Pr[A^{G_3} \Rightarrow 1] - \Pr[A^{G_4} \Rightarrow 1]| \leq \mathbf{Adv}_{E,E^{-1}}^{\text{RK-CCA-PRP}}(q+\ell).$$

Finally, we have to upper bound the advantage for the adversary A to win the game G_4. A can only win this game if the condition in line 238 (resp. 438 for game G_4) is `true`. As usual, we assume *wlog.* that A doesn't ask a question if the answer is already known which implies that $(H, C, T) \notin \mathcal{Q}_{|H,C,T}$. For our analysis we distinguish between three cases. So we formally adjust line 240 (*i.e.* choose as the tag computation operation either E or E^{-1}) such that we always have enough randomness left for our result.

[3] Since the difference is very minor, we do not provide an extra figure.

Case 1: *H has already been used in an Encrypt or Verify query before and $U \in B$.* Since we already have computed τ in the past, the chance of success is upper bounded by the probability $\Pr[E_U^{-1}(T) = \tau]$ which can be upper bounded by $1/(2^n - (q + \ell))$.

Case 2: *H has never been used before, also U has never been used as a chaining value.* Then the tagging operation uses a 'new key' – essentially due since φ is injective – and therefore the output of $E_U(\tau)$ is uniformly distributed and the success probability is $\leq 1/2^n$.

Case 3: *$H \in A$ but U has never been used as a chaining value.* The chance of success is upper bounded by $\Pr[E_U^{-1}(T) = \tau]$ which can be upper bounded by $1/2^n$.

Note that the 'missing' fourth case has been explicitly excluded by line 240 (resp. 440). Since these three cases are mutually exclusive, we can upper bound the success probability for q queries as

$$\Pr[A^{G_4} \Rightarrow 1]| \leq \frac{q}{2^n - (q + \ell)}.$$

Our claim follows by adding up the individual bounds. □

Lemma 2. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a* McOE-X *scheme as in Definition 6 (i). Let q be the number of total queries an adversary A is allowed to ask and ℓ be an integer representing the total length of the queries to $\mathcal{E}$ and $\mathcal{D}$. Then,*

$$\mathbf{Adv}_{\Pi}^{\mathrm{CPA(AOE,NI)}}(q, \ell, t) \leq \ 2 \left(\frac{(q+\ell)(q+\ell+1)}{2^n - (q+\ell)} + \frac{q+\ell}{2^n - (q+\ell)} + \mathbf{Adv}_{E}^{\text{RK-CPA-PRP}}(q+\ell) \right).$$

The proof is given in Appendix C.

6 Discussion

New Challenges for Research. At the this point of time, cryptographic research has developed an inpressive number of good schemes for encryption, authentication, and authenticated encryption. Many of these schemes have been proven secure under standard assumptions on the underlying primitives. In practice, however, such schemes are often used in a way that undermines security. Trying to design cryptosystems as "misuse resistant" as possible still stands as a challenge for cryptographers.

Furthermore, our research seems to pose new challenges for the design of symmetric primitives. Ideally, we would like to implement McOE using a tweakable n-bit block cipher with n-bit tweaks, supporting fast random tweak changes. Due to the current lack of such a primitive, we designed McOE-X, which requires an ordindary n-bit block cipher being secure against XOR-related key attacks, and supporting fast random key changes. Much beyond McOE, cryptosystem designers could benefit from new tweak-agile tweakable block ciphers and new key-agile ordinary block ciphers.

It is mentionable that McOE-X, when using Threefish-512 in software, performs considerably better as when using software or even hardware AES-128. (Note that Threefish-512 actually is a tweakable block cipher, but the 128-bit tweak is too short for McOE.) As an alternative, we developed further variants of McOE using double encryption and Galois field arithmetic. These two variants also don't expose the underlying block cipher to related-key attacks.

Conclusion. Originally, this research has been inspired by the search for a default authenticated encryption mode of operation for a general-purpose cryptographic library. It should offer, by default, a huge failure tolerance for practical software developers and still allow being used in an on-line manner.

Since the well-known schemes as, such as OCB and SIV, did not fit our requirements, we searched for other ways to achieve the security and functionality we were looking for. Apart from McOE, generic composition (Encrypt-then-Mac) of a secure on-line cipher for encryption and a secure deterministic MAC for authentication, using two independent keys might be another solution. As it turned out, using McOE, one can save the additional key and the time to generate the MAC by using a slightly tweaked on-line cipher for both encryption and authentication.

Acknowledgments

We like to thank Jakob Wenzel for very helpful comments, Phil Rogaway for making us aware of the Galois field native instructions, and the participants of the Dagstuhl Seminar on Symmetric Cryptography 2012 for inspiring discussions.

References

1. Mihir Bellare, Alexandra Boldyreva, Lars R. Knudsen, and Chanathip Namprempre. Online Ciphers and the Hash-CBC Construction. In *CRYPTO*, pages 292–309, 2001.
2. Mihir Bellare, Alexandra Boldyreva, Lars R. Knudsen, and Chanathip Namprempre. On-Line Ciphers and the Hash-CBC Constructions. *IACR Cryptology ePrint Archive*, 2007:197, 2007.
3. Mihir Bellare, Alexandra Boldyreva, Lars R. Knudsen, and Chanathip Namprempre. Online Ciphers and the Hash-CBC Construction. Cryptology ePrint Archive, Report 2007/197; full version of [1], 2007. `http://eprint.iacr.org/`.
4. Mihir Bellare and Chanathip Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. *J. Cryptology*, 21(4):469–491, 2008.
5. Mihir Bellare and Phillip Rogaway. Encode-Then-Encipher Encryption: How to Exploit Nonces or Redundancy in Plaintexts for Efficient Cryptography. In *ASIACRYPT*, pages 317–330, 2000.
6. Mihir Bellare, Phillip Rogaway, and David Wagner. The EAX Mode of Operation. In *FSE*, pages 389–407, 2004.
7. John Black, Martin Cochran, and Thomas Shrimpton. On the Impossibility of Highly-Efficient Blockcipher-Based Hash Functions. In *EUROCRYPT*, pages 526–541, 2005.
8. John Black, Phillip Rogaway, and Thomas Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In *CRYPTO*, pages 320–335, 2002.
9. Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting Mobile Communications: The Insecurity of 802.11. In *MOBICOM*, pages 180–189, 2001.
10. Enrico Buonanno, Jonathan Katz, and Moti Yung. Incremental Unforgeable Encryption. In *FSE*, pages 109–124, 2001.
11. Intel Corporation. AES-NI Sample Library v1.2. `http://software.intel.com/en-us/articles/download-the-intel-aesni-sample-library/`, 2010.
12. Joan Daemen. *Hash Function and Cipher Design: Strategies Based on Linear and Differential Cryptanalysis.* Ph.D. thesis, Katholieke Universiteit Leuven, Leuven, Belgium, March 1995.
13. Morris Dworkin. *Special Publication 800-38C: Recommendation for block cipher modes of operation: the CCM mode for authentication and confidentiality.* National Institute of Standards and Technology, U.S. Department of Commerce, May 2005.
14. Ewan Fleischmann, Christian Forler, and Stefan Lucks. McOE: A Foolproof On-Line Authenticated Encryption Scheme. *IACR Cryptology ePrint Archive*, 2011:644, 2011.
15. Pierre-Alain Fouque, Gwenaëlle Martinet, Frédéric Valette, and Sébastien Zimmer. On the Security of the CCM Encryption Mode and of a Slight Variant. In *ACNS*, pages 411–428, 2008.
16. Brian Gladman. Brian Gladman's AES Implementation, 19th June 2006. `http://gladman.plushost.co.uk/oldsite/AES/index.php`.
17. Virgil D. Gligor and Pompiliu Donescu. Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes. In *FSE*, pages 92–108, 2001.
18. Shafi Goldwasser and Silvio Micali. Probabilistic Encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
19. Shay Gueron and Michael E. Kounavis. Efficient implementation of the Galois Counter Mode using a carry-less multiplier and a fast reduction algorithm, journal = Inf. Process. Lett. 110(14-15):549–553, 2010.
20. George Hotz. Console Hacking 2010 - PS3 Epic Fail. 27th Chaos Communications Congress, 2010. `http://events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf`.
21. ISO/IEC. *19772:2009, Information technology – Security techniques – Authenticated Encryption*, 2009.

22. Tetsu Iwata. New Blockcipher Modes of Operation with Beyond the Birthday Bound Security. In *FSE*, pages 310–327, 2006.
23. Tetsu Iwata. Authenticated Encryption Mode for Beyond the Birthday Bound Security. In *AFRICACRYPT*, pages 125–142, 2008.
24. Tetsu Iwata and Kan Yasuda. BTM: A Single-Key, Inverse-Cipher-Free Mode for Deterministic Authenticated Encryption. In *Selected Areas in Cryptography*, pages 313–330, 2009.
25. Tetsu Iwata and Kan Yasuda. HBS: A Single-Key Mode of Operation for Deterministic Authenticated Encryption. In *FSE*, pages 394–415, 2009.
26. Charanjit S. Jutla. Encryption Modes with Almost Free Message Integrity. *J. Cryptology*, 21(4):547–578, 2008.
27. Jonathan Katz and Moti Yung. Unforgeable Encryption and Chosen Ciphertext Secure Modes of Operation. In *FSE*, pages 284–299, 2000.
28. Tadayoshi Kohno. Attacking and Repairing the WinZip Encryption Scheme. In *ACM Conference on Computer and Communications Security*, pages 72–81, 2004.
29. Tadayoshi Kohno, John Viega, and Doug Whiting. CWC: A High-Performance Conventional Authenticated Encryption Mode. In *FSE*, pages 408–426, 2004.
30. Ted Krovetz and Phillip Rogaway. New Blockcipher Modes of Operation with Beyond the Birthday Bound Security. In *FSE*, pages 310–327, 2006.
31. Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable Block Ciphers. In *CRYPTO*, pages 31–46, 2002.
32. Stefan Lucks. Ciphers secure against related-key attacks. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 359–370. Springer, 2004.
33. Stefan Lucks. Two-Pass Authenticated Encryption Faster Than Generic Composition. In *FSE*, pages 284–298, 2005.
34. David A. McGrew and John Viega. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In *In INDOCRYPT, volume 3348 of LNCS*, pages 343–355. Springer, 2004.
35. Niels Ferguson and Stefan Lucks and Bruce Schneier and Doug Whiting and Mihir Bellare and Tadayoshi Kohno and Jon Callas and Jesse Walker. Skein source code and test vectors. `http://www.skein-hash.info/downloads`.
36. Kenneth G. Paterson and Gaven J. Watson. Plaintext-dependent decryption: A formal security treatment of ssh-ctr. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 345–361. Springer, 2010.
37. Phillip Rogaway. Authenticated-Encryption with Associated-Data. In *ACM Conference on Computer and Communications Security*, pages 98–107, 2002.
38. Phillip Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In *ASIACRYPT*, pages 16–31, 2004.
39. Phillip Rogaway. Nonce-Based Symmetric Encryption. In *FSE*, pages 348–359, 2004.
40. Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In *ACM Conference on Computer and Communications Security*, pages 196–205, 2001.
41. Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In *EUROCRYPT*, pages 373–390, 2006.
42. Phillip Rogaway and Thomas Shrimpton. Deterministic Authenticated-Encryption: A Provable-Security Treatment of the Key-Wrap Problem. Cryptology ePrint Archive, Report 2006/221; full version of [41], 2006. `http://eprint.iacr.org/`.
43. Phillip Rogaway and Haibin Zhang. Online Ciphers from Tweakable Blockciphers. In *CT-RSA*, pages 237–249, 2011.
44. Todd Sabin. Vulnerability in Windows NT's SYSKEY encryption. *BindView Security Advisory*, 1999. Available at `http://marc.info/?l=ntbugtraq&m=94537191024690&w=4`.
45. Hongjun Wu. The Misuse of RC4 in Microsoft Word and Excel. Cryptology ePrint Archive, Report 2005/007, 2005. `http://eprint.iacr.org/`.

A Misuse-Attacks: The weak point of current Authenticated Encryption (AE) Schemes.

A.1 Attacking Schemes without Claimed Resistance Against Nonce Reuse

Cipher-block-chaining (CBC) is an unauthenticated encryption mode which is sometimes used as the encryption component of an AE scheme. But one can easily distinguish CBC encryption from a good on-line cipher, if the nonce (or the IV) is constant. The attack from [1] only needs three chosen plaintexts. Counter mode, which has been very popular among the designers of AE schemes, fails terribly in nonce reuse settings, since it generates exactly the same keystream twice when the nonce is reused. It was to be expected that a scheme using counter mode or CBC inherits the nonce reuse issue from that mode. But, as it turned out, common AE schemes also fail at the authenticity frontier, see Table 2 for an overview. This is an unpleasant surprise, since the cryptographic community has known well deterministic MACs for a long time – so why is the authenticity provided by most authenticated encryption schemes so much more fragile than the authenticity provided by well-known MACs?

The following two attack patterns will be used in most of our attacks.

Repeated Keystream. Many AE schemes generate a keystream $S = F_K(V)$ of length $|M|$, depending on the secret key K and the nonce V, and encrypt the message M by computing the ciphertext $C = S \oplus M$, typically by applying a block cipher in counter mode. If the same nonce is used more than once, the following attack straightforwardly breaks the privacy:

- Encrypt a plaintext M under the nonce V to a ciphertext C with tag T.
- Encrypt a plaintext $M' \neq M$ under the same V to a ciphertext C' and a tag T'.
- It turns out that $C' = C \oplus M \oplus M'$ holds.

Linear Tag. Many AE schemes, which generate a keystream $S = F_K(V)$ as above, apply the encrypt-then-authenticate paradigm and allow to rewrite the authentication tag T as

$$T = f(V) \oplus g(C),$$

where V is the nonce, C is the ciphertext, and f and g are some key-dependent functions. This enables the adversary to mount the following attack:

- Encrypt the plaintext M under the nonce V to (C, T) with $T = f(V) \oplus g(C)$.
- Encrypt the plaintext $M' \neq M$ with $|M'| = |M|$ under the nonce $V' \neq V$ to (C', T') with the tag $T' = f(V') \oplus g(C')$.
- Set $M'' := M' \oplus C' \oplus C$. Encrypt M'' under the nonce V' to (C'', T''). Observe $C'' = C$, thus $T'' = f(V') \oplus g(C)$.
- Set $T^* = T \oplus T' \oplus T'' = f(V) \oplus g(C')$, The adversary accepts (C', T^*) under V.

Two-Pass AE(AD) Modes: CWC [29], GCM [34], CCM [13], EAX [6], CHM [22] . All the common two-pass AE(AD) modes, CHM,CWC, GCM, CCM and EAX, use the counter mode as the underlying encryption operation and are thus vulnerable to the repeated keystream attack pattern. Four of them, CHM, CWC, GCM, and EAX, are designed according to the encrypt-then-authenticate paradigm, and are thus vulnerable to the linear tag attack pattern. The designers of CCM followed authenticate-then-encrypt, which seems to defend against the linear tag pattern. Forgery attacks against CCM have been presented in [15], though.

Mixed AE(AD) Modes: RPC [10] and CCFB [33]. RPC combines counter mode and electronic codebook mode. Given an n-bit block cipher E under a key K and a c-bit counter cnt, RPC takes an $(n-c)$-bit plaintext block M_i and computes the ciphertext block $C_i := E_K(M_i || (\text{cnt} + i) \bmod 2^c)$. Authentication is performed locally for each ciphertext block: During decryption, RPC computes $(M_i || X_i) = E_K^{-1}(C_i)$ and accepts M_i as authentic if and only if $X_i = (\text{cnt} + i) \bmod 2^c$. The nonce defines cnt.

Under nonce reuse, the same sequence $(\text{cnt} + i) \bmod 2^c$ of counter values is used for different messages. This makes it easy to attack the privacy – essentially, when encrypting messages of m $(n - c)$-bit blocks, RPC degrades into m independent electronic codebooks. Also, given two authentic ciphertexts, $(C_1^0, \dots, C_L^0)$ and $(C_1^1, \dots, C_L^1)$, any ciphertext $(C_1^{\sigma(1)}, \dots, C_L^{\sigma(L)})$ with $\sigma(i) \in \{0, 1\}$ is valid, since authenticity is verified locally for each $C_i^{\sigma(i)}$.

Similarly to RPC, CCFB is a combination of Counter and CFB mode. Unlike RPC, CCFB generates a single "global" authentication tag. Variants of the repeated keystream pattern applies and the linear tag pattern apply to CCFB.

One-Pass AE(AD) Modes: IAPM [26], OCB1[40], OCB2[38], OCB3[30], TAE [31]. Given a nonce V and a secret key K, IAPM [26] encrypts a plaintext $(M_1, \dots, M_m)$ to a ciphertext $(C_1, \dots C_m)$ and an authentication tag T as follows.

Initial step: Generate $m + 2$ values $s_0, s_1, \dots s_{m+1}$ depending on V and K, but not on the plaintext $(M_1, \dots, M_m)$.
Encryption: $x_0 := V$; For $i \in \{1, \dots, m\}$: $C_i := E_K(M_i \oplus s_i) \oplus s_i$.
Authentication tag: $T := E_K(s_{m+1} \oplus \sum_{1 \le i \le m} M_i) \oplus s_0$.

Similarly to RPC, IAPM behaves like a set of m independent electronic codebooks and is vulnerable to the same distinguishing attack. A forgery can exploit the fact that two different same-length messages $(M_1, \dots, M_m)$ and $(M'_1, \dots, M'_m)$, encrypted under the same nonce, have the same authentication tag $T = E_K(s_{m+1} \oplus \sum_{1 \le i \le m} M_i) \oplus s_0 = E_K(s_{m+1} \oplus \sum_{1 \le i \le m} M'_i) \oplus s_0$ if $\sum_{1 \le i \le m} M_i = \sum_{1 \le i \le m} M'_i$.

As much as our attacks are concerned, OCB1–3 and TAE are quite similar to IAPM, and the attacks are the same.

More One-Pass Modes: IACBC [26] and XCBC [17]. Given a nonce V and a secret key K, IACBC [26] encrypts $(M_1, \dots, M_m)$ to $(C_1, \dots C_m)$ and an authentication tag T as follows.

Initial step: Generate $m+1$ values $s_0, s_1, \dots s_m$ depending on V and K, but not on the plaintext $(M_1, \dots, M_m)$.
Encryption: $x_0 := V$; For $i \in \{1, \dots, m\}$: $x_i := E_K(M_i \oplus x_{i-1})$, $C_i := x_i \oplus s_i$.
Authentication tag: $T := E_K(x_m \oplus \sum_{1 \le i \le m} M_i) \oplus s_0$.

The following nonce-reuse attack distinguishes IACBC encryption from an online permutation and also provides an existential forgery. For simplicity, we only consider 1-block messages $V \neq W$, which we also use as nonces: Encrypt W under V to (C_1, T). Encrypt V under W to (C'_1, T'). Encrypt V under V to (C''_1, T''). Set $C'''_1 := C_1 \oplus C'_1 \oplus C''_1$ and $T''' := T \oplus T' \oplus T''$. (C'''_1, T) is a valid encryption of W under W.

Given a nonce V and secret keys K and K', XCBC encrypts a plaintext $(M_1, \dots, M_m)$ to a ciphertext $(C_1, \dots C_m)$ and an authentication tag T as follows.

Initial step: Generate $m+1$ values $s_1, \dots s_{m+1}$ depending on V and K, but not on the plaintext $(M_1, \dots, M_m)$.
Encryption:
1. $C_0 := E_K(V)$; $x_0 := E_{K'}(V)$;
2. Generate an additional message word $M_{m+1} := x_0 \oplus M_1 \oplus \dots \oplus M_m$ for authentication.
3. For $i \in \{1, \dots, m+1\}$: $x_i := E_K(M_i \oplus x_{i-1})$, $C_i := (x_i + s_i) \bmod 2^n$.

The best attack we have found for XCBC is not quite as damaging as the attacks on the other schemes, as the attack workload is at $O(2^{n/4})$, and the attack only provides a distinguisher, not a forger. For this reuse-nonce chosen-plaintext attack, we ignore the authentication tag: Generate $2^{n/4}$ encryptions of messages M_1^i under a nonce V to C_1^i. Statistically, expect one pair $i \neq j$ such that the least significant $n/2$ bits of C_1^i are identical to the least significant $n/2$ bits of C_1^j. Generate $2^{n/4}$ encryptions of messages (M_1^i, M_2^k) and (M_1^j, M_2^ℓ) under V to (C_1^i, C_2^k) and (C_1^j, C_2^ℓ), where the least significant $n/2$ bits of M_2^k and M_2^ℓ are the same. (Statistically, expect one pair $k \neq \ell$ such that $C_2^k = C_2^\ell$ holds.) Choose an arbitrary M_3. Encrypt (M_1^i, M_2^k, M_3) and (M_1^j, M_2^ℓ, M_3) under V to $(C_1^i, C_2^k, C_3^{i,k})$ and $(C_1^j, C_2^\ell, C_3 j, \ell)$. Observe $C_3^{i,k} = C_3^{j,\ell}$.

A.2 Offline Schemes, Defeating Nonce-Reuse (SIV [41], HBS [25], BTM [24])

Given a nonce N, a message M and associated Data H, these schemes perform two steps:

1. Generate the authentication tag T from H, M, and N.
2. Encrypt M in counter mode, using T as the nonce.

This is inherently offline, because one must finish step 1 before one can start step 2. All of SIV, HBS, and BTM perform counter mode encryption, but employ different MACs schemes to generate the tag T.

This usage of the counter mode is vulnerable in an *online decryption misuse* case, where, during decryption, a would-be plaintext is compromised before the tag has been verified. A chosen-ciphertext adversary can exploit that to determine an unknown keystream and then to decrypt an unknown message.

Another misuse case may apply when nonce-reuse is possible and the sender reads the message twice, once for each of the two steps – if there is any chance that *the message has been modified* between the two read operations.

Note that both misuse cases become quite harmless if one replaces the counter mode encryption by the application of an online permutation.

B Proof of Theorem 1

Consider games G_0, G_1, G_2 of Figure 5. For a fixed CCA3(ω,ν) adversary A on the scheme Π it holds that

$$\begin{aligned}\Pr[A_{\Pi}^{\mathrm{CCA3}(\omega,\nu)} \Rightarrow 1] &= \Pr[A^{G_0} \Rightarrow 1]\\ &= \Pr[A^{G_1} \Rightarrow 1] + (\Pr[A^{G_0} \Rightarrow 1] - \Pr[A^{G_1} \Rightarrow \texttt{true}])\\ &\leq \Pr[A^{G_1} \Rightarrow 1] + \Pr[A^{G_1} \textit{sets}\ \texttt{bad}].\end{aligned}$$

Since the `Decrypt` oracles of G_1 and G_2 always return $\perp$,

$$\Pr[A^{G_1} \Rightarrow 1] = \Pr[A^{G_2} \Rightarrow 1].$$

Now, we design two adversaries A_c and A_p so that

$$\begin{aligned}&\Pr[A^{G_1} \textit{sets}\ \texttt{bad}] \leq \Pr[A_c{}_{\Pi}^{\text{INT-CTXT}(\omega,\nu)} \Rightarrow 1] \text{ and}\\ &\Pr[A^{G_2} \Rightarrow 1] \leq \Pr[A_p{}_{\Pi}^{\text{CPA}(\omega,\nu)} \Rightarrow 1].\end{aligned}$$

A_p: Adversary A_p simply runs A answering A's **Encrypt** queries using its own **Encrypt** oracle, and answers **Decrypt** queries with $\perp$. A_p outputs whatever A outputs.

A_c: Adversary A_c runs A answering A's **Encrypt** queries using its own **Encrypt** oracle. It submits A's **Decrypt** queries to it's **Verify** oracle (*cf.* Figure 3) and, regardless of the response, returns $\perp$. Note that the **Verify** oracle sets `win` to **true** if and only if a fresh **Decrypt** query is valid. Just such a query would set the variable `bad` to **true**. □

```
1 Initialize($\omega, \nu$)
2   b $\xleftarrow{\$}$ {0,1};
3   if(b=1) then
4     K ← $\mathcal{K}()$;

5 Finalize(d)
6   return (d=b);
```

```
7 Encrypt(H,M)
8   if ($\nu = $ NR and $V \in B$) then
9     return ⊥;
10  else
11    $B \leftarrow B \cup \{V\}$;
12  if(b=1) then
13    C ← $\mathcal{E}_K$(H,M);
14  else
15    C ← $\$^{\omega}$(H,M);
16  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(H,C)\}$;
17  return C;
```

```
100 Decrypt(H,C)          Game $G_0$, $\boxed{G_1}$
101   M ← ⊥;
102   if ($(H,C) \notin \mathcal{Q}$ and b=1) then
103     M ← $\mathcal{D}_K$(H,C);
104   if(M≠ ⊥) then
105     bad ← true; $\boxed{M \leftarrow \perp;}$
106   return M;

200 Decrypt(H,C)          Game $G_2$
201   return ⊥;
```

Fig. 5. Games G_0, G_1 and G_2 for the proof of Theorem 1. Game G_1 contains the code in the box while G_0 does not. H_0 denotes the first block of the header representing the nonce/initial value.

C Proof of Lemma 2

Proof (of Lemma 2). Our bound is derived by game playing arguments. Consider games G_1 and G_2 of Figure 6. The functions **Initialize** and **Finalize** are identical for any of those games.

At first we investigate the differences between the $CPA(\text{AOE}, \text{NI})$ game from Figure 2 and G_1 from Figure 6. In G_1 we have replaced $\mathcal{E}$ by its definition of McOE-X, and $\$^w$ by an on-line encryption oracle **OnlinePermutation** (line 102) that just models a 'perfect' OPRP, *i.e.* for two plaintexts with an equal prefix it returns two ciphertexts that also share a prefix of the same length. We again assume this oracle to be implemented by lazy sampling. Then, set B collects all chaining values (lines 113 and 119) in order to intercept the occurrence of two equal chaining values which do lead – due to the injectivity of φ – to two equal keys for the encryption of a block.

In line 105, the oracle LLCP$_n$ is invoked returning the length of the longest common prefix of (H, M) and $\mathcal{Q}_{|H,M}$. Finally, the variable `bad` is set to `true` if (one of) the conditions of lines 111/211 or 117/217 holds. These changes do not affect the success probability of an adversary, because the output of the oracle remains unchanged. More precisely, the distribution of the output does not change. This means that – using a new game G_3 described shortly –

$$\mathbf{Adv}_{\Pi}^{\text{CPA(AOE, NI)}}(A) = 2 \cdot |\Pr[A^{G_1} \Rightarrow 1] - 0.5|.$$

Therefore, by common game playing arguments,

$$\begin{aligned}\Pr[A^{G_1} \Rightarrow 1] &= \Pr[A^{G_2} \Rightarrow 1] + |\Pr[A^{G_1} \Rightarrow 1] - \Pr[A^{G_2} \Rightarrow 1| \\ &\leq \Pr[A^{G_2} \Rightarrow 1] + \Pr[A^{G_2} sets\ \texttt{bad}] \\ &\leq \Pr[A^{G_3} \Rightarrow 1] + |\Pr[A^{G_2} \Rightarrow 1] - \Pr[A^{G_3} \Rightarrow 1]| + \Pr[A^{G_2} sets\ \texttt{bad}].\end{aligned}$$

The success probability of game G_2 does not differ from the success probability of G_1 unless a chaining value U occurs twice. In this case, the adversary must either have found a collision for $E_{\varphi(K,X)}(Y) \oplus Y$, *i.e.* she has found (X, Y) and (X', Y') such that $E_{\varphi(K,X)}(Y) \oplus Y = E_{\varphi(K,X')}(Y') \oplus Y'$ or must have found a preimage of $\varphi(K, 0^n)$. In both cases, the variable `bad` would have been set to `true`, and it follows again by [8] that

$$\Pr[A^{G_2} sets\ \texttt{bad}] \leq \frac{(q+\ell)(q+\ell+1)}{2^n - (q+\ell)} + \frac{q+\ell}{2^n - (q+\ell)}$$

The aforementioned new game G_3 is equal to the game G_2 *except* that the block cipher E and its inverse E^{-1} are replaced by randomly chosen functions **EncryptBlock** and **DecryptBlock**, which are modeled as a pseudo random permutations. We assume that they are implemented

```
Initialize()
  $b \xleftarrow{\$} \{0,1\}$; $K \xleftarrow{\$} \mathcal{K}()$; $B \leftarrow \{\varphi(K, 0^n)\}$;

Finalize(d)
  return (b=d);

Encrypt$(H, M)$ Game $G_1$, $\boxed{G_2}$
  if $(b = 0)$ then
    $C \leftarrow$ OnlinePermutation$(H, M)$;
  else
    $L_H \leftarrow |H|/n$; $L \leftarrow |M|/n$;
    $p \leftarrow \mathrm{LLCP}_n(\mathcal{Q}, (H, M))$;
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup (H, M)$;
    $U \leftarrow \varphi(K, 0^n)$;
    for $i = 1, \ldots, L_H$ do
      $\tau \leftarrow E_U(H_i)$;
      $U \leftarrow \varphi(K, H_i \oplus \tau)$;
      if $(U \in B$ and $i > p)$ then
        bad $\leftarrow$ true; $\boxed{U \xleftarrow{\$} \{0,1\}^n \setminus B;}$
      $B \leftarrow B \cup U$;
    for $i = 1, \ldots, L$ do
      $C_i \leftarrow E_U(M_i)$;
      $U \leftarrow \varphi(K, C_i \oplus M_i)$;
      if $(U \in B$ and $i + L_H > p)$ then
        bad $\leftarrow$ true; $\boxed{U \xleftarrow{\$} \{0,1\}^n \setminus B;}$
      $B \leftarrow B \cup U$;
  return $C$;
```

Fig. 6. Games G_1 and G_2 for the proof of Lemma 2. Game G_2 contains the code in the box while G_1 does not.

via lazy sampling. More precisely, the call $E_K(A)$ is replaced by an invocation of **Encrypt-Block**$_K(A)$ and the call $E_K^{-1}(A)$ is replaced by an invocation of **DecryptBlock**$_K(A)$. We now upper bound the difference between G_2 and G_3. So, by definition of G_4, we have

$$|\Pr[A^{G_2} \Rightarrow 1] - \Pr[A^{G_3} \Rightarrow 1]| \leq \mathbf{Adv}_E^{\text{RK-CPA-PRP}}(q + \ell).$$

Finally, we have to upper bound the advantage for an adversary A to win the game G_3. Since the U cannot collide and it is not possible to compute a preimage for any query, the algorithm for $b = 0$ *is* an OPRP, and therefore the success probability to win G_3 for any adversary is 0.5, *i.e.* she has no advantage in winning this game.

Our claim follows by adding up the individual bounds. □

Cycling Attacks on GCM, GHASH and Other Polynomial MACs and Hashes

Markku-Juhani O. Saarinen

REVERE SECURITY
4500 Westgrove Drive, Suite 335, Addison, TX 75001, USA.
mjos@reveresecurity.com

Abstract. The Galois/Counter Mode (GCM) of operation has been standardized by NIST to provide single-pass authenticated encryption. The GHASH authentication component of GCM belongs to a class of Wegman-Carter polynomial hashes that operate in the field $\mathrm{GF}(2^{128})$. We present message forgery attacks that are made possible by its extremely smooth-order multiplicative group which splits into 512 subgroups. GCM uses the same block cipher key K to both encrypt data and to derive the generator H of the authentication polynomial for GHASH. In present literature, only the trivial weak key $H = 0$ has been considered. We show that GHASH has much wider classes of weak keys in its 512 multiplicative subgroups, analyze some of their properties, and give experimental results on AES-GCM weak key search. Our attacks can be used not only to bypass message authentication with garbage but also to target specific plaintext bits if a polynomial MAC is used in conjunction with a stream cipher. These attacks can also be applied with varying efficiency to other polynomial hashes and MACs, depending on their field properties. Our findings show that especially the use of short polynomial-evaluation MACs should be avoided if the underlying field has a smooth multiplicative order.

Keywords: Cryptanalysis, Galois/Counter Mode, AES-GCM, Cycling Attacks, Weak Keys.

1 Introduction

Authenticated encryption modes and algorithms provide confidentiality and integrity protection in a single processing step. This results in performance and cost advantages as data paths can be shared.

The Galois/Counter Mode (GCM) has been standardized by NIST [14] to be used in conjunction with a 128-bit block cipher for providing authenticated encryption functionality. When paired with the AES [13] algorithm, the resulting AES-GCM combination has been used as a replacement to dedicated hash-based HMAC [1] in popular cryptographic protocols such as SSH [9], IPSec [11] and TLS [16].

In AES-GCM, data is encrypted using the Counter Mode (CTR). A single AES key K is used to both encrypt data and to derive authentication secrets. The component that is used by GCM to produce a message authentication code is called GHASH. GCM also supports Additional Authenticated Data (AAD) which is authenticated using GHASH but transmitted as plaintext.

The GHASH algorithm belongs to a widely studied class of Wegman-Carter [19, 20] polynomial MACs. These were originally proposed in context of polynomial evaluation independently by three authors [6, 18, 5]. A good overview of their genealogy and evolution is by Bernstein [3, 2]. The security bounds known for these algorithms indicate that a n-bit tag will give $2^{-\frac{n}{2}}$ security against forgery [3, 17].

In this paper we give further evidence that this is not only the security lower bound but an upper bound as well. It can be argued that universal hashes sacrifice communication bandwidth for convenience as traditional hash-based MACs are designed to reach the information theoretic 2^{-n} bound against message forgery and are therefore technically somewhat inferior, especially for short MACs. The security against cycling attacks depends very sharply on the properties of the underlying field.

This paper is structured as follows. We give a description of GHASH in Section 2, followed by a key observation regarding collisions derived from cycles in Section 3. Section 4 contains an analysis of cycle lengths and group orders. In Section 5 we discuss the probability of successful forgery. Section 6 briefly considers targeted attacks against underlying protocols. Section 7 contains a test and experimental results related to cycle lengths. We discuss the security of other polynomial mac constructions in Section 8 and conclude in Section 9.

2 Description of GHASH

Let X be a concatenation of unencrypted authenticated data, CTR-encrypted ciphertext, and padding. This data is split into m 128-bit blocks X_i:

$$X = X_1 \mid\mid X_2 \mid\mid \cdots \mid\mid X_m.$$

AES is used to derive the root authentication key $H = E_K(0)$. The same AES key K is also used as the data encryption key. In the present work we assume that H is unknown to the attacker as the scheme would be otherwise trivially breakable.

GHASH is based on operations in the finite field $\mathrm{GF}(2^{128})$. Horner's rule is used in this field to evaluate the polynomial Y.

$$Y_m = \sum_{i=1}^{m} X_i \times H^{m-i+1}. \tag{1}$$

Figure 1 illustrates how this value is usually computed (together with the CTR mode). The authentication tag is finalized with $T = Y_m + E_K(IV \mid\mid 0^{31} \mid\mid 1)$, assuming that a 96-bit Initialization Vector (IV) is used. The IV value must never be reused as that would lead to the "forbidden attack" discussed by Joux in [10].

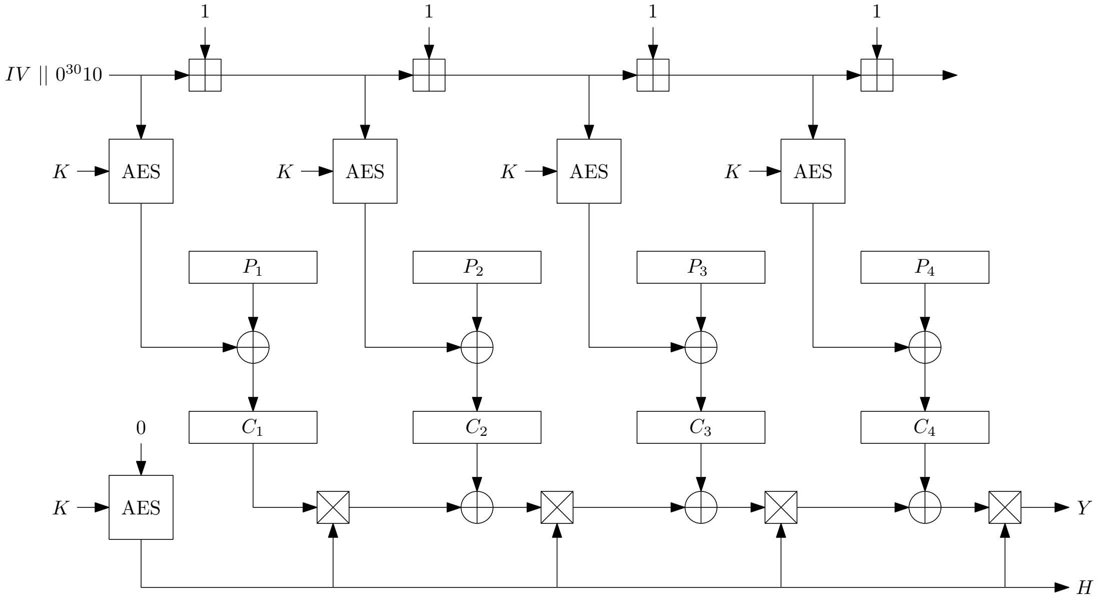


Fig. 1. Basic operation of first four rounds of GCM-CTR (without unencrypted authenticated data or padding). Here ⊞ denotes regular modular addition, ⊕ bitwise XOR operation, and ⊠ multiplication in $\mathrm{GF}(2^{128})$. The counter is initialized with IV and incremented by 1 for each block. This is used to to produce a keystream that is XORed over plaintext blocks P_i to produce ciphertext blocks C_i (or vice versa). The lower half of the diagram shows how the authentication tag is processed; each authenticated block is XORed over the state Y and multiplied with $H = E_K(0)$. The final processing of the authentication tag Y is omitted from this picture.

3 Collisions from Weak Keys

It has been observed that if $E_K(0) = H = 0$, the polynomial Y evaluates to zero and the security of GHASH breaks down. In fact, some sources assume that this pathological case is the only weak key [8]. AES keys K that produce this fixed point are not known.[1] However, It is easy to see why such keys should exist for AES, especially when the size of K is more than 128 bits.

Our main observation is that sometimes the powers of H will repeat in a relatively short cycle. A trivial example occurs when H is equal to the identity element 1, which will lead to all powers being equal. Due to the commutativity of addition in Equation 1, a GHASH collision can be achieved by swapping any two ciphertext blocks X_i and X_j. This amounts to message forgery.

More generally, if we know that $H^{m-i+1} = H^{m-j+1}$ with $i \neq j$, we may simply swap ciphertext blocks X_i and X_j and the resulting authentication tag stays unmodified which amounts to message forgery. This can be easily observed from Equation 1. Elementary group theory tells us that the powers of H will repeat in cycles which are determined by $n = \mathrm{ord}(H)$, the multiplicative order of H. Hence we may produce collisions by swapping X_i and X_{i+nm} for arbitrary i and m.

4 Cycle Lengths and Group Orders

From Lagrange's theorem in group theory we know that all subgroups divide the group of order $2^{128} - 1$. Numbers of this type factor into Fermat numbers

$$2^{2^n} - 1 = \prod_{i=1}^{n} 2^{2^{i-1}} + 1. \tag{2}$$

We can easily obtain the full factorization of $2^{128} - 1$:

$$3 * 5 * 17 * 257 * 641 * 65537 * 274177 * 6700417 * 67280421310721. \tag{3}$$

As this is a "smooth number", we can see that there are classes of H and therefore K values that produce cycles of length $n = 1, 3, 5, 15, 17, 51, \ldots$; any one of the $2^9 = 512$ subset products of the primes in Equation 3 is a valid group order.[2]

4.1 Illustrating Multiplicative Subgroup Cycles

Due to the peculiar way finite field arithmetic is defined in the GCM standard [14], the identity element with $\mathrm{ord}(H) = 1$ is:

```
H = 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Apparently this was considered as the "first bit" by those who originally implemented GCM. Otherwise standard polynomial arithmetic is used with the field representation defined by the reducing polynomial $x^{128} + x^7 + x^2 + x + 1$.

The following two elements will produce a cycle of length $\mathrm{ord}(H) = 3$ (the cycle obviously goes through the identity as well):

```
H = 10 D0 4D 25 F9 35 56 E6 9F 58 CE 2F 8D 03 5A 94
H = 90 D0 4D 25 F9 35 56 E6 9F 58 CE 2F 8D 03 5A 94
```

These four elements have $\mathrm{ord}(H) = 5$:

[1] Some block ciphers such as GOST allow such fixed-point keys to be very easily found.

[2] The term *smooth number* comes from factorization theory and indicates that a number factors into a large number of small primes.

```
H = 46 36 BD BD 1C 76 43 D3 4E E4 BB 1B F9 CA 08 4F
H = 92 17 8D 40 26 DA 1D CA 42 96 77 87 30 EB 9A 9E
H = 82 C7 C0 65 DF EF 4B 2C DD CE B9 A8 BD E8 C0 0A
H = D6 E6 F0 98 E5 43 15 35 D1 BC 75 34 74 C9 52 DB
```

We do not know which actual AES keys produce these H values, nor do we recommend testing against these particular values as the probability of hitting them is exceedingly small.

Note that a cycle of length such as $15 = 3 * 5$ also contains the beforementioned component groups of order 1, 3 and 5, in addition to the 8 unique elements that can act as a generator of the cycle of order 15. This is entirely analogous to arithmetic in the addition group of integers modulo 15; 0 will generate a "cycle" of one element when repeatedly added to itself, 5 and 10 will generate a cycles of order 3, the four elements { 3, 6, 9, 12 } cycles of order 5 and the rest of the numbers will have order 15. This is illustrated in Figure 2.

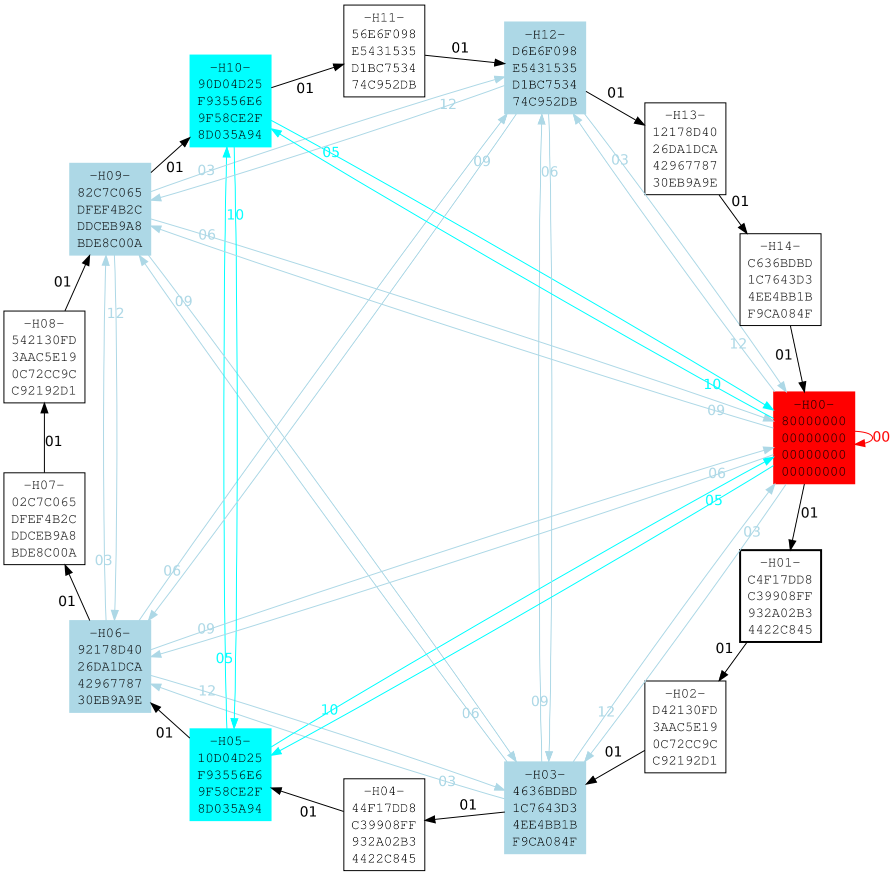


Fig. 2. The cycle of length 15 generated by the element $H =$ C4 F1 7D D8 C3 99 08 FF 93 2A 02 B3 44 22 C8 45. This is one of eight elements that generate a multiplicative subgroup in GCM's $\mathrm{GF}(2^{128})$ which is isomorphic to the additive group $\mathbb{Z}_{15}$. The identity element and subgroups of sizes 3 and 5 are also demonstrated. There are 512 multiplicative subgroups of different sizes in this particular field.

5 Message Forgery

We know that the field $\mathrm{GF}(2^{128})$ offers a generous serving of $2^9 = 512$ different multiplicative subgroups. Figure 3 shows that these are quite evenly distributed in the range due to the nearly log-uniform progression of the factors.

In our attack the adversary does not know H but will simply attempt a blind forgery by swapping two (or more) message blocks in transit as discussed in Section 3.

It is easy to show that it is sufficient that the group order divides the distance between swapped elements. Since each subgroup of size n has exactly n elements, we arrive at the following observation:

Theorem 1. *Let n be a number satisfying $\gcd(2^{128} - 1, n) = n$. Blindly swapping blocks X_i and X_j, where $i \equiv j \pmod{n}$ will result in a successful forgery with probability of at least $\frac{n+1}{2^{128}}$ for some random H.*

Proof. The distance congruence implies that the distance between X_i and X_j is a multiple of n. The $\gcd(2^{128} - 1, n) = n$ condition implies that n is one of the $2^9 = 512$ possible multiplicative subgroup sizes in $\mathrm{GF}(2^{128})$. If indeed $\mathrm{ord}(H) \mid n$ then $H^i = H^j$ and the forgery is successful due to commutativity of equation 1. We observe that the cycles are unique; there are n members in a subgroup of size n and the set of n elements is unique to each subgroup size. Hence the probability of hitting one of these cycle elements is $\frac{n}{2^{128}}$. In addition there is the pathological case $H = 0$ which completes the proof. □

If the $\gcd$ condition given in Theorem 1 does not hold, we have no reason to expect that the forgery is successful with a probability higher than $\frac{1}{2^{128}}$.

Assuming that an oracle has indicated a successful message forgery, any number of consecutive forgeries can be produced with probability 1 if the key remains unchanged (IV may change).

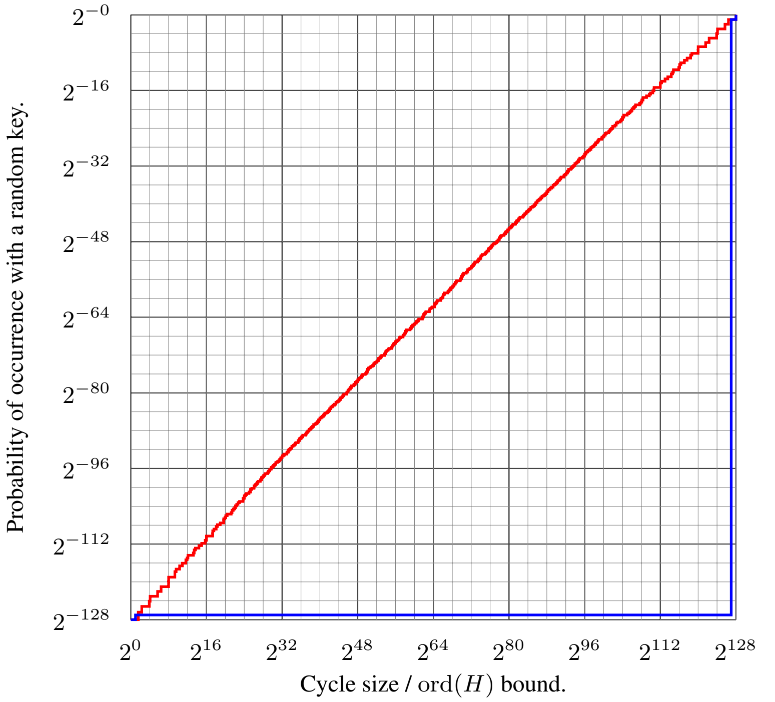


Fig. 3. GCM / GHASH: probability of hitting a multiplicative subgroup (cycle) of given (or smaller) size with a random authentication generator H in $\mathrm{GF}(2^{128})$. For comparison we also graph the security for $\mathrm{GF}(2^{127})$, which is entirely contained in the lower and right borders of the graph due to the fact that its multiplicative group order $2^{127} - 1$ is a prime.

6 Targeted Multiple Bit Forgeries

Our attacks enable elaborate message forgeries against authenticated encryption hybrids such as GCM due to the fact that the CTR encryption mode behaves like a stream cipher; flipping a ciphertext bit will result the corresponding plaintext bit to be flipped. This is especially true for lightweight protocols that combine a short binary polynomial MAC with a stream cipher.

If $\mathrm{ord}(H)|(i-j)$ the authentication tag will remain valid as long as the equation

$$X_i \times H^{m-i+1} + X_j \times H^{m-j+1} = c \tag{4}$$

holds for some (unknown) constant c related to the authentication tag. If we write $H^{m-i+1} = H^{m-j+1} = H_c$, this can be simplified to

$$X_i + X_j = c \times H_c^{-1}. \tag{5}$$

We see that the authentication tag will be valid if the sum of ciphertext blocks on the left side of Equation 5 remains constant. One may therefore flip *individual bits* in block X_i if the corresponding bit in X_j is also flipped. Any number of such modifications can be done to a message without affecting the probability of success (assuming that the same distance is used) indicated by Theorem 1.

7 Testing for AES-GCM Weak Keys

We know that finding weak H values is easy, so a natural question arises on how to determine weak AES keys K that produce these weak H roots.

To determine group order, we use a simple algorithm which is related to the Silver-Pohlig-Hellman algorithm for discrete logarithms [15]. Our algorithm is based on the following elementary observation:

Theorem 2. *Let p be one of the prime divisors given in Equation 3. If and only if p divides* $\mathrm{ord}(H)$ *we have*

$$H^{\frac{2^{128}-1}{p}} \neq 1. \tag{6}$$

Proof. Let g be a generator of the full multiplicative group; $\mathrm{ord}(g) = 2^{128} - 1$. Then each element $H \neq 0$ can be expressed as a power $H = g^h$ for some h, $0 \leq h < 2^{128} - 1$. Raising an element to power q, where $q \mid 2^{128} - 1$, sets the index modulo q to zero: $(g^h)^q = g^{qh}$. Since $\frac{2^{128}-1}{p}$ is divisible with all prime divisors q_i of the group order except p, we see that the condition of Equation 6 only holds if $h \neq 0 \pmod{p}$, which is equivalent to the condition $p \mid \mathrm{ord}(H)$. □

By performing the exponentiation test of Theorem 2 for each one of the nine prime divisors of $2^{128}-1$ in Equation 3, we may completely determine the multiplicative order of H.

7.1 An Efficient Algorithm for Subgroup Size

Raising a finite field element to a Fermat $F_n = 2^{2^n} + 1$ power can be done efficiently. It is well known that squaring operation is “linear” in $\mathrm{GF}(2^n)$ [7]. For $\mathrm{GF}(2^{128})$, a unique 128×128 bit matrix $\mathbf{M}_0$ exists that satisfies

$$X^2 = \mathbf{M}_0 X \tag{7}$$

for all X. In the following $\mathbf{M}_0 X$ denotes a matrix multiplication where X is interpreted as a vector of 128 bits and $X \times X = X^2$ is a multiplication where X is interpreted as a (polynomial) member of $\mathrm{GF}(2^{128})$.

By squaring $\mathbf{M}_0$, we obtain $\mathbf{M}_1 = \mathbf{M}_0^2$ which satisfies $X^4 = \mathbf{M}_1 X$ for all X. By repeating this process we can rapidly compute $\mathbf{M}_0, \mathbf{M}_1, \ldots, \mathbf{M}_6$ that satisfy

$$X^{2^{2^i}} = \mathbf{M}_i X. \tag{8}$$

Once the matrices (table lookups) $\mathbf{M}_i$ have been initialized, raising the authentication key H to a Fermat number power can be achieved with:

$$H^{F_n} = \mathbf{M}_n H \times H. \tag{9}$$

Therefore this operation can be made with a table lookup (multiplication with $\mathbf{M}_n$) and a single Galois Field multiplication. The matrices need to be computed only once as they are independent from particular H.

Since $2^{128} - 1 = \prod_{i=0}^{6} F_i$, checking whether the group order is of H is divisible with Fermat number F_i involves raising H to all Fermat powers F_j *except* F_i. For example, to check whether or not group order is divisible with $F_3 = 257$, we may see if this equation holds:

$$\mathbf{M}_6(\mathbf{M}_5(\mathbf{M}_4(\mathbf{M}_2(\mathbf{M}_1(\mathbf{M}_0 H \times H) \times H) \times H) \times H) \times H) \times H = 1. \tag{10}$$

The Fermat numbers F_5 and F_6 are not primes (unlike F_0, F_1, F_2, F_3 and F_4 which are indeed the only known Fermat primes). Here the technique involves first powering H to all Fermat powers except $F_5 = 641 * 6700417$ or $F_6 = 274177 * 67280421310721$. Then then we use a conventional square-multiply exponentiation method to individually check these two subfactors.

In practice the matrix $\mathbf{M}_i$ multiplication is implemented as byte-based table lookups with seven $16 \times 256 \times 128$ - bit tables. The initialization of these tables is very fast as $\mathbf{M}_{i+1}$ can be developed from $\mathbf{M}_i$ with a loop of $16 * 256$ table lookups. Significant speedups are achieved by reusing partial results.

7.2 Experimental Results

Using the techniques outlined in the previous subsection, we have developed a reasonably efficient cycle determination code specifically for GCM's GF(2^{128}), together with an AES-128 key setup and encryption function for deriving H values from K values.

Our implementation is currently able to fully determine the order of 25000 AES keys per second on a low-end Linux laptop that has a single 1.7 gHz AMD V140 processor.

Over couple of days we tested 2^{32} AES-128 keys and found progressively smaller subgroups:

$n \approx 2^{126.4}$	$K =$ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 02
$n \approx 2^{125.6}$	$K =$ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 03
$\dots$	
$n \approx 2^{96.52}$	$K =$ 00 00 00 00 00 00 00 00 00 00 00 00 24 3E 8B 40
$n \approx 2^{96.00}$	$K =$ 00 00 00 00 00 00 00 00 00 00 00 00 37 48 CF CE
$n \approx 2^{93.93}$	$K =$ 00 00 00 00 00 00 00 00 00 00 00 00 42 87 3C C8
$n \approx 2^{93.41}$	$K =$ 00 00 00 00 00 00 00 00 00 00 00 00 EC 69 7A A8

As indicated by Figure 3, a significantly smaller group than $2^{128-32} = 2^{96}$ was found with 2^{32} effort, due to the large number of multiplicative subgroup sizes available in GF(2^{128}).

There is clearly room for improvement. The search is fully parallelizable, and hence a massively parallel FPGA or GPU-based search could be performed to find subgroups of magnitude $n \approx 2^{64}$ or less.

8 Other Polynomial-Evaluation MACs

The security of Polynomial-evaluation MACs against attacks of this type can be determined from the factorization of the group size in straightforward fashion. Trivial changes can introduce radical differences.

One may consider this difference by comparing the binary field $GF(2^{127})$ and the prime field GF($2^{127} - 1$). Here the binary field is perfectly secure due to the fact that $2^{127} - 1$ is indeed a prime (if the message is processed in 127-bit blocks). However, the latter prime field has a multiplicative order $2^{127} - 2$ which factors spectacularly into 15 pieces and is exceptionally weak against a cycling attack! We note that the HASH127 MAC is based on the latter [4]. This is illustrated in Figure 3.

If a prime field is to be used, we recommend Sophie Germain primes where $q = (p - 1)/2$ is also a prime. Such a field has well-understood cycle properties which may be easily determined using the Legendre symbol from elementary number theory. A practical alternative to GCM would use a Sophie Germain prime such as $GF(2^{128}+12451)$, which is slightly larger than the 2^{128} to deter trivial collisions.

It is clear that risks rise quadratically when GCM is used with a 64-bit block cipher as suggested in Appendix A of [12]. There is a substantial risk of hitting a bad long-term key and therefore we recommend against using the 64-bit GCM.

9 Conclusions and Future Work

We have shown that the GHASH algorithm has other weak key classes besides the trivial $H = 0$ case considered in current literature [8]. This is a result of the multiplicative group of $\mathrm{GF}(2^{128})$ having a particularly smooth order.

Our attacks allow specific plaintext bits to be targeted by modifying ciphertext bits, which can have a devastating effect when a short polynomial MAC over a binary field is combined with a stream cipher in a (lightweight) communication protocol. The probability of randomly hitting an exploitable weak key with a AES-GCM cryptographic protocol such as SSH [9], IPSec [11] or TLS [16] is very small.

However, malicious players may exploit subtle weaknesses in cryptographic protocols in surprising ways. One feature of cycle attacks is that an attacker may first test for short cycles and then force a re-keying event if the test fails; once a long-term key with a short cycle is found, she may exploit it any number of times.

We have also described a straightforward method of detecting GHASH weak keys. We performed an exhaustive experiment that found many AES-128 keys that produce H with order below $n \approx 2^{96}$.

We suggest that binary fields $\mathrm{GF}(2^n)$ with prime $2^n - 1$ or Sophie Germain prime fields are used in constructions of this type as this minimizes the total number of weak keys. This was illustrated with the surprising observation that $\mathrm{GF}(2^{127})$ is perfectly secure against this type of attack while GCM's $\mathrm{GF}(2^{128})$ is not.

One interesting future research direction and open question is the feasibility of mapping the weak H values to K symmetric keys with various block ciphers other than AES.

References

1. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: CRYPTO '96. LNCS, vol. 1109, pp. 1 – 55. Springer (1996)
2. Bernstein, D.J.: The Poly1305-AES message-authentication code. In: FSE 2005. LNCS, vol. 3557, pp. 32–49. Springer (2005)
3. Bernstein, D.J.: Stronger security bounds for Wegman-Carter-Shoup authenticators. In: EUROCRYPT 2005. LNCS, vol. 3494, pp. 164–180. Springer (2005)
4. Bernstein, D.J.: Floating-point arithmetic and message authentication. `http://cr.yp.to/papers.html#hash127` (1999)
5. Bierbrauer, J., Johansson, T., Kabatianskii, G., Smeets, B.: On families of hash functions via geometric codes and concatenation. In: CRYPTO '93. LNCS, vol. 773, pp. 331–342. Springer (1994)
6. den Boer, B.: A simple and key-economical unconditional authentication scheme. Journal of Computer Security 2, 65–71 (1993)
7. Ferguson, N.: Authentication weaknesses in GCM. NIST Comment. (May 2005)
8. Handschuh, H., Preneel, B.: Key-recovery attacks on universal hash function based MAC algorithms. In: CRYPTO 2008. LNCS, vol. 5157, pp. 144–161. Springer (2008)
9. Igoe, K., Solinas, J.: AES Galois counter mode for the secure shell transport layer protocol. IETF Request for Comments 5647 (2009)
10. Joux, A.: Authentication failures in NIST version of GCM. NIST Comment (2006)

11. Law, L., Solinas, J.: Suite B cryptographic suites for IPsec. IETF Request for Comments 4869 (2007)
12. McGrew, D.A., Viega, J.: The Galois/counter mode of operation (GCM). Submission to NIST. (2005)
13. NIST: The advanced encryption standard (AES). FIPS Publication 197 (2001)
14. NIST: Recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC. NIST Special Publication 800-38D (2007)
15. Pohlig, S., Hellman, M.: An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. IEEE Transactions on Information Theory 24(1), 106–110 (1978)
16. Salter, M., Rescorla, E., Housley, R.: Suite B profile for transport layer security (TLS). IETF Request for Comments 5430 (2009)
17. Sarkar, P.: A trade-off between collision probability and key size in universal hashing using polynomials. Designs, Codes and Cryptography 58(3), 271–278 (2011)
18. Taylor, R.: An integrity check value algorithm for stream ciphers. In: CRYPTO '93. LNCS, vol. 773, pp. 40–48. Springer (1994)
19. Wegman, M.N., Carter, J.L.: New classes and applications of hash functions. In: 20th annual symposium on foundations of computer science. IEEE Computer Society, New York (1979)
20. Wegman, M.N., Carter, J.L.: New hash functions and their use in authentication and set equality. Journal of Computer and System Sciences 22, 265–279 (1981)

Document version 20120229072900.

Collision Attacks on the Reduced Dual-Stream Hash Function RIPEMD-128

Florian Mendel[1], Tomislav Nad[2], Martin Schläffer[2]

[1] Katholieke Universiteit Leuven, ESAT/COSIC and IBBT, Belgium
[2] Graz University of Technology, IAIK, Austria

Abstract. In this paper, we analyze the security of RIPEMD-128 against collision attacks. The ISO/IEC standard RIPEMD-128 was proposed 15 years ago and may be used as a drop-in replacement for 128-bit hash functions like MD5. Only few results have been published for RIPEMD-128, the best being a preimage attack for the first 33 steps of the hash function with complexity $2^{124.5}$. In this work, we provide a new assessment of the security margin of RIPEMD-128 by showing attacks on up to 48 (out of 64) steps of the hash function. We present a collision attack reduced to 38 steps and a near-collisions attack for 44 steps, both with practical complexity. Furthermore, we show non-random properties for 48 steps of the RIPEMD-128 hash function, and provide an example for a collision on the compression function for 48 steps.
For all attacks we use complex nonlinear differential characteristics. Due to the more complicated dual-stream structure of RIPEMD-128 compared to its predecessor, finding high-probability characteristics as well as conforming message pairs is nontrivial. Doing any of these steps by hand is almost impossible or at least, very time consuming. We present a general strategy to analyze dual-stream hash functions and use an automatic search tool for the two main steps of the attack. Our tool is able to find differential characteristics and perform advanced message modification simultaneously in the two streams.

Keywords: hash functions, RIPEMD-128, collisions, near-collisions, differential characteristic, message modification, automatic tool

1 Introduction

In the last few years, the cryptanalysis of hash functions has become an important topic within the cryptographic community. Especially the collision attacks on the MD4 family of hash functions have weakened the security assumptions of many commonly used hash functions. Still, most of the existing cryptanalytic work has been published for this particular family of hash functions [17,19,20]. In fact, practical collisions have been shown for MD4, MD5, RIPEMD and SHA-0. For SHA-1, a collision attack has been proposed with a complexity of about 2^{63} [18]. However, some members of this family including the ISO/IEC standard RIPEMD-128 (the successor of RIPEMD) seems to be more resistant against these attacks. In this paper, we analyze the security of RIPEMD-128 against collision attacks and show that the security margin is less than expected.

Related Work. Since its proposal 15 years ago only a few results have been published for RIPEMD-128. Most published results are concerning the preimage resistance of the hash function [13,16]. The best currently known attack is a preimage attack for 33 steps and 36 intermediate steps of the hash function with a complexity only slightly faster than the generic complexity of 2^{128} [16]. The only work regarding the collision resistance of RIPEMD-128 has been published by Mendel et al. [11], where the application of the differential attacks on RIPEMD by Dobbertin [5] and Wang et al. [17] is studied. However, due to the increased number of steps and the fact that the two streams are more different than in RIPEMD, they concluded that RIPEMD-128 is secure against this type of attacks.

Our Contribution. In this paper, we first provide a general strategy to analyze dual-stream hash functions in Sect. 2. We analyze different methods to find high-probability differential characteristics which work for both streams. Similar as in the attack on RIPEMD [17], characteristics in two streams are impossible with a high probability. Therefore, in our attacks an automatic search tool is essential for finding valid differential characteristics [4, 10]. This is especially important in the first round of a hash function where characteristics are usually quite dense. In this first round, one usually assumes that conditions imposed by the characteristic can be fulfilled efficiently using message modification techniques. However, message modification is much more difficult in the dual-stream case since two state words are updated using a single message word. This reduced freedom could in general be compensated with hand-tuned advanced message modification techniques [8, 9, 15, 20]. However, another contribution of our work is to provide a fully automatic tool which can be used to find conforming message pairs in the first round of a dual-stream hash function.

Table 1. Summary of our new and previous results on RIPEMD-128.

component	attack	steps	complexity	generic	reference
hash	collision	38	example, 2^{14}	2^{64}	Sect. 4
hash	near-collision	44	example, 2^{32}	$2^{47.8}$	Sect. 5.1
hash	non-randomness	48	2^{70}	2^{76}	Sect. 5.2
compression	collision	48	example, 2^{40}	2^{64}	Sect. 5.3
hash	preimage	33	$2^{124.5}$	2^{128}	[13]
hash	preimage	interm. 35	2^{121}	2^{128}	[13]
hash	preimage	interm. 36	$2^{126.5}$	2^{128}	[16]

We apply our attack strategy and tools to the ISO/IEC standard RIPEMD-128 which we describe in Sect. 3. Using our automatic tools, we are able to construct the first practical collisions for up to 38 steps of RIPEMD-128 with a complexity of 2^{14}. We describe the collision attack in details in Sect. 4. The attack can be extended (Sect. 5) to practical near-collisions on 44 steps with complexity 2^{32}. Furthermore, we provide a theoretical distinguisher of the hash function for 48 steps (3 out of 4 rounds) and show that 3 rounds of the RIPEMD-128 compression function are not collision free. Our results are summarized in Table 1, together with all known previous results. Finally, we conclude in Sect. 6 and discuss directions of future work on hash functions with parallel state update transformation.

2 Cryptanalysis of Dual-Stream Hash Functions

In this section, we describe our attack strategy for the cryptanalysis of dual-stream hash functions. The general attack strategy is based on the recent results in cryptanalysis of the MD4-family of hash functions [17, 20]. However, the application of this strategy is nontrivial in the case of dual stream hash functions. Since in each step, one message word is used to update two state words, the freedom of an attacker in finding valid differential characteristics and performing message modification is limited. Hence, a more careful analysis is required to overcome this problem.

2.1 Collision Attacks on Hash Functions

In the following, we first give a brief overview of the attack strategy used in the recent collision attacks on the MD4-family of hash functions [17, 20]. All attacks basically use the same strategy which we adopt for dual-stream hash functions. The high-level strategy can be summarized as follows:

1. Find a characteristic for the hash function that holds with high probability after the first round of the hash function.
2. Find a characteristic (not necessary with high probability) for the first round of the hash function.
3. Use message modification techniques to fulfill conditions imposed by the characteristic in the first round. This increases the probability of the characteristic.
4. Use random trials to find values for the remaining free message bits such that the message follows the characteristic.

The most difficult and important part of the attack is to find a good differential characteristic for both the first round and the remaining rounds of the hash function, since this defines the final attack complexity. There are several methods to find good differential characteristics. The second important part of the attack is to find conforming inputs for the complex nonlinear differential characteristic in the first round of the hash function using message modification techniques.

2.2 Collision Attacks on Dual-Stream Hash Functions

In the following, we will describe our approach to construct good differential characteristics and find colliding message pairs for dual-stream hash functions. We focus on hash functions like RIPEMD-128, but the general idea is applicable to any hash function with two or more streams.

Finding suitable differential characteristics. If the two streams of the hash function are the same except for constant additions, the same differential characteristic can be used in both streams. For instance, in the case of RIPEMD, the permutation and rotation values are indeed equal for both streams. Hence, it is sufficient to find a collision-producing characteristic for only one stream (3 rounds) and apply it simultaneously to both streams [17]. Nevertheless, the number of necessary conditions increases for two streams. Hence, it is more likely to have contradicting conditions. In fact, Wang et al. reported that among 30 selected collision-producing characteristics only one can produce a real collision.

If the two streams are more different, we first need to find a differential characteristic for the hash function after round 1, which holds with a high probability in both streams. One approach is to find such characteristics is to use a linearized model of the hash function and algorithms from coding theory [2, 7, 14]. This works quite well for hash functions with a regular message expansion and step update transformation (like SHA-1), and can be applied to dual-stream hash functions in a straight-forward way.

However, the linearization approach does not work well for hash functions with a permutation of words in the message expansion and different rotation values in the state update transformation (RIPEMD-128 and RIPEMD-160). One usually gets linear differential characteristics with high Hamming weight and hence, a high complexity. However, for such hash functions, we can still make use of the approach of Wang et al. in the attacks on MD4, RIPEMD and MD5 [17,20]. The idea is to use differences in one or more message words to find local (or inner) collisions within a few steps in the last round(s) of the hash function. Then a suitable characteristic for the remaining steps, preferably also using short local collisions, has to be constructed. Although this is obviously more difficult for dual-stream hash functions, we were able to construct such high-probability differential characteristics for reduced RIPEMD-128 (see Sect. 4.1).

Once, the characteristic after round 1 is fixed we need to find a characteristic (not necessary with high probability) for the first round of the hash function for both streams. Note that in the previous part of the attack it might still be possible to construct inner collisions with hand by choosing the differences carefully. However, to construct a valid nonlinear differential characteristic for both streams in the first round, an automatic search tool is needed. While one can use complex differential characteristics in both streams, we aim for differential characteristics that are sparse in at least one of the two streams, since such sparse characteristics will then also reduce the complexity of the message modification step.

Using message modification techniques. Once we have fixed the differential characteristic for both streams we start with the message search. In the first round, the freedom of the whole message block can be used to get a conforming message pair for the first 16 steps. For single-stream hash function, basic message modification techniques simply choose conforming state words and invert each step update transformation to get the message word [20]. However, as already noted by Wang et al. [17], message modification is more complicated for two streams since the conditions on two state words need to be fulfilled using a single message word. While in RIPEMD the same message word is used in the same step of the left and right stream, this is not the case in RIPEMD-128, which significantly increases the complexity of message modification.

In the attack on RIPEMD, two techniques have been proposed exploiting the freedom of other message bits using carry effects, the Boolean function and previous message words. The same rotation values in RIPEMD allow an easier application of this idea since it is still possible to fulfill conditions from LSB to MSB. However, for streams with different rotation values, previously corrected conditions may become invalid again. In general, conditions on two state words using a single message word can be fulfilled using advanced message modification techniques. Many dedicated techniques have been proposed in recent years [8, 9, 15, 20], which could also be used to fulfill conditions in the first round of dual-stream hash functions.

To simplify the message modification we use a more general approach. Instead of complicated, dedicated techniques, we use an automated tool for the message modification in the first round. To be more precise, we use the same tool as for the differential path search in the first round. Instead of searching for valid differential characteristics in both streams, we search for valid bit-wise assignments of 0's and 1's to the message and state bits in the first round. Since we solve for conforming message words bit-wise, a different message word permutation, different rotation values and carry effects are handled automatically, similar as in the search for differential characteristics. Moreover, this approach can be generalized to any ARX based design.

The disadvantage of our automated bit-wise approach is a slightly higher complexity, compared to a hand-tuned word-wise approach. However, this increased costs can be amortized by randomizing message words at the end of round 1 to find solutions efficiently for the high-probability characteristic of the remaining rounds.

2.3 Automatic Search Tool

The application of the above strategies is far from being trivial and requires an advanced set of techniques and tools to be successful. Due to the increased complexity of dual-stream hash functions with different streams, finding good differential characteristics by hand is almost impossible. Therefore, we have developed an automatic tool which can be used for finding complex nonlinear differential characteristics as well as for solving nonlinear equations involving conditions on state words and free message bits, i.e. to find confirming message pairs. Our tool is based on the approach of Mendel et al. [10] to find both complex nonlinear differential characteristics and conforming message pairs for SHA-2.

The basic idea is to consider differential characteristics which impose arbitrary conditions on pairs of bits using generalized conditions [4]. Generalized conditions are inspired by signed-bit differences and take all 16 possible conditions on a pair of bits into account. Table 2 lists all these possible conditions and introduces the notation for the various cases.

Table 2. Notation for possible generalized conditions on a pair of bits [4].

$(X_i, {X_i}^*)$	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$	(X_i, X_i^*)	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
?	✓	✓	✓	✓	3	✓	✓	-	-
-	✓	-	-	✓	5	✓	-	✓	-
x	-	✓	✓	-	7	✓	✓	✓	-
0	✓	-	-	-	A	-	✓	-	✓
u	-	✓	-	-	B	✓	✓	-	✓
n	-	-	✓	-	C	-	-	✓	✓
1	-	-	-	✓	D	✓	-	✓	✓
#	-	-	-	-	E	-	✓	✓	✓

Using these generalized conditions and propagating them in a bitsliced manner, we can construct complex differential characteristics in an efficient way. The basic idea of the search algorithm is to randomly pick a bit from a set of bit positions with predefined conditions, impose a more restricted condition and compute how this new condition propagates. This is repeated until an inconsistency is found or all unrestricted bits from the set are eliminated. Note that this general approach can be used for both, finding differential characteristics and conforming message pairs.

For example, the search strategy for finding nonlinear characteristics works as follows (for a more detailed description of the search algorithm or how the conditions are propagated we refer to [4,10]):

1. Define a set of unrestricted bits (?) and unsigned differences (x).
2. Pick a random bit from the set.
3. Impose a zero-difference (-) on unrestricted bits (?), or randomly choose a sign (u or n) for unsigned differences (x).
4. Check how the new conditions propagate.
5. If an inconsistency occurs, remember the last bit and jump back until this bit can be restricted without leading to a contradiction.
6. Repeat from step 2 until all bits from the set have been restricted.

We use the same strategy to find conforming input pairs for a given differential characteristic. Instead of picking an unrestricted bit (?) we pick an undetermined bit without difference (-) and assign randomly a value (0 or 1) until a solution is found:

1. Define a set of undetermined bits without difference (-).
2. Pick a random bit from the set.
3. Randomly choose the value of the bit (0 or 1).
4. Check how the new conditions propagate.
5. If an inconsistency occurs, remember the last bit and jump back until this bit can be restricted without leading to a contradiction.
6. Repeat from step 2 until all bits from the set have been restricted.

Note that the efficiency of finding a conforming message pair can be increased if the undetermined bits without difference (-) are picked in a specific order. The order strongly depends on the specific hash function. In general, fully determining word after word turns out to be a good approach for word-wise defined ARX-based hash functions. Using this approach, we can instantly (milliseconds) find solutions for the first round of dual-stream hash functions without the need for hand-tuned advanced message modification techniques.

3 Description of RIPEMD-128

RIPEMD-128 was designed by Dobbertin, Bosselaers and Preneel in [6] as a replacement for RIPEMD. It is an iterative hash functions based on the Merkle-Damgård design principle [3,12] and processes 512-bit input message blocks and produces a 128-bit hash value. To guarantee that the message length is a multiple of 512 bits, an unambiguous padding method is applied. For the description of the padding method we refer to [6].

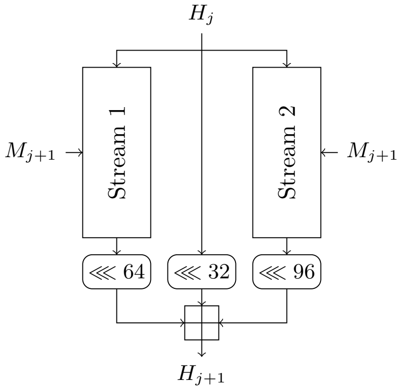


Fig. 1. Structure of the RIPEMD-128 compression function.

Like its predecessor, the function of RIPEMD-128 consists of two parallel streams. In each stream the state variables are updated corresponding to the message block and combined with the previous chaining value after the last step, depicted in Figure 1. While RIPEMD consists of two parallel streams of MD4, the two streams are designed differently in the case of RIPEMD-128. In the following, we describe the compression function in detail.

Each stream of the compression function of RIPEMD-128 basically consists of two parts: the state update transformation and the message expansion. Furthermore, RIPEMD-128 consists of a feed-forward where the input and output state words are added in a different order. For a detailed description we refer to [6].

State Update Transformation. The state update transformation of each stream starts from a (fixed) initial value IV of four 32-bit words B_{-4}, B_{-3}, B_{-2}, B_{-1}. and updates them in 4 rounds of 16 steps each. In each step one message word is used to update the four state variables. Figure 2 shows one step of the state update transformation of each stream of RIPEMD-128.

The function f is different in each round. f_r is used for the r-th round in the left stream, and f_{5-r} is used for the r-th round in the right stream ($r = 1, \ldots, 4$):

$$\begin{aligned} f_1(x, y, z) &= x \oplus y \oplus z, \\ f_2(x, y, z) &= (x \wedge y) \vee (\neg x \wedge z), \\ f_3(x, y, z) &= (x \vee \neg y) \oplus z, \\ f_4(x, y, z) &= (x \wedge z) \vee (y \wedge \neg z). \end{aligned}$$

A step constant K_r is added in every step; the constant is different for each round and for each stream. For the actual values of the constants we refer to [6], since we do not need them in the analysis. For both streams the following rotation values s given in Table 3 are used.

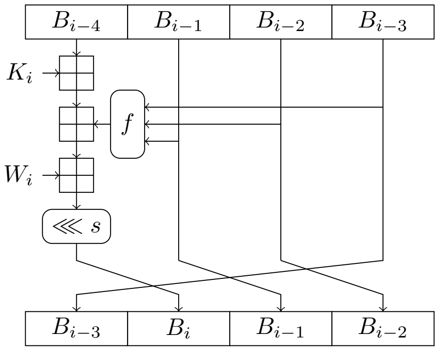


Fig. 2. The step update transformation of RIPEMD-128.

Table 3. The rotation values s for each step and each stream of RIPEMD-128.

	Step	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
left stream	Round 1	11	14	15	12	5	8	7	9	11	13	14	15	6	7	9	8
	Round 2	7	6	8	13	11	9	7	15	7	12	15	9	11	7	13	12
	Round 3	11	13	6	7	14	9	13	15	14	8	13	6	5	12	7	5
	Round 4	11	12	14	15	14	15	9	8	9	14	5	6	8	6	5	12
right stream	Round 1	8	9	9	11	13	15	15	5	7	7	8	11	14	14	12	6
	Round 2	9	13	15	7	12	8	9	11	7	7	12	7	6	15	13	11
	Round 3	9	7	15	11	8	6	6	14	12	13	5	14	13	13	7	5
	Round 4	15	5	8	11	14	14	6	14	6	9	12	9	12	5	15	8

Message Expansion. The message expansion of RIPEMD-128 is a permutation of the 16 message words in each round. Different permutations are used for the left and the right stream. For both streams the message words are permuted according to Table 4.

Table 4. The index of the message words m_i which are used as the expanded message words W_i in each step and each stream of RIPEMD-128.

	Step	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
left stream	Round 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	Round 2	7	4	13	1	10	6	15	3	12	0	9	5	2	14	11	8
	Round 3	3	10	14	4	9	15	8	1	2	7	0	6	13	11	5	12
	Round 4	1	9	11	10	0	8	12	4	13	3	7	15	14	5	6	2
right stream	Round 1	5	14	7	0	9	2	11	4	13	6	15	8	1	10	3	12
	Round 2	6	11	3	7	0	13	5	10	14	15	8	12	4	9	1	2
	Round 3	15	5	1	3	7	14	6	9	11	8	12	2	10	0	4	13
	Round 4	8	6	4	1	3	11	15	0	5	12	2	13	9	7	10	14

Feed-Forward. After the last step of the state update transformation, the initial values $B_{-4}, \ldots, B_{-1}$ and the output values of the last step of the left stream $B_{63}, \ldots, B_{60}$ and the last step of the right stream $B'_{63}, \ldots, B'_{60}$ are combined, resulting in the final value of one iteration (feed-forward). The result is the final hash value or the initial value for the next message block:

$$\begin{gathered} B_{-1} \boxplus B_{62} \boxplus B'_{61} \\ B_{-4} \boxplus B_{63} \boxplus B'_{62} \\ B_{-3} \boxplus B_{60} \boxplus B'_{63} \\ B_{-2} \boxplus B_{61} \boxplus B'_{60} \end{gathered}$$

4 Collision Attacks on RIPEMD-128

To find collisions in reduced RIPEMD-128 we use the strategy proposed in Sect. 2.2. The attack consists of 3 major parts given as follows:

1. **Starting Point:** Find a good start setting, i.e. differences in only a few specific message words that may lead in a differential characteristic with high probability after step 15.
2. **Differential Characteristic:** Search for a high-probability differential characteristic for the whole hash function where at most one stream has a low probability in step 0-15.
3. **Message Pair:** Find a colliding message pair using automated message modification and random trials.

4.1 Finding a Starting Point

In MD4-like hash functions, differences are introduced and canceled using differences in the expanded message words. Since RIPEMD-128 has two streams with different permutation of message words, the first step in the attack is to determine those message words which may contain differences. We have several constraints such that the whole attack can be carried out efficiently.

First of all, we aim for a high probability differential characteristics after step 15 in both streams. Such high probability differential characteristics can be constructed if the differences introduced by the message words are canceled immediately using local collisions spanning over only a few steps. The shortest local collision in the MD4 step update goes over 4 steps. However, due to the different message permutation used in each stream, it is difficult to achieve short local collisions in both streams simultaneously.

Another possibility is to cancel all differences in each stream as early as possible in round 2 and find message words, such that new differences are introduced late in round 3. A further constraint is to have a short local collision and hence sparse differential characteristic in one stream between step 0-15 such that the message modification part can be carried out more efficiently (see Sect. 2.2).

A single message word which seems to be a good choice is m_{13}. In this case, we get one short local collision between round 1 and round 2 in the left stream and one slightly longer local collision between round 1 and round 2 in the right stream. Both local collisions end in the first few steps of round 2. Furthermore, the message word m_{13} introduces differences very late in the last few steps of round 3 (see Fig. 3). Note that a similar approach was used by Dobbertin in the attack on RIPEMD [5]. Unfortunately, no local collision spanning over 5 steps in the left stream between round 1 and 2 can be constructed which renders the attack impossible.

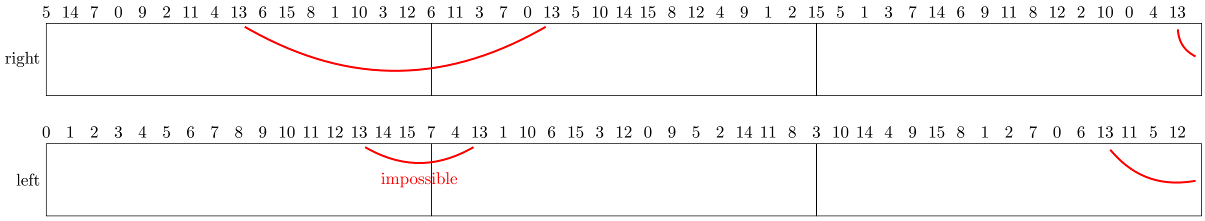


Fig. 3. Using only message word m_{13}.

A better choice is to use differences in two message words, like it was done by Wang et al. in the attack on RIPEMD [17]. If we choose differences in m_0 and m_6 then we get for the left stream one local collision over 6 steps in round 1, and another local collision over 4 steps in round 2. Note that in the right stream a short local collision over 4 steps (step 16-20) is actually impossible. This is due to the fact that for f_3 (ONX-function), a local collision over 4 steps with differences in only two message does not exist. Hence, we combine in the right stream the two local collisions resulting in one long local collision between step 3 and 20. In round 3, the first difference is added in step 38. Hence, using this starting point we can get a collision for 38 steps of RIPEMD-128.

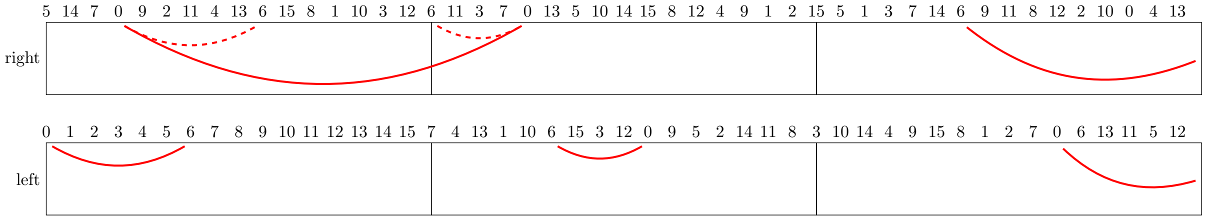


Fig. 4. Using message words m_0 and m_6.

4.2 Finding a Differential Characteristic

Once we have fixed the starting point, i.e. the message words which may contain differences, we use an automated tool to find high-probability differential characteristics. The 38-step start characteristic given in Table 8 is the starting point for almost all our attacks. Note that we do not fix the message difference prior to the search to allow the tool to find an optimal solution.

In order to get a differential characteristics resulting in a low attack complexity, we aim for a low Hamming weight difference in state word B_{21}. The best we could find is a differential characteristic with 2 differences in B_{21} (see Table 9). Furthermore, the Boolean function XOR in the first round of the left stream provides less freedom in constructing local collisions than the non-linear functions. Hence, we first search for a differential characteristic in the left stream.

Once the characteristic in the left stream is fixed, we use an arbitrary first message block to fulfill the conditions on the chaining value. Since we have 14 conditions on the chaining value (see Table 9), finding the 1st block has a complexity of about 2^{14}.

Next, we search for a differential characteristic in the right stream. To get a low complexity for the message search in round 2, we search for characteristics with only a few differences in state words B'_{14} and B'_{15}. Using our search tool, we can find many differential characteristic for the left and right stream within only a few minutes on an ordinary PC. A colliding differential characteristic for 38 steps of RIPEMD-128 is given in Table 10.

4.3 Finding a Confirming Message Pair

To fulfill all conditions imposed by the differential characteristic in the first round, we need to apply message modification techniques. Since we have many conditions in the first 6 steps of the left stream and the first 15 steps of the right stream this may not be an easy task. However, using our tool and generalized conditions, we can do message modification for the first 16 steps efficiently and immediately within milliseconds on a PC. Of course, by hand-tuning basic message modification the complexity might be improved, but using our tool this phase of the message search can be fully automated. Furthermore, the cost of message modification is fully amortized by randomizing e.g. message word m_{12} to find a solution also for the high-probability characteristic in round 2 (and 3). Using the approximately 2^{30} possible value for m_{12}, we can find a solution for the differential characteristic (complexity 2^{14} after round 1) including message modification in less than a second on our PC. The resulting message pair for a collision on 38 steps of RIPEMD-128 is given in Table 5.

Table 5. Collision for 38 steps of RIPEMD-128.

M_1	9431bddf 7b9827d6 f54a64a9 df41a58a fd707a50 dad10eb6 48b0cc76 be66cb8c ab3b7afa 084ba98e ab0a4798 2a4b0d06 a79bf8b7 3fd6008a 4da2112d 849c5b9c
M_2	952bc70f d0840848 eafffa57 0ca3c38a 45383ffb ddc6a9a1 796f1e20 0b9ff55f ddb80113 f0ffe1b5 b7d75dc0 82c7298f f2c442f4 96cbf293 c441d662 06e9eec2
M_2^*	952bc50e d0840848 eafffa57 0ca3c38a 45383ffb ddc6a9a1 79ef1e21 0b9ff55f ddb80113 f0ffe1b5 b7d75dc0 82c7298f f2c442f4 96cbf293 c441d662 06e9eec2
ΔM_2	00000201 00000000 00000000 00000000 00000000 00000000 00800001 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
H_2	a0a00507 fd4c7274 ba230d53 87a0d10a
H_2^*	a0a00507 fd4c7274 ba230d53 87a0d10a
ΔH_2	00000000 00000000 00000000 00000000

5 Extending the Attack to More Steps

In this section, we will show how the collision attack on 38 steps can be extended to more steps of the hash function by using a weaker attack setting, i.e. near-collisions and subspace distinguisher. Furthermore, we present a free-start collision for 48 steps of RIPEMD-128 compression function.

5.1 Near-Collisions for the Hash Function

It is easy to see that by appending 6 steps to the characteristic for 38 steps (see Table 11) one gets a near-collision for 44 steps of the hash function with only 6 differences in the hash value. However, note that while in the collision attack one can always append a message block with the correct padding this can not be done for a near-collision. Hence, in order to construct a near-collision for the hash function the padding has to be fixed on beforehand. Luckily, we have such a high amount of freedom in our attack the we can easily fix m_{15}, m_{14} and parts of m_{13} in the attack to guarantee that the padding is correct. The result is a practical near-collision (see Table 6) for 44 steps of RIPEMD-128 with complexity of 2^{32}. Note that the generic attack to find a near-collision with only 6 differences in the hash value has a complexity of about $2^{47.8}$.

Table 6. Near-collision for 44 steps of RIPEMD-128.

M_1	2ca95052 425a8f73 08be4537 c790e019 0dcc7d4e 29075123 75327262 8d0d4803 1e57a6a4 73550688 59263eb1 98c6f6ce f03b8b4b 62d3fdf7 638db196 68c0b7b3
M_2	aa1437ef f3646663 c339343a 52c43a1a 779995d5 7b6bd784 e927bb74 5e7cb217 7af2ac15 93392ccf 07e847cf 86318b70 d9d33105 809693dd 000003b8 00000000
M_2^*	aa1435ee f3646663 c339343a 52c43a1a 779995d5 7b6bd784 e9a7bb75 5e7cb217 7af2ac15 93392ccf 07e847cf 86318b70 d9d33105 809693dd 000003b8 00000000
ΔM_2	00000201 00000000 00000000 00000000 00000000 00000000 00800001 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
H_2	92dd7ef7 b1f15ee4 b3e6a250 9db2131b
H_2^*	929d5ef7 b1f15ee4 b3e6a250 bdb21b5f
ΔH_2	00402000 00000000 00000000 20000844

5.2 Non-Randomness for the Hash Function

In this section, we show non-random properties for 48 steps (3 rounds) of the hash function. It is based on the differential q-multicollision distinguisher introduced by Biryukov et al. [1] and the characteristic for 44 steps which is extended to 48 steps (see Table 11).

Differential q-multicollisions were introduced by Biryukov et al. in the cryptanalysis of the block cipher AES-256 [1]. Note that in [1] the attack is described for a block cipher. However, it can be easily adapted for a hash function. Below we repeat the basic definition and lemma, we need for the attack on RIPEMD-128.

Definition 1. *A set of one difference and q inputs*

$$\{\Delta M; (M^1), (M^2), \cdots, (M^q)\}$$

is called a differential q-multicollision for $h(\cdot)$ *if*

$$\begin{aligned} h(M^1) \boxminus h(M^1 \boxminus \Delta M) &= h(M^2) \boxminus h(M^2 \boxminus \Delta M) \\ &= \cdots = h(M^q) \boxminus h(M^q \boxminus \Delta M). \end{aligned}$$

The complexity of the generic attack is measured in the number of queries.

Lemma 1. *To construct a differential q-multicollision for an ideal has function with an n-bit output an adversary needs at least*

$$O(q \cdot 2^{\frac{q-1}{q+1} \cdot n})$$

queries on the average for small q.

The proof for Lemma 1 works similar as in [1] for an ideal cipher. Finally, based on the characteristic given in Table 11 we construct a differential q-multicollision to show non-random properties for RIPEMD-128 reduced to 48 steps. The attack has a complexity of about $4 \cdot 2^{68}$ while the generic attack has a complexity of about 2^{76}.

5.3 Collisions for the Compression Function

When attacking the compression function an adversary has additional the possibility to inject difference in the chaining input. Using this additional freedom and the same techniques as for the collision attack on the RIPEMD-128 hash function (see Section 4), we can construct a collision for the compression function of RIPEMD-128 reduced to 48 steps. In Table 12 the differential characteristic is shown, resulting in a practical collision for 48 steps of the compression function with a complexity of 2^{40}. The example is given in Table 7.

6 Conclusions and Future Work

In this work, we have presented new results on the ISO/IEC standard RIPEMD-128, a dual-stream hash function where the message permutation and rotation values are different in the two streams. More specifically, we have presented a collision attack on reduced RIPEMD-128 and get practical collisions for 38 steps of the hash function with a complexity of about 2^{14}. Furthermore, our attack can be extended to near-collisions on 44 steps with complexity 2^{32} and a theoretical distinguisher on the hash function for 48 steps (3 out of 4 rounds) with complexity 2^{70}. Furthermore, we present practical collisions for the RIPEMD-128 compression function, also reduced to 48 steps with complexity 2^{40}.

Apart from these new results, we have outlined a strategy to analyze ARX-based dual-stream hash functions more efficiently. More precisely, we have shown how to automate the most

Table 7. Free-start collision for 48 steps of RIPEMD-128.

H_0	5a1d2fbd cd6d40c7 128dd546 900e0e65
H_0^*	5a1927bd edad5cc7 128dd542 900e0e65
ΔH_0	00040800 20c01c00 00000004 00000000
M_1	06083719 9ae0b19b 7ffae1ec 637041ad 28d722d7 fa0082c3 5e78f84e 416ee5e7 faf2b4fc 56738a9f 363c6155 cc7d7ae3 0cb5fc95 b362a16f 6cac81a9 cc11fedd
M_1^*	06083719 9ae0b19b 7ffae1ec 637041ad 28d722d7 fa0082c3 5e78f84e 416ee5e7 faf2b4fc 56738a9f 363c6155 cc7d7ae3 0cb5fc95 b362a16f 6cac81a9 cc11fedd
ΔM_1	00000200 00000000 00000000 00000000 00000000 00000000 00000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
H_1	e6428c57 a9f1f589 fc045baf a9cdbc1f
H_1^*	e6428c57 a9f1f589 fc045baf a9cdbc1f
ΔH_1	00000000 00000000 00000000 00000000

difficult parts of an attack involving more than one stream: finding a differential characteristic and performing message modification in the first round. In particular, message modification had to be hand-tuned or was omitted in previous attacks on ARX-based hash functions. What remains for an attacker is to determine a good starting point (possibly using tools from coding theory) and to assist the tools in the order of guessing words or parts of the state, to improve the overall complexity.

Ideally, these tools can immediately be applied to more complicated hash functions. However, the obtained results depend mainly on the choice of the starting point for the nonlinear tool. If no good starting point can be found or the search space is too large, no attack can be obtained. Future work is to analyze also other, stronger dual-stream hash functions like RIPEMD-160. Furthermore, the tools and techniques used in this paper can also be applied to other ARX-based hash functions, where more than one state word is updated using a single message word. Examples are SHA-2 or the SHA-3 candidates Blake and Skein.

Acknowledgments

This work was supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007), by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy) and by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II. In addition, this work was supported by the Research Fund KU Leuven, OT/08/027 and by the Austrian Science Fund (FWF, project P21936).

References

1. Biryukov, A., Khovratovich, D., Nikolić, I.: Distinguisher and Related-Key Attack on the Full AES-256. In: Halevi, S. (ed.) CRYPTO. LNCS, vol. 5677, pp. 231–249. Springer (2009)
2. Brier, E., Khazaei, S., Meier, W., Peyrin, T.: Linearization Framework for Collision Attacks: Application to CubeHash and MD6. In: Matsui, M. (ed.) ASIACRYPT. LNCS, vol. 5912, pp. 560–577. Springer (2009)
3. Damgård, I.: A Design Principle for Hash Functions. In: Brassard, G. (ed.) CRYPTO. LNCS, vol. 435, pp. 416–427. Springer (1989)
4. De Cannière, C., Rechberger, C.: Finding SHA-1 Characteristics: General Results and Applications. In: Lai, X., Chen, K. (eds.) ASIACRYPT. LNCS, vol. 4284, pp. 1–20. Springer (2006)
5. Dobbertin, H.: RIPEMD with Two-Round Compress Function is Not Collision-Free. J. Cryptology 10(1), 51–70 (1997)
6. Dobbertin, H., Bosselaers, A., Preneel, B.: RIPEMD-160: A Strengthened Version of RIPEMD. In: Gollmann, D. (ed.) FSE. LNCS, vol. 1039, pp. 71–82. Springer (1996)
7. Indesteege, S., Preneel, B.: Practical Collisions for EnRUPT. In: Dunkelman, O. (ed.) FSE. LNCS, vol. 5665, pp. 246–259. Springer (2009)
8. Joux, A., Peyrin, T.: Hash Functions and the (Amplified) Boomerang Attack. In: Menezes, A. (ed.) CRYPTO. LNCS, vol. 4622, pp. 244–263. Springer (2007)

9. Klíma, V.: Tunnels in Hash Functions: MD5 Collisions Within a Minute. IACR Cryptology ePrint Archive 2006, 105 (2006)
10. Mendel, F., Nad, T., Schläffer, M.: Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT. LNCS, Springer (2011), to appear
11. Mendel, F., Pramstaller, N., Rechberger, C., Rijmen, V.: On the Collision Resistance of RIPEMD-160. In: Katsikas, S.K., Lopez, J., Backes, M., Gritzalis, S., Preneel, B. (eds.) ISC. LNCS, vol. 4176, pp. 101–116. Springer (2006)
12. Merkle, R.C.: One Way Hash Functions and DES. In: Brassard, G. (ed.) CRYPTO. LNCS, vol. 435, pp. 428–446. Springer (1989)
13. Ohtahara, C., Sasaki, Y., Shimoyama, T.: Preimage Attacks on Step-Reduced RIPEMD-128 and RIPEMD-160. In: Lai, X., Yung, M., Lin, D. (eds.) Inscrypt. LNCS, vol. 6584, pp. 169–186. Springer (2010)
14. Pramstaller, N., Rechberger, C., Rijmen, V.: Exploiting Coding Theory for Collision Attacks on SHA-1. In: Smart, N.P. (ed.) IMA Int. Conf. LNCS, vol. 3796, pp. 78–95. Springer (2005)
15. Sugita, M., Kawazoe, M., Imai, H.: Gröbner Basis Based Cryptanalysis of SHA-1. IACR Cryptology ePrint Archive 2006, 98 (2006)
16. Wang, L., Sasaki, Y., Komatsubara, W., Ohta, K., Sakiyama, K.: (Second) Preimage Attacks on Step-Reduced RIPEMD/RIPEMD-128 with a New Local-Collision Approach. In: Kiayias, A. (ed.) CT-RSA. LNCS, vol. 6558, pp. 197–212. Springer (2011)
17. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the Hash Functions MD4 and RIPEMD. In: Cramer, R. (ed.) EUROCRYPT. LNCS, vol. 3494, pp. 1–18. Springer (2005)
18. Wang, X., Yao, A., Yao, F.: New Collision Search for SHA-1. Presented at rump session of CRYPTO (2005)
19. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) CRYPTO. LNCS, vol. 3621, pp. 17–36. Springer (2005)
20. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer, R. (ed.) EUROCRYPT. LNCS, vol. 3494, pp. 19–35. Springer (2005)

A Differential Characteristics and Conditions

Table 8. Starting point for a collision on 38 steps of RIPEMD-128.

i	∇B_i	$\nabla B'_i$	∇m_i
-4	--------------------------------		
-3	--------------------------------		
-2	--------------------------------		
-1	--------------------------------		
0	????????????????????????????????	--------------------------------	????????????????????????????????
1	????????????????????????????????	--------------------------------	--------------------------------
2	????????????????????????????????	--------------------------------	--------------------------------
3	--------------------------------	????????????????????????????????	--------------------------------
4	--------------------------------	????????????????????????????????	--------------------------------
5	--------------------------------	????????????????????????????????	--------------------------------
6	--------------------------------	????????????????????????????????	????????????????????????????????
7	--------------------------------	????????????????????????????????	--------------------------------
8	--------------------------------	????????????????????????????????	--------------------------------
9	--------------------------------	????????????????????????????????	--------------------------------
10	--------------------------------	????????????????????????????????	--------------------------------
11	--------------------------------	????????????????????????????????	--------------------------------
12	--------------------------------	????????????????????????????????	--------------------------------
13	--------------------------------	????????????????????????????????	--------------------------------
14	--------------------------------	????????????????????????????????	--------------------------------
15	--------------------------------	????????????????????????????????	--------------------------------
16	--------------------------------	????????????????????????????????	
17	--------------------------------	--------------------------------	
18	--------------------------------	--------------------------------	
19	--------------------------------	--------------------------------	
20	--------------------------------	--------------------------------	
21	????????????????????????????????	--------------------------------	
22	--------------------------------	--------------------------------	
23	--------------------------------	--------------------------------	
24	--------------------------------	--------------------------------	
25	--------------------------------	--------------------------------	
26	--------------------------------	--------------------------------	
27	--------------------------------	--------------------------------	
28	--------------------------------	--------------------------------	
29	--------------------------------	--------------------------------	
30	--------------------------------	--------------------------------	
31	--------------------------------	--------------------------------	
32	--------------------------------	--------------------------------	
33	--------------------------------	--------------------------------	
34	--------------------------------	--------------------------------	
35	--------------------------------	--------------------------------	
36	--------------------------------	--------------------------------	
37	--------------------------------	--------------------------------	

Table 9. Starting point for a collision on 38 steps of RIPEMD-128 after characteristic for the left stream is fixed. Bits with gray background have one additional conditions.

i	∇B_i	$\nabla B'_i$	∇m_i
-4	--------------------------------		
-3	--------------------------------		
-2	--------------------------------		
-1	--------------------------------		
0	-------unnnnunnnnnnnn-----------	--------------------------------	----------------------u--------u
1	------n--------------nuuuunnnnnn	--------------------------------	--------------------------------
2	------unnunnnnnnnnnnnnnnnnnnnnnn	--------------------------------	--------------------------------
3	--------------------------------	-----------u--------u-----------	--------------------------------
4	--------------------------------	????????-----------?????????????	--------------------------------
5	--------------------------------	????????????????????????????????	--------------------------------
6	--------------------------------	????????????????????????????????	--------n----------------------n
7	--------------------------------	????????????????????????????????	--------------------------------
8	--------------------------------	????????????????????????????????	--------------------------------
9	--------------------------------	????????????????????????????????	--------------------------------
10	--------------------------------	????????????????????????????????	--------------------------------
11	--------------------------------	????????????????????????????????	--------------------------------
12	--------------------------------	????????????????????????????????	--------------------------------
13	--------------------------------	????????????????????????????????	--------------------------------
14	--------------------------------	????????????????????????????????	--------------------------------
15	--------------------------------	----------------------x--------x	--------------------------------
16	--------------------------------	----------------------n--------n	
17	--------------------------------	--------------------------------	
18	--------------------------------	--------------------------------	
19	--------------------------------	--------------------------------	
20	--------------------------------	--------------------------------	
21	----------------------n--------n	--------------------------------	
22	----------------------0--------0	--------------------------------	
23	----------------------1--------1	--------------------------------	
24	--------------------------------	--------------------------------	
25	--------------------------------	--------------------------------	
26	--------------------------------	--------------------------------	
27	--------------------------------	--------------------------------	
28	--------------------------------	--------------------------------	
29	--------------------------------	--------------------------------	
30	--------------------------------	--------------------------------	
31	--------------------------------	--------------------------------	
32	--------------------------------	--------------------------------	
33	--------------------------------	--------------------------------	
34	--------------------------------	--------------------------------	
35	--------------------------------	--------------------------------	
36	--------------------------------	--------------------------------	
37	--------------------------------	--------------------------------	

Table 10. Characteristic for a collision on 38 steps of RIPEMD-128. Bits with gray background have one additional conditions.

i	∇B_i	$\nabla B'_i$	∇m_i
-4	--------------------------------		
-3	--------------------------------		
-2	--------------------------------		
-1	--------------------------------		
0	-------unnnnunnnnnnnn-----------	--------------------------------	----------------------u--------u
1	------n--------------nuuuunnnnnn	-----------0--------0-----------	--0-----------------------------
2	------unnunnnnnnnnnnnnnnnnnnnnnn	-----------0--------0-----------	--------------------------------
3	--------------------------------	--0100-----u--------u----0110---	--------------------------------
4	--------------------------------	--1101----1-1-------1----1111---	---0----------------------------
5	--------------------------------	--unnn00--1-1-------1----unnn-00	--------------------------------
6	--------------------------------	--000010--n-u---00--n----0111-10	--------n----------------------n
7	--------------------------------	001nuuuu--0-----11111----1001-nu	--------------------------------
8	--------------------------------	110100----1-----un11n-------u---	--------------------------------
9	--------------------------------	un1n00----------1-unn---1---1---	--------------------------------
10	--------------------------------	--n0u1----------0-10000-----1---	--------------------------------
11	--------------------------------	--0nuu------------01n11-----n---	--------------------------------
12	--------------------------------	--110--------------nuuu---------	--------------------------------
13	--------------------------------	---01--------------11-1---------	--------------------------------
14	--------------------------------	-------------------00-1--------0	--------------------------------
15	--------------------------------	----------------------n--------n	--------------------------------
16	--------------------------------	----------------------n--------n	
17	--------------------------------	----------------------0--------0	
18	--------------------------------	--------------------------------	
19	--------------------------------	--------------------------------	
20	--------------------------------	--------------------------------	
21	----------------------n--------n	--------------------------------	
22	----------------------0--------0	--------------------------------	
23	----------------------1--------1	--------------------------------	
24	--------------------------------	--------------------------------	
25	--------------------------------	--------------------------------	
26	--------------------------------	--------------------------------	
27	--------------------------------	--------------------------------	
28	--------------------------------	--------------------------------	
29	--------------------------------	--------------------------------	
30	--------------------------------	--------------------------------	
31	--------------------------------	--------------------------------	
32	--------------------------------	--------------------------------	
33	--------------------------------	--------------------------------	
34	--------------------------------	--------------------------------	
35	--------------------------------	--------------------------------	
36	--------------------------------	--------------------------------	
37	--------------------------------	--------------------------------	

Table 11. Starting point for all other attacks on the hash function after characteristic for the left stream is fixed. Bits with gray background have one additional conditions.

i	∇B_i	$\nabla B'_i$	∇m_i
-4	--------------------------------		
-3	--------------------------------		
-2	--------------------------------		
-1	--------------------------------		
0	-------unnnnunnnnnnnn-----------	--------------------------------	----------------------u--------u
1	------n--------------nuuuunnnnnn	--------------------------------	--------------------------------
2	------unnunnnnnnnnnnnnnnnnnnnnnn	--------------------------------	--------------------------------
3	--------------------------------	-----------u--------u-----------	--------------------------------
4	--------------------------------	????????-----------?????????????	--------------------------------
5	--------------------------------	????????????????????????????????	--------------------------------
6	--------------------------------	????????????????????????????????	--------n----------------------n
7	--------------------------------	????????????????????????????????	--------------------------------
8	--------------------------------	????????????????????????????????	--------------------------------
9	--------------------------------	????????????????????????????????	--------------------------------
10	--------------------------------	????????????????????????????????	--------------------------------
11	--------------------------------	????????????????????????????????	--------------------------------
12	--------------------------------	????????????????????????????????	--------------------------------
13	--------------------------------	????????????????????????????????	10000000------------------------
14	--------------------------------	????????????????????????????????	00000000000000000000001110111000
15	--------------------------------	----------------------x--------x	00000000000000000000000000000000
16	--------------------------------	----------------------n--------n	
17	--------------------------------	--------------------------------	
18	--------------------------------	--------------------------------	
19	--------------------------------	--------------------------------	
20	--------------------------------	--------------------------------	
21	----------------------n--------n	--------------------------------	
22	----------------------0--------0	--------------------------------	
23	----------------------1--------1	--------------------------------	
24	--------------------------------	--------------------------------	
25	--------------------------------	--------------------------------	
26	--------------------------------	--------------------------------	
27	--------------------------------	--------------------------------	
28	--------------------------------	--------------------------------	
29	--------------------------------	--------------------------------	
30	--------------------------------	--------------------------------	
31	--------------------------------	--------------------------------	
32	--------------------------------	--------------------------------	
33	--------------------------------	--------------------------------	
34	--------------------------------	--------------------------------	
35	--------------------------------	--------------------------------	
36	--------------------------------	--------------------------------	
37	--------------------------------	--------------------------------	
38	--------------------------------	--n----------------------n------	
39	--------------------------------	--0----------------------0------	
40	--------------------------------	--1----------------------1------	
41	---------0--------0-------------	--------------------------------	
42	--0------u--------u------0------	--------------------n--------n--	
43	--n------1--------1------n------	--------------------0--------0--	
44	--1---0------------------1---0--	--------------------1--------1--	
45	--1---n----0------0--------0-n--	---------u--------u-------------	
46	--u---1----u------n--------u-1--	---------0---n----0---n---------	
47	--------------------n-------n---	--------------------------------	

Table 12. Characteristic for a free-start collision for 48 steps of RIPEMD-128 compression function. Bits with gray background have one additional conditions.

i	∇B_i	$\nabla B'_i$	∇m_i
-4	-------------u------u-----------		
-3	--0-----00---------011-------1--		
-2	-00-00--10-011-----101---10--u--		
-1	-1n-11--nu-011-----nnn---10--1-1		
0	un1nnnnn00nu01---un101nuunnnn0n0	010-1u-----nuu--1-101101-010-1-1	-----------0----------u--------1
1	nnnnnnnnnnnnnnnnn--010---01--n-u	1nn-n1-----n00-01-110110-n11-00u	-------------------100----------
2	--0-----10--unnnnnnnnnnnnnnnnnnn	u1n--1000-1n10-0n-un-nnn-unu---1	-----------11--------0---1------
3	--1-----00---------110---10----0	0n0--n1111-01u-u---01uu--1---nu1	-------------------------------1
4	-------------------110---10----1	u11---unn0u11111---001---10--0nu	------------------------110--111
5	--------------------------------	u1----011uuun010-0-1nu----0-01u1	--------------------------------
6	--------------------------------	0u----10u11uu0n--1-n----10u---0u	-------------------------------n
7	--------------------------------	00------n1n0101--n-0----111---11	-------------------0------------
8	--------------------------------	-1------0unnnnn0000u0000un1----0	--------------------------------
9	--------------------------------	--------110---n11u10111-10n00---	0----------1---------0----------
10	--------------------------------	---------01---0unnnnnnnn1u011---	--------------------------------
11	--------------------------------	--------------1011-----nuuuuu---	-------0------------------------
12	--------------------------------	---------------100-----010------	--------------------------------
13	--------------------------------	-----------------------101------	------------------------------11
14	--------------------------------	----------------------0---------	--------------------------------
15	--------------------------------	----------------------n---------	------------------------1-------
16	--------------------------------	----------------------n---------	
17	--------------------------------	----------------------0---------	
18	--------------------------------	--------------------------------	
19	--------------------------------	--------------------------------	
20	--------------------------------	--------------------------------	
21	----------------------n---------	--------------------------------	
22	----------------------0---------	--------------------------------	
23	----------------------1---------	--------------------------------	
24	--------------------------------	--------------------------------	
25	--------------------------------	--------------------------------	
26	--------------------------------	--------------------------------	
27	--------------------------------	--------------------------------	
28	--------------------------------	--------------------------------	
29	--------------------------------	--------------------------------	
30	--------------------------------	--------------------------------	
31	--------------------------------	--------------------------------	
32	--------------------------------	--------------------------------	
33	--------------------------------	--------------------------------	
34	--------------------------------	--------------------------------	
35	--------------------------------	--------------------------------	
36	--------------------------------	--------------------------------	
37	--------------------------------	--------------------------------	
38	--------------------------------	-------------------------n------	
39	--------------------------------	-------------------------0------	
40	--------------------------------	-------------------------1------	
41	---------0----------------------	--------------------------------	
42	---------u---------------0------	--------------------n-----------	
43	---------1---------------n--0---	--------------------0-----------	
44	---------1---------------1--10--	--------------------1-----------	
45	--0---------------0--0---1--nu--	---------u----------------------	
46	--u---------------u--n------11--	---------0---n------------------	
47	-------------------nu-----------	--------------------------------	

Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 family

Dmitry Khovratovich[1] and Christian Rechberger[2] and Alexandra Savelieva[3]

[1]Microsoft Research Redmond, USA
[2]DTU MAT, Denmark
[3]National Research University Higher School of Economics, Russia

Abstract. We present a new concept of biclique as a tool for preimage attacks, which employs many powerful techniques from differential cryptanalysis of block ciphers and hash functions.
The new tool has proved to be widely applicable by inspiring many authors to publish new results of the full versions of AES, KASUMI, IDEA, and Square. In this paper, we demonstrate how our concept results in the first cryptanalysis of the round-reduced Skein hash function, and describe an attack on the SHA-2 hash function with more rounds than before.

Keywords: SHA-2, SHA-256, SHA-512, Skein, SHA-3, hash function, meet-in-the-middle attack, splice-and-cut, preimage attack, initial structure, biclique.

1 Introduction

In recent years, tremendous progress has been made in developing preimage attacks on hash functions. Major breakthrough happened in 2008 when the so-called *splice-and-cut* framework was introduced. Its applications to MD4 and MD5 [2, 25], and later to Tiger [11] brought amazing results. Internal properties of message schedule appeared to be limiting application of this framework to SHA-x family [1, 3]. However, the new concept of *biclique* that we are going to introduce in this paper allows to mitigate such obstacles as demonstrated by recent results on preimage attacks and, surprisingly, key recovery attacks on block ciphers.

This paper presents the first work on the concept of biclique cryptanalysis. We concentrate on the hash function setting alone, and focus on new definitions and algorithms. As applications, we present an attack on the Skein hash function (the only one existing so far) and briefly describe attacks on SHA-2 hash functions.

Splice-and-cut framework and its progress. Both splice-and-cut and meet-in-the-middle attacks exploit the property that a part of a primitive does not make use of particular key/message bits. If the property holds, the computation of this part remains the same if we flip those bits in the other part of a primitive. Assume the property is mutual, i.e. such bits can be found for both parts (also called *chunks*). Then a cryptanalyst prepares a set of independent computations for all possible values of those bits (called *neutral bits*) and subsequently checks for the match in the middle. The gain of the attack is proportional to the number of neutral bits.

Sasaki and Aoki observed [2, 25] that compression functions with permutation-based message schedule are vulnerable to this kind of attack as chunks can be long. They also proposed various improvements. For example, since the number of computations to match decreases together with the number of neutral bits, the match can be performed on a small part of the state. In turn, the matching bits depend on fewer message bits, which in fact leads to an even larger number of neutral bits and a reduction in complexity.

The most interesting trick, however, is a so called initial structure [3, 26]. The initial structure can be informally defined as an overlapping of chunks, where neutral bits, although formally belonging to both chunks, are involved in the computation of the proper chunk only. Concrete examples of the initial structure are much more sophisticated and hard to generalize. While the other improvements of splice-and-cut framework seem exhausted already, the concept of initial structure has a large potential and few boundaries.

Our contributions. We replace the idea of the initial structure with a more formal and general concept of biclique, which provides us with several layers of understanding and applications. We derive a system of functional equations linking internal states several rounds apart. Then we show that it is equivalent to a system of differentials, so the full structure of states can be built out of a structure of trails. These structures are two sets of internal states with each state having a relation with all states in another set. In terms of graph theory, these structures are referred to as *bicliques*. A differential view, that builds up on this formalism, allows us to apply numerous tools from collision search and enhanced differential attacks, from message modifications to local collisions. We propose several algorithms constructing these bicliques, which are generic and flexible.

Our first and simple example of biclique application is the hash function and the SHA-3 finalist Skein-512, which lacks any attacks in the hash setting. We develop an attack on 22 rounds of Skein-512, which is comparable to the best attacks on the compression function that survived the last tweak. Our attack on the compression function of Skein-512 utilizes many more degrees of freedom as we control the full input, and thus results in a 37-round attack.

Our second group of applications is the SHA-2 family. Enhanced with the differential analysis, we heavily use differential trails in SHA-2, message modification techniques from SHA-1 and SHA-0, and trail backtracking techniques from RadioGatun, Grindahl, SHA-1, and many others. As a result, we build attacks on 45-round SHA-256 and 50-round SHA-512, both the best attacks in the hash mode. Regarding the compression functions, we penetrate up to seven more rounds, thus reaching 52 rounds and violating the security of about 80% of SHA-256. We summarize and compare our findings in Table 1.

Throughout the paper, and in line with most cryptanalytic work, we seek to obtain results on as many rounds as possible. For the case of full 72-round Skein-512, we also study ways to use the MITM approach to speed-up brute force search and mention the result in Table 1 as well.

Reference	Target	Steps	Complexity			Memory (words)
			Pseudo-preimage	Second Preimage	Preimage	
Section 4	Skein-512	22	2^{508}	2^{511}	-	2^{6}
Section 6	Skein-512	37	$2^{511.2}$	-	-	2^{64}
Appendix E.1	Skein-512	72	-	$2^{511.71}$	-	negl.
[1, 11]	SHA-256	43	$2^{251.9}$	$2^{254.9}$	$2^{254.9}$	2^{6}
Section 5	SHA-256	45	2^{253}	$2^{255.5}$	$2^{255.5}$	2^{6}
Section 6	SHA-256	52	2^{255}	-	-	2^{6}
[1, 11]	SHA-512	46	2^{509}	$2^{511.5}$	$2^{511.5}$	2^{6}
Section 5	SHA-512	50	2^{509}	$2^{511.5}$	$2^{511.5}$	2^{4}
Section 6	SHA-512	57	2^{511}	-	-	2^{6}

Table 1. New (second) preimage attacks on Skein-512 and the SHA-2 family.

Other applications of biclique cryptanalysis

Soon after the initial circulation of this work, the idea of biclique cryptanalysis found other applications. Among them we mention key recovery faster than brute force for AES-128, AES-192, and AES-256 by Bogdanov et al. [8]. Cryptanalysis of AES employed algorithms for biclique construction which are partly covered in Section 3. In this context we also mention new and improved results on Kasumi by Jia et al. [14] and IDEA by Biham et al. [7] as well as more results announced both publicly [12, 19, 30] and privately.

2 Bicliques

In this section we introduce splice-and-cut preimage attacks with bicliques. We consider hash functions with block cipher based compression functions $H = E_N(X) \oplus X$, where E is the block cipher keyed with parameter N (notation $\xrightarrow{N}$ and $\xleftarrow{N}$ will be used for E computed in forward and backward direction respectively). Depending on the design, parameters (N, X) will be: (M, CV) for the most popular Davies-Meyer mode, (CV, M) for Matyas-Meyer-Oseas mode, where CV is the chaining variable and M is the message.

Let f be a sub-cipher of E, and $\mathcal{N} = \{N[i,j]\}$ be a group of parameters for f. Then a *biclique of dimension* d over f for $\mathcal{N}$ is a pair of sets $\{Q_i\}$ and $\{P_j\}$ of 2^d states each such that

$$Q_i \xrightarrow[f]{N[i,j]} P_j. \tag{1}$$

A biclique is used in the preimage search as follows (Figure 1). First, we note that if $N[i,j]$ is a preimage, then

$$E: \quad X \xrightarrow{N[i,j]} Q_i \xrightarrow[f]{N[i,j]} P_j \xrightarrow{N[i,j]} H.$$

An adversary selects a variable v outside of f (w.l.o.g. between P_j and H) and checks, for appropriate choices of sub-ciphers g_1 and g_2, if

$$\exists i,j: \; P_j \xrightarrow[g_1]{N[i,j]} v \overset{?}{=} v \xleftarrow[g_2]{N[i,j]} Q_i.$$

A positive answer yields a candidate preimage. Here, to compute v from Q_i, the adversary first computes X and then derives the output of E as $X \oplus H$.

To benefit from the meet-in-the-middle framework the variable v is chosen so that g_1 and g_2 are independent of i and j, respectively:

$$P_j \xrightarrow[g_1]{N[*,j]} v \overset{?}{=} v \xleftarrow[g_2]{N[i,*]} Q_i.$$

Then the complexity of testing 2^{2d} messages for preimages is computed as follows:

$$C = 2^d(C_{g_1} + C_{g_2}) + C_{bicl} + C_{recheck},$$

where C_{bicl} is the biclique construction cost, and $C_{recheck}$ is the complexity of rechecking the remaining candidates on the full state. We explain how to amortize the biclique construction in the next section. Clearly, one needs 2^{n-2d} bicliques of dimension d to test 2^n parameters.

3 Biclique construction algorithms

Here we introduce several algorithms for the biclique construction. They differ in complexity and requirements to the dimension of a biclique and properties of the mapping f.

For most algorithms we adopt a differential view on bicliques as it allows for numerous tools from differential cryptanalysis to be employed. Consider a single mapping in Equation (1)

$$Q_0 \xrightarrow[f]{N[0,0]} P_0. \tag{2}$$

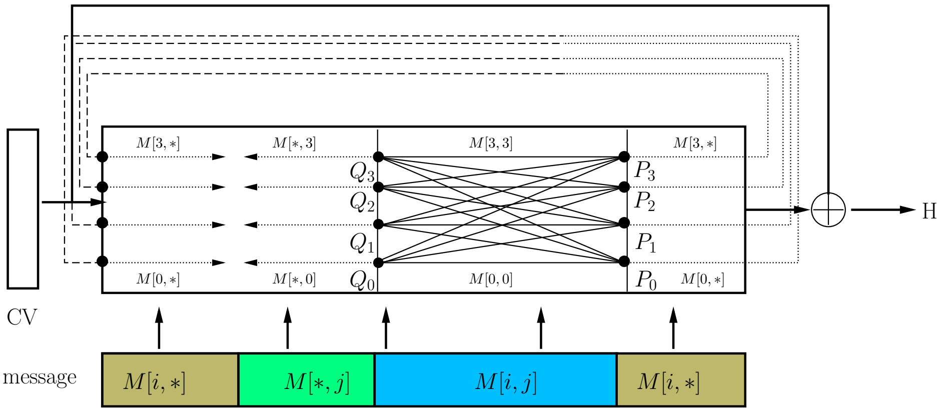


Fig. 1. Biclique of dimension 2 in the meet-in-the-middle attack on a Davies-Meyer compression function.

We call this a *basic computation*. Consider the other mappings as differentials to the basic computation:

$$\nabla_i \xrightarrow[f]{\Delta^N_{i,j}} \Delta_j, \tag{3}$$

so that

$$Q_i = Q_0 \oplus \nabla_i, \quad P_j = P_0 \oplus \Delta_j, \quad N[i,j] = N[0,0] \oplus \Delta^N_{i,j}.$$

Vice versa, if a computation (2) is a solution to 2^{2d} differentials in (3), then it is a basic computation for a biclique.

In the following algorithms we show how to reduce the number of differentials needed for a biclique, and hence construct a biclique efficiently.

Algorithm 1. Let the differences in the set $\mathcal{N}$ be defined as the following linear function:

$$\Delta^N_{i,j} = \Delta^N_j \oplus \nabla^N_i \tag{4}$$

Let us fix Q_0 and construct P_j as follows:

$$Q_0 \xrightarrow[f]{N[0,j]} P_j. \tag{5}$$

As a result, we get a set of trails:

$$0 \xrightarrow[f]{\Delta^N_j} \Delta_j. \tag{6}$$

Let us also construct Q_i out of P_0:

$$Q_i \xleftarrow[f]{N[i,0]} P_0, \tag{7}$$

and get another set of trails:

$$\nabla_i \xrightarrow[f]{\nabla^N_i} 0. \tag{8}$$

Suppose that the trails (8) do not affect active non-linear elements in the trails (6). Then Q_i are solutions to the trails (6), so we get the biclique equation:

$$Q_i \xrightarrow[f]{N[i,j]} P_j. \tag{9}$$

To estimate the complexity, assume that the computation (7) does not affect active non-linear elements in the trails (6) with probability 2^{-t}. Then the probability that 2^d such computations affect no condition is 2^{-t2^d}. Therefore, Equation (9) is satisfied with probability 2^{-t2^d}, so we need 2^{t2^d} solutions to Equation (6) to build a biclique (which is feasible for small d). This approach is used in the preimage attack on the hash function Skein-512.

For non-ARX primitives with predictable diffusion this algorithm can be easily made deterministic. For example, it is easy to construct the truncated differential trails for AES [8] and Square that do not share active non-linear components with probability 1 (Figure 2). As a result, an attack algorithm can be simply explained using a picture of trails. ■

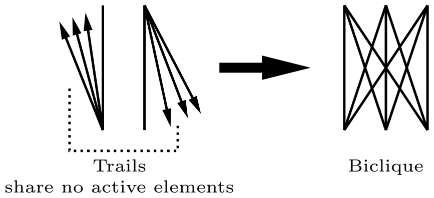


Fig. 2. Biclique out of non-interleaving trails.

Algorithm 2. (Modification of Algorithm 1 for the case when the hash function operates in Davies-Meyer mode, and we can control internal state and injections of message M within the biclique). Assume that the mapping f uses several independent parts (blocks) of message M via the message injections (like in SHA-2). Consider a message group with property (4) but do not define the messages yet. Choose a state Q_0 satisfying sufficient conditions to build sets of trails (6) and (8) that do not share active non-linear components. Then find $N[0,0]$ - a message such that the computation

$$Q_0 \xrightarrow[f]{N[0,0]} P_0$$

conforms to both sets of trails. Since the sets do not share active non-linear components, we get

$$Q_i \xrightarrow[f]{N[i,j]} P_j,$$

where $Q_i = Q_0 \oplus \nabla_i, \quad P_j = P_0 \oplus \varDelta_j$.

Since we control message injections in f, we are able to define $N[0,0]$ block by block similarly to the trail backtracking approach [5]. If the message schedule is non-linear, the differential trails (6) and (8) may depend on $N[0,0]$. Furthermore, parts of message $N[0,0]$ may remain undefined as they are not used in f. A procedure that ensures that the message $N[0,0]$ is well-defined, and the trails (6) and (8) do not contradict, was first proposed in [1] and is referred to as *message compensation.* ■

Algorithm 3. (for bicliques of dimension 1) We apply this rebound-style [20] algorithm if the mapping f is too long for differential trails with reasonable number of sufficient conditions. Then we split it into two parts f_1 and f_2 and consider two differential trails with probabilities p and q, respectively:

$$0 \xrightarrow[f_1]{\varDelta^N} \varDelta, \quad \nabla \xrightarrow[f_2]{\nabla^N} 0. \qquad (10)$$

We fix the state S between f_1 and f_2, and consider a quartet of states:

$$S,\ S \oplus \Delta,\ S \oplus \nabla,\ S \oplus \Delta \oplus \nabla.$$

Suppose that a quartet of states is a quartet in the middle of the boomerang attack, which happens with probability p^2q^2 for a random N under an approriate independency assumption. Then we derive input states Q_0, Q_1 and output states P_0, P_1, which are linked as follows (see also Figure 3):

$$\begin{aligned}
Q_0 &\xrightarrow[f_1]{N} S \xrightarrow[f_2]{N} P_0;\\
Q_0 &\xrightarrow[f_1]{N\oplus\Delta^N} S\oplus\Delta \xrightarrow[f_2]{N\oplus\Delta^N} P_1;\\
Q_1 &\xrightarrow[f_1]{N\oplus\nabla^N} S\oplus\nabla \xrightarrow[f_2]{N\oplus\nabla^N} P_0;\\
Q_1 &\xrightarrow[f_1]{N\oplus\Delta^N\oplus\nabla^N} S\oplus\Delta\oplus\nabla \xrightarrow[f_2]{N\oplus\Delta^N\oplus\nabla^N} P_1.
\end{aligned}$$

Therefore, we get a biclique, where the set of parameters $\mathcal{N}$ is defined as follows:

$$N[0,0] = N;\ N[0,1] = N \oplus \Delta^N;\ N[1,0] = N \oplus \nabla^N;\ N[1,1] = N \oplus \Delta^N \oplus \nabla^N. \blacksquare$$

We use Algorithm 2 in the attacks on SHA-256 and SHA-512, and Algorithm 3 is applied in the preimage attack on the Skein compression function. In practice, we use freedom in the internal state and in the message injection fulfill conditions in both trails with tools like message modification and auxiliary paths.

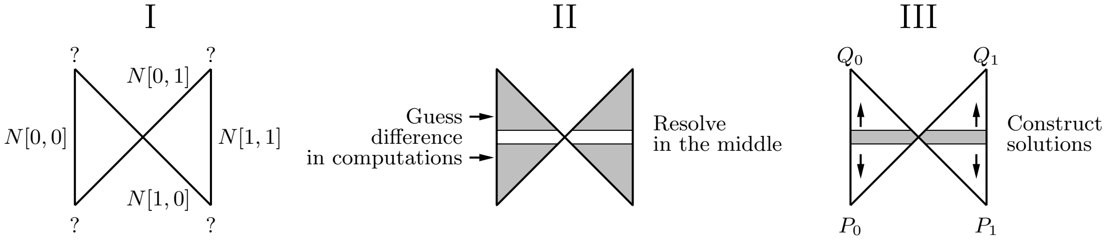


Fig. 3. Rebound-style algorithm for biclique construction.

4 Simple case: second preimage attack on Skein-512 hash

Skein [10] is a SHA-3 finalist, and hence gets a lot of cryptanalytic attention. Differential [4] and rotational cryptanalysis [17] led the authors of Skein to tweak the design twice. As a result, a rotational property, which allowed cryptanalyst to penetrate the highest number of rounds, does not exist anymore in the final-round version of Skein. Hence the best known attack are near-collisions on up to 24 rounds (rounds 20-43) of the compression function of Skein [4, 27]. Very recently near-collisions attacks on up to 32 rounds of Skein-256 were demonstrated [29].

The cryptanalysis of the Skein hash function, however, is very limited, and since the first publication of this work there has been no advance in this direction. Rotational attacks did not extend to the hash function setting, and the differential attacks were not applied in this model. In fact there is no cryptanalytic attack known on any round-reduced version of Skein at all. We subsequently give the first attack in this arguably much more relevant setting.

Since this is the first application of our method, we prefer to give the simplest example in the strongest model rather than attack the highest number of rounds. We consider the Skein-512 hash function reduced to rounds 3-24 (22-round version). In addition to using the biclique concept, one of the interesting features of our attack is that we, apparently for the first time, utilize a statistical hypothesis test to improve the matching phase instead of a direct or a symbolic (indirect) matching. Without it, less rounds could be covered with basically the same computational complexity.

4.1 Description of Skein-512

Skein-512 is based on the block cipher Threefish-512 — a 512-bit block cipher with a 512-bit key parametrized by a 128-bit tweak. Both the internal state I and the key K consist of eight 64-bit words, and the tweak T is two 64-bit words. The compression function $F(CV, T, M)$ of Skein is defined as:

$$F(CV, T, M) = E_{CV,T}(M) \oplus M,$$

where $E_{K,T}(P)$ is the Threefish cipher, CV is the previous chaining value, T is the tweak, and M is the message block. The tweak value is a function of parameters of message block M.

Threefish-512 transforms the plaintext P in 72 rounds as follows:

$$P \rightarrow \text{Add subkey } K^0 \rightarrow \text{4 rounds} \rightarrow \text{Add } K^1 \rightarrow \ldots \rightarrow \text{4 rounds} \rightarrow \text{Add } K^{18} \rightarrow C.$$

The subkey $K^s = (K^s_0, K^s_1, \ldots, K^s_7)$ is produced out of the key $K = (K[0], K[1], \ldots, K[7])$ as follows:

$$\begin{aligned} K^s_j &= K[(s+j) \bmod 9], \quad 0 \le j \le 4; & K^s_5 &= K[(s+5) \bmod 9] + T[s \bmod 3]; \\ K^s_6 &= K[(s+6) \bmod 9] + T[(s+1) \bmod 3]; & K^s_7 &= K[(s+7) \bmod 9] + s, \end{aligned}$$

where s is a round counter, $T[0]$ and $T[1]$ are tweak words, $T[2] = T[0] + T[1]$, and $K[8] = C_{240} \oplus \bigoplus_{j=0}^{7} K[j]$ with constant C_{240} optimized against rotation attacks.

One round transforms the internal state as follows. The eight words $I^0, I^1, \ldots, I^7$ are grouped into pairs and each pair is processed by a simple 128-bit function MIX. Then all the words are permuted by the operation PERM. The details of these operations are irrelevant for the high-level description, for completeness they can be found in Appendix A. We use the following notation for the internal states in round r:

$$S^{r-A} \xrightarrow{MIX} S^{r-M} \xrightarrow{PERM} S^{r-P}$$

4.2 Second preimage attack on the reduced Skein-512

We consider Skein-512 reduced to rounds 3–24. In the hash function setting we are given the message M and the tweak value T, and have to find a second preimage. We produce several pseudo-preimages (CV, M') to a call of the compression function that uses 512 bits of M and then find a valid prefix that maps the original IV to one of the chaining values that we generated. Let f map the state after round 11 to the state before round 16. We construct a biclique of dimension 3 for f following Algorithm 1 (Section 3):

1. Define $\Delta_j^N = (0, j \lll 58, j \lll 58, 0, 0, 0, 0, 0)$ and $\nabla_i^N = (0, 0, 0, i \lll 55, i \lll 55, 0, 0, 0)$.
2. Generate Q_0 and compute $P_0, P_1, \ldots, P_7$. If the trails $0 \xrightarrow[f]{\Delta_j^N} \Delta_j$ are not based on the linear difference propagation, repeat the step.
3. Compute Q_i and check if the condition on active non-linear elements is fulfilled. If so, output a biclique.

We use a differential trail that follows a linear approximation that is a variant of the 4-round differential trail, which can be obtained in a similar way to the one presented in the paper [4]. The number of active bits is given in Table 2, with further details of the trail provided Appendix A in Table 3. For the trails based on the 3-bit difference Δ_j^N we have 206 sufficient conditions in total. A computation of Q_i out of P_0 do not affect those conditions with probability $2^{-0.3}$ (verified experimentally). Therefore, for the eight states P_j the probability is $2^{-0.3 \cdot 8} \approx 2^{-3}$. We construct a 4-round biclique with complexity at most $2^{206+3} = 2^{209}$. Note that we have $1024 - 209 = 815$ degrees of freedom left.

	I^0	I^1	I^2	I^3	I^4	I^5	I^6	I^7	Conditions in the round
S^{12-A}								3	3
S^{13-A}				6	3				9
S^{14-A}	6		3			3		12	24
S^{15-A}	3	6	3	24	12	6	6	3	63
S^{15-P}	21	9	12	4	3	18	3	37	$107(messageaddition)$

Table 2. Number of active bits in the most dense Δ-trail in 4 rounds of Skein-512.

Probabilistic matching. The matching variable v consists of bits 30, 31, 53 of the word 1 after round 24. Due to carry effects, there is a small probability that those bits require the knowledge of the full message to be computed in both directions. This probability has been computed experimentally and equals 0.09. Therefore, a matching pair of computations yields a pseudo-preimage with probability $2^{-509} \cdot 0.09 \approx 2^{-509.1}$, and we need to use $2^{506.1}$ bicliques for this purpose.

1. Build a biclique of dimension 3 in rounds 12-15 with key additions (key addition + 4 rounds + key addition).
2. Compute forward chunk in rounds 16-19, backward chunks in rounds 8-11, and bits $I^1_{30,31,53}$ of the the state S^{24-P} in both directions in the partial matching procedure.
3. Check for the match in these bits, produce 2^3 key candidates, which get reduced to $2^{2.9}$ due to the type-I error [22], i.e. a false positive. Check them for the match on the full state.
4. Generate a new biclique out of the first one by change of key bits.
5. Repeat steps 2-5 $2^{507.5}$ times and generate $2^{507.5-509+2.9} = 2^{1.6}$ full pseudo-preimages.
6. Match one of the pseudo-preimages with the real CV_0.

On step 3. We have checked experimentally that the matching bits can be computed from both chunks independently with probability 0.91, so with probability $2^{-0.1}$ we have a type-I error, and the candidate is discarded. Insisting on probability 1, as done in earlier work, would have lead to a redesign of the attack for a smaller number of rounds.

Complexity. The biclique construction cost can be made negligible, since many bicliques can be produced out of one. Indeed, we are able to flip most of the bits in the message so that the biclique computation between the message injections remain unaffected, and only output states are changed. Every new biclique needs half of rounds 8-11 and 16-19 recomputing, and half of rounds 3-5 and 21-24 computing to derive the value of the matching variable. Hence each biclique tests 2^6 preimage candidates at cost of $(2+2+1.5)\cdot 8+(2+2+2)\cdot 8 = 92$ rounds of 22-round Skein, or $2^{2.3}$ calls of the compression function, taking a recheck into account. As a result, a full pseudo-preimage is found with complexity $2^{508.4}$. We need $2^{1.6} \approx 3$ pseudo-preimages to match one of $2^{510.4}$ prefixes, so the total complexity is $2^{511.2}$.

5 Preimage attacks on the SHA-2 hash functions

The SHA-2 family is the object of very intensive cryptanalysis in the world of hash functions. In contrast to its predecessors, collision attacks are no longer the major threat with the best attack on 24 rounds of the hash function [13, 24]. So far the best attacks on the SHA-2 family are preimage attacks on the hash function in the splice-and-cut framework [1] and a boomerang distinguisher that is only applicable for the compression function [18]. We demonstrate that our concept of biclique adds two rounds to the attack on SHA-256, four rounds to the attack on SHA-512, and many more when attacking the compression functions. The number of rounds we obtain for the compression function setting is in both cases comparable to [18], the later however does not allow extension to the hash function nor does it violate any “traditional” security requirement.

Details of SHA-2 hash functions specification [23] can be found in Appendix B. Since message schedule is nonlinear, the number of attacked rounds depends significantly on the position of the biclique. We apply the following reasoning:

- The message injections in rounds 14-15 are partially determined by the padding rules;
- Freedom in the message reduces the biclique amortized cost;
- Chunks do not bypass the feedforward operation due to high nonlinearity of the message schedule;
- There exists a 6-round trail with few conditions easy to use as a ∇-differential.
- Chunks do not have maximal length, otherwise the biclique trail becomes too dense.

SHA-256. Taking these issues into account, we base our attack on a 6-round biclique in rounds 17-22. The full layout is provided in Table 5. The biclique is constructed with Algorithm 2, Section 3.

SHA-512. Our attack on SHA-512 does not fix all the 129 padding bits of the last block. This approach still allows to generated short second preimages by using the first preimage to invest the last block that includes the padding and perform the preimage attack in the last chaining input as the target.

For a preimage attack without a first preimage, expandable messages as e.g. described in [16] can be used. This adds no noticeable cost as the effort for this is only slightly above the birthday bound. In addition, the compression function attack needs to fulfill the following two properties:

Firstly, the end of the message (before the length encoding, i.e., the LSB of W^{13}) has to be ’1’. Secondly, the length needs to be an exact multiple of the block length, i.e., fix the last nine bits of W^{15} to ”1101111111” (895). In total eleven bits would need to be fixed for this. In the further text we show how to fulfill these conditions.

The biclique is constructed by an algorithm similar to the attack on SHA-256 (Algorithm 2, Section 3).

6 Attacks on the compression functions: SHA-2 and Skein

6.1 Preimage attacks on the Skein compression functions

In this section we provide an attack on the 37-round Skein-512 compression function. In the compression function setting we control the tweak value, which gives us additional freedom both in chunks and the construction of the biclique.

The attack parameters are listed in Table 4 in the Appendix. We build a biclique in rounds 24-31, and apply the attack to rounds 2-38, i.e., to the 37-round compression function.

Bicliques are constructed by Algorithm 3 (Section 3). We use two differential trails: based on Δ^M (Δ-trail) for rounds 16-19 (including key addition in round 19) and based on ∇^M (∇-trail) for rounds 20-23. The differential trails are based on the evolution of a single difference in the linearized Skein. The Δ-trail has probability 2^{-52}. The ∇-trail has probability 2^{-29}.

The biclique is constructed as follows. First, we restrict to rounds 19-20, where the compression function can be split into two independent 256-bit transformations. A simple approach with table lookups gives a solution to restricted trails with amortized cost 1 (more efficient methods certainly exist). Then we extend this solution to an 8-round biclique by the bits of K^5. We use K^5 in the messagemodification-like process and adjust the sufficient conditions in rounds 16-23. We have 221 degrees of freedom for that (computed on a PC). As many as 96 bits of freedom do not affect the biclique at all and are used to reduce the amortized cost to only a single round.

In the matching part we recompute 29 rounds per biclique. However, a single key bit flip affects only half of rounds 12-15 and 24-27, and also we need to compute only a half of rounds 2-5 and 35-38. In total, we recompute 42 rounds, or $2^{1.2}$ calls of the compression function per structure, and get 2 candidates matching on one bit. The full preimage is found with complexity $2^{511.2}$.

6.2 Preimage attacks on the SHA-2 compression functions

In this section we provide short description of attacks on the SHA-2 compression functions. As long as we do not attack the full hash function, the preimage attack on the compression function is relevant if it is faster than 2^n, though not all these attacks are convertible to the hash function attacks. As a result, we can apply the splice-and-cut attack with the minimum gain to squeeze out the maximum number of rounds. This implies that we consider bicliques of dimension 1. In differential terms, we consider single bit differences Δ_1^M and ∇_1^M. As a result, we get sparse trails with few conditions, and may extend them for more rounds.

- Build 11-round biclique out of a 11-round ∇-trail in rounds 17-27 (SHA-256) and 21-31 (SHA-512). The trail is a variant of the trail in Table 6 that starts with one-bit difference.
- Construct message words in the biclique as follows. In SHA-256 fix all the message words to constants, then apply the difference Δ_1^M to W^{17}, and assume the linear evolution of Δ_1^M when calculating ΔW^{17+i} from $W^2, \ldots, W^{17}$. Assume also the linear evolution of ∇M when calculating ∇W^{27-i} from $W^{28}, \ldots, W^{42}$. Analogously for SHA-512.
- Build the biclique using internal message words as freedom, then spend the remaining 5 message words to ensure the Δ and ∇-trails in the message schedule. As a result, we get the longest possible chunks (2-16 and 28-42 in SHA-256).

Therefore, we gain 5 more rounds in the biclique, and two more rounds in the forward chunk. This results in a 52-round attack on the SHA-256 compression function, and a 57-round attack on the SHA-512 compression function.

7 Discussion and Conclusions

We reconsidered meet-in-the-middle attacks and introduced a new concept of bicliques. Bicliques have large potential in attacks on narrow-pipe hash functions and block ciphers, as has been demonstrated by recent attacks on the full versions of popular block ciphers.

To emphasize new ideas and methods behind the new concept, we focused on clear definitions and a variety of construction algorithms. As for applications, in the main text we described basic steps in the best attacks so far on SHA-256, SHA-512, and the SHA-3 finalist Skein, with more details left for the Appendix. We can outline the following benefits of applying the biclique concept:

- Use of differential trails in a biclique with a small number of sufficient conditions;
- Deterministic algorithms to build a biclique, which can be adapted for a particular primitive;
- Use of various tools from differential cryptanalysis like trail backtracking [5], message modification and neutral bits [6, 15, 21, 28], condition propagation [9], and rebound techniques [20];
- Utilizing a statistical test for matching, instead of a direct or symbolic matching.

Overall, the differential view gives us much more freedom and flexibility compared to previous attacks. Though all the functions in this paper are ARX-based, our technique can be as well applied to other narrow-pipe designs.

Status of SHA-2 and Skein-512. For SHA-256, SHA-512, and Skein-512, we considered both the hash function and the compression function setting. In all settings we obtained cryptanalytic results on more rounds than any other known method. Using these data points, it seems safe to conclude that Skein-512 is more resistant against splice-and-cut cryptanalysis than SHA-512. An interesting problem to study would be possibilities for meaningful bounds on the length of biclique structures.

For Skein-512 we also apply the meet-in-the-middle approach to obtain a computational-complexity gain for its full 72-round version that goes beyond minimizing computations done for the first and last rounds. It seems worthwhile to point out that this is the only hash function in the SHA-3 finalist selection that allows for such an approach.

Future work. Apart from applying these techniques for other ciphers or hash functions, it will be interesting to find applications for generalizations of the biclique technique, i.e. situations were a graph is used that deviates from the biclique definition.

Acknowledgements

Part of this work was done while Christian Rechberger was with KU Leuven and visiting MSR Redmond, and while Alexandra Savelieva was visiting MSR Redmond. This work was supported by the European Commission under contract ICT-2007-216646 (ECRYPT II) and the Federal Target Program “Scientific and scientific-pedagogical personnel of innovative Russia“ in 2009-2013 under contract No. P965 from 27 May, 2010. The authors would like to thank Eik List for useful comments on earlier versions of the paper.

References

1. Kazumaro Aoki, Jian Guo, Krystian Matusiewicz, Yu Sasaki, and Lei Wang. Preimages for step-reduced SHA-2. In *ASIACRYPT'09*, volume 5912 of *LNCS*, pages 578–597. Springer, 2009.
2. Kazumaro Aoki and Yu Sasaki. Preimage attacks on one-block MD4, 63-step MD5 and more. In *Selected Areas in Cryptography'08*, volume 5381 of *LNCS*, pages 103–119. Springer, 2008.

3. Kazumaro Aoki and Yu Sasaki. Meet-in-the-middle preimage attacks against reduced SHA-0 and SHA-1. In *CRYPTO'09*, volume 5677 of *LNCS*, pages 70–89. Springer, 2009.
4. Jean-Philippe Aumasson, Çagdas Çalik, Willi Meier, Onur Özen, Raphael C.-W. Phan, and Kerem Varici. Improved cryptanalysis of Skein. In *ASIACRYPT'09*, volume 5912 of *LNCS*, pages 542–559. Springer, 2009.
5. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. RadioGatun, a belt-and-mill hash function. *NIST Cryptographic Hash Workshop, available at http://radiogatun.noekeon.org/*, 2006.
6. Eli Biham and Rafi Chen. Near-Collisions of SHA-0. In Matthew K. Franklin, editor, *CRYPTO'04*, volume 3152 of *LNCS*, pages 290–305. Springer, 2004.
7. Eli Biham, Orr Dunkelman, Nathan Keller, and Adi Shamir. New Data-Efficient Attacks on Reduced-Round IDEA. Cryptology ePrint Archive, Report 2011/417, 2011. http://eprint.iacr.org/.
8. Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. Cryptology ePrint Archive, Report 2011/449, 2011. http://eprint.iacr.org/2011/449, to appear in ASIACRYPT 2011.
9. Christophe De Cannière and Christian Rechberger. Finding SHA-1 characteristics: General results and applications. In *ASIACRYPT'06*, volume 4284 of *LNCS*, pages 1–20. Springer, 2006.
10. Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. The Skein hash function family (version 1.3, 1 Oct, 2010).
11. 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'10*, volume 6477 of *LNCS*, pages 56–75. Springer, 2010.
12. Deukjo Hong. Biclique attack on the full HIGHT, 2011. To appear in ICISC 2011.
13. Sebastiaan Indesteege, Florian Mendel, Bart Preneel, and Christian Rechberger. Collisions and Other Non-random Properties for Step-Reduced SHA-256. In *Selected Areas in Cryptography'08*, volume 5381 of *LNCS*, pages 276–293. Springer, 2008.
14. Keting Jia, Honbo Yu, and Xiaoyun Wang. A meet-in-the-middle attack on the full KASUMI. Cryptology ePrint Archive, Report 2011/466, 2011. http://eprint.iacr.org/.
15. Antoine Joux and Thomas Peyrin. Hash functions and the (amplified) boomerang attack. In *CRYPTO'07*, volume 4622 of *LNCS*, pages 244–263. Springer, 2007.
16. John Kelsey and Bruce Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In *EUROCRYPT'05*, volume 3494 of *LNCS*, pages 474–490. Springer, 2005.
17. Dmitry Khovratovich, Ivica Nikolic, and Christian Rechberger. Rotational Rebound Attacks on Reduced Skein. In *ASIACRYPT'10*, volume 6477 of *LNCS*, pages 1–19. Springer, 2010.
18. Mario Lamberger and Florian Mendel. Higher-order differential attack on reduced SHA-256. available at http://eprint.iacr.org/2011/037.pdf, 2011.
19. Hamid Mala. Biclique cryptanalysis of the block cipher SQUARE. Cryptology ePrint Archive, Report 2011/500, 2011. http://eprint.iacr.org/.
20. Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The rebound attack: Cryptanalysis of reduced Whirlpool and Grøstl. In *FSE'09*, volume 5665 of *Lecture Notes in Computer Science*, pages 260–276. Springer, 2009.
21. Yusuke Naito, Yu Sasaki, Takeshi Shimoyama, Jun Yajima, Noboru Kunihiro, and Kazuo Ohta. Improved collision search for SHA-0. In *ASIACRYPT'06*, volume 4284 of *LNCS*, pages 21–36. Springer, 2006.
22. J. Neyman and E.S Pearson. The testing of statistical hypotheses in relation to probabilities a priori. *Proc. Camb. Phil. Soc.*, 1933.
23. NIST. FIPS-180-2: Secure Hash Standard, August 2002. Available online at http://www.itl.nist.gov/fipspubs/.
24. Somitra Kumar Sanadhya and Palash Sarkar. New collision attacks against up to 24-step SHA-2. In *INDOCRYPT'08*, volume 5365 of *LNCS*, pages 91–103. Springer, 2008.
25. Yu Sasaki and Kazumaro Aoki. Preimage attacks on step-reduced MD5. In *ACISP'08*, volume 5107 of *LNCS*, pages 282–296. Springer, 2008.
26. Yu Sasaki and Kazumaro Aoki. Finding preimages in full MD5 faster than exhaustive search. In *EUROCRYPT'09*, volume 5479 of *LNCS*, pages 134–152. Springer, 2009.
27. Bozhan Su, Wenling Wu, Shuang Wu, and Le Dong. Near-Collisions on the Reduced-Round Compression Functions of Skein and BLAKE. Cryptology ePrint Archive, Report 2010/355, 2010. http://eprint.iacr.org/.
28. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In *CRYPTO'05*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.
29. Hongbo Yu, Jiazhe Chen, Keting Jia, and Xiaoyun Wang. Near-Collision Attack on the Step-Reduced Compression Function of Skein-256. Cryptology ePrint Archive, Report 2011/148, 2011. http://eprint.iacr.org/.
30. Shao zhen Chen and Tian min Xu. Biclique Attack of the Full ARIA-256. Cryptology ePrint Archive, Report 2012/011, 2012. http://eprint.iacr.org/.

A More details on Skein specification and differential trail design

The operation MIX has two inputs x_0, x_1 and produces two outputs y_0, y_1 with the following transformation:

$$\begin{aligned} y_0 &= x_0 + x_1 \\ y_1 &= (x_1 \lll_{R_{(d \bmod 8)+1,j}}) \oplus y_0 \end{aligned}$$

The exact values of the rotation constants $R_{i,j}$ as well the permutations π (which are different for each version of Threefish) can be found in [10].

Round	Active bits
Before Round 12	0000000000000000000000000000000000000000000000000000000000000000
	0000000000000000000000000000000000000000000000000000000000000000
	0000000000000000000000000000000000000000000000000000000000000000
	0000000000000000000000000000000000000000000000000000000000000000
	0000000000000000000000000000000000000000000000000000000000000000
	0000000000000000000000000000000000000000000000000000000000000000
	0000000000000000000000000000000000000000000000000000000000000000
	0001110000000000000000000000000000000000000000000000000000000000
After Round 12	0000000000000000000000000000000000000000000000000000000000000000
	0000000000000000000000000000000000000000000000000000000000000000
	0000000000000000000000000000000000000000000000000000000000000000
	0001110000000000000000000000000000000000000111000000000000000000
	0001110000000000000000000000000000000000000000000000000000000000
	0000000000000000000000000000000000000000000000000000000000000000
	0000000000000000000000000000000000000000000000000000000000000000
	0000000000000000000000000000000000000000000000000000000000000000
After Round 13	0001110000000000000000000000000000000000000111000000000000000000
	0000000000000000000000000000000000000000000000000000000000000000
	0001110000000000000000000000000000000000000000000000000000000000
	0000000000000000000000000000000000000000000000000000000000000000
	0000000000000000000000000000000000000000000000000000000000000000
	0001110000000000000000000000000000000000000000000000000000000000
	0000000000000000000000000000000000000000000000000000000000000000
	0001110000000000001110000000000000000000000111000000000001110000
After Round 14	0001110000000000000000000000000000000000000000000000000000000000
	0001110000000000000000000000000000000000000111000000000000000000
	0001110000000000000000000000000000000000000000000000000000000000
	1111110000000011111100001110000000000011100111000000000001110000
	0001110000000000001110000000000000000000000111000000000001110000
	0001110000000000000000000000111000000000000000000000000000000000
	0001110000000000000000000000000000000000000111000000000000000000
	0001110000000000000000000000000000000000000000000000000000000000
After Round 15	1110000000000011111100001110000000000011100111000000000001110000
	0000000000000000000000000000000000011100000111000000000000011100
	0000000000000000001110000000111000000000000111000000000001110000
	0000000000000000000000000000000000000000000110110000000000000000
	0000000000000000000000000000000000000000000111000000000000000000
	0000000000011100011100000000111000001110000111000000000001110000
	0000000000000000000000000000000000000000000111000000000000000000
	1111110011100011111100110110011111100011100000111000011101110000

Table 3. Details of the most dense $\varDelta$-trail for the result on the reduced Skein-512 hash function.

Local collision in Skein-512. If an attacker controls both the IV and the tweak he is able to introduce difference in these inputs so that one of subkeys has zero difference. As a result, he gets a differential which has no difference in internal state for 8 rounds. The lowest weight of input and output differences is achieved in the following combination:

$$\varDelta K[6] = \varDelta K[7] = \varDelta T[1] = \delta,$$

which gives difference $(0, 0, \ldots, 0, \delta)$ in the subkey K^0 and $(\delta, 0, 0, \ldots, 0)$ in K^8, and zero difference in the subkey K^4. The local collisions for further rounds are constructed analogously.

We use the following differences in the compression function attack to make a local collision in rounds 8-15 and 24-31:

$$\Delta K[0] = \Delta T[0] = \Delta T[1] = 1 \ll 63; \qquad \Delta K[3] = \Delta K[4] = \Delta T[1] = 1 \ll 63.$$

Biclique					
Rounds	Dimension	Δ^M bits	∇^M bits	Complexity	Freedom used
16-23	1	$K[0]$	$K[4]_{63}$	2^{256}	162

Chunks		Matching			
Forward	Backward	Partial matching	Matching bit	Matching pairs	Complexity
8-15	24-31	$32 \rightarrow 39 = 2 \leftarrow 7$	I^3_{25}	2^2	$2^{1.1}$

Table 4. Parameters of the preimage attack on the Skein-512 compression function

B Specification of the SHA-2 Family of Hash Functions

We briefly review parts of the specification [23] needed for the cryptanalysis. The SHA-2 hash functions are based on a compression function that updates the state of eight 32-bit state variables A, ..., H according to the values of 16 32-bit words M_0, ..., M_{15} of the message. SHA-384 and SHA-512 operate on 64-bit words. For SHA-224 and SHA-256, the compression function consists of 64 rounds, and for SHA-384 and SHA-512 — of 80 rounds. The full state in round r is denoted by S^r.

The i-th step uses the i-th word W^i of the expanded message. The message expansion works as follows. An input message is split into 512-bit or 1024-bit message blocks (after padding). The message expansion takes as input a vector M with 16 words and outputs a vector W with n words. The words W^i of the expanded vector are generated from the initial message M according to the following equations (n is the number of steps of the compression function):

$$W^i = \begin{cases} M^i & \text{for } 0 \leq i < 15 \\ \sigma_1(W^{i-2}) + W^{i-7} + \sigma_0(W^{i-15}) + W^{i-16} & \text{for } 15 \leq i < n \end{cases}. \tag{11}$$

where $\sigma_0(x)$ and $\sigma_1(x)$ are linear functions.

The round function of all the SHA-2 functions operates as follows:

$$\begin{aligned}
&T_1^{(i)} = H^i + \Sigma_1(E^i) + \mathrm{Ch}(E^i, F^i, G^i) + K^i + W^i,\\
&T_2^{(i)} = \Sigma_0(A_i) + \mathrm{Maj}(A_i, B_i, C_i),\\
&A^{i+1} = T_1^{(i)} + T_2^{(i)}, B^{i+1} = A^i, C^{i+1} = B^i, D^{i+1} = C^i,\\
&E^{i+1} = D^i + T^1_{(i)},\ F^{i+1} = E^i,\ G^{i+1} = F^i,\ H^{i+1} = G^i.
\end{aligned}$$

Here K^i is a round constant.The round function uses the bitwise boolean functions Maj and Ch, and two GF(2)-linear functions $\Sigma_0(x)$ and $\Sigma_1(x)$. Functions Maj and Ch are defined identically for all the SHA-2 functions:

$$\mathrm{Ch}(x, y, z) = x \wedge y \oplus \overline{x} \wedge z \tag{12}$$

$$\mathrm{Maj}(x, y, z) = x \wedge y \oplus x \wedge z \oplus y \wedge z \tag{13}$$

For SHA-224 and SHA-256, $\Sigma_0(x)$ and $\Sigma_1(x)$ are defined as follows:

$$\Sigma_0(x) = (x \ggg 2) \oplus (x \ggg 13) \oplus (x \ggg 22), \qquad \Sigma_1(x) = (x \ggg 6) \oplus (x \ggg 11) \oplus (x \ggg 25).$$

For SHA-384 and SHA-512, they are defined as follows:

$$\Sigma_0(x) = (x \ggg 28) \oplus (x \ggg 34) \oplus (x \ggg 39), \qquad \Sigma_1(x) = (x \ggg 14) \oplus (x \ggg 18) \oplus (x \ggg 41).$$

Operations $\ggg$ and $\gg$ denote bit-rotation and bit-shift of A by x positions to the right respectively. The message schedule functions $\sigma_0(x)$ and $\sigma_1(x)$ are defined as follows for SHA-224 and SHA-256:

$$\sigma_0(x) = \ (x \ggg 7) \oplus (x \ggg 18) \oplus (x \gg 3), \qquad \sigma_1(x) = \ (x \ggg 17) \oplus (x \ggg 19) \oplus (x \gg 10)\,.$$

and for SHA-384 and SHA-512:

$$\sigma_0(x) = \ (x \ggg 1) \oplus (x \ggg 8) \oplus (x \gg 7)\,, \qquad \sigma_1(x) = \ (x \ggg 19) \oplus (x \ggg 61) \oplus (x \gg 6)\,.$$

C Details on the 46-round SHA-256 attack

C.1 Biclique construction

Here we provide more details on the biclique construction algorithm:

1. Fix a group of 6-round differential trails (the one based on 3-bit difference is listed in Table 6)

   $$\nabla_i \xrightarrow{\nabla_i^M} 0.$$

   Derive the set of sufficient conditions on the internal states (Table 8).
2. Fix the message compensation equations with constants $c_1, c_2, \ldots, c_9$ (Section C.2).
3. Fix an arbitrary Q_0 and modify it so that most of conditions in the computation $Q_0 \to P_0$ are fulfilled. Derive Q_i out of Q_0 by applying ∇_i.
4. Fix a group of 2-round trails (the one based on 3-bit difference is given in Table 7) ($\Delta W^{17} \to \Delta S^{19}$) as a Δ-trail (Equation (6)) in rounds 17-19.
5. Choose $W^{17}, W^{18}, \ldots, W^{22}$ and constants c_8, c_9 so that the conditions in the computations $Q_0 \to P_j, j = 0, \ldots, 7$ are fulfilled. Produce all P_j.

An algorithm for the biclique is detailed in Appendix, Section C.3. Finally, we produce $Q_0, \ldots, Q_7$ and $P_0, \ldots, P_7$ that conform to the biclique equations.

Biclique					
Rounds	Dimension	Δ^M bits	∇^M bits	Complexity	Freedom used
17-22	3	$W^{17}_{25,26,27}$	$W^{22}_{22,23,31}$	2^{32}	416

Message compensation	
Equations	Constants used in the biclique
9	2

Chunks		**Matching**		
Forward	Backward	Partial matching	Matching bits	Complexity per match
2-16	23-36	$37 \to 38 \leftarrow 1$	$A^{38}_{0,1,2,3}$	2^3

Table 5. Parameters of the preimage attack on the 45-round SHA-256

The complexity of building a single biclique is estimated as 2^{32}. However, as many as 7 message words are left undefined in the message compensation equations, which gives us enough freedom to reuse a single biclique up to 2^{256} times. The complexity to recalculate the chunks is upper bounded by 2^2 calls of the compression function. The total amortized complexity of running a single biclique and produced 2^2 matches on 4 bits is 2^3 calls of the compression function (see details in Appendix). Since we need 2^{252} matches, the complexity of the pseudo-preimage search is 2^{253}. Therefore, a full preimage can be found with complexity approximately $2^{1+(253+256)/2} \approx 2^{255.5}$ by restarting the attack procedure $2^{\frac{256-253}{2}} = 2^{1.5}$ times. Memory requirements are approximately $2^{1.5} \times 24$ words.

C.2 Message compensation.

Since any consecutive 16 message words in SHA-2 bijectively determine the rest of the message block used at an iteration of compression function, we need to place the initial structure within a 16-round block and define such restrictions on message dependencies that maximize the length of chunks.

We use a heuristic algorithm to check how many steps forward and backward can be calculated independently with a 6-step initial structure. We discovered that with W^{17} and W^{22} selected as the words with neutral bits, it is possible to expand 16-round message block $\{W^{12}, \ldots, W^{27}\}$ by 10 steps backwards and 9 steps forwards, so that $\{W^2, \ldots, W^{16}\}$ are calculated independently of W^{17} , and $\{W^{23}, ..., W^{36}\}$ are calculated independently of W^{22}. Below we define the message compensation conditions that make such chunk separation possible (neutral bit words are outlined in frames):

$$\begin{array}{rrr} -\sigma_1(W^{25}) + W^{27} = c_1; & -W^{19} - \sigma_1(W^{24}) + W^{26} = c_2 & -\sigma_1(W^{23}) + W^{25} = c_3 \\ -\boxed{W^{17}} + W^{24} = c_4 & -\sigma_1(W^{21}) + W^{23} = c_5; & -\sigma_1(W^{19}) + W^{21} = c_6 \\ -\sigma_1(\boxed{W^{17}}) + W^{19} = c_7; & W^{12} + \sigma_0(W^{13}) = c_8; & W^{13} + \boxed{W^{22}} = c_9 \end{array} \tag{14}$$

Fig. 4 explains how the message compensation dependencies are constructed. Columns and rows correspond to message words and equations respectively, where X at the intersection of row i ad column j shows that W^j is a part of i^{th} equation. Colour of a column reflects whether the appropriate message word is set independently of both words with neutral bits (white), calculated using NW^1 (blue) or NW^2 (yellow). We start with $\{W^2, \ldots, W^{11}, W^{22}\}$ colored blue and $\{W^{17}, W^{28}, \ldots, W^{36}\}$ colored yellow (Fig. 4, a) and aim to get rid of equations that involve both 'blue' and 'white' message words. We split these equations and introduce constants $\{c_1, \ldots, c_8, c_9\}$ (in other words, we create additional dependencies between controlled messages and words with neutral bits as shown in Fig. 4, b).

It is easy to see that words $W^{14}, \ldots, W^{16}, W^{18}$, and W^{20} can be chosen independently of both W^{17} and W^{22}, so we can assign W^{14} and W^{15} with 64-bit length of the message to satisfy padding rules (additionally, 1 bit of W^{13} needs to be fixed). W^{18} and W^{20} are additional freedom for constructing the biclique.

C.3 Trails

The basic differential trail for the biclique is a 6-round trail in the backward direction ($\Delta_Q \leftarrow \nabla M$) that starts with the difference in bits 22, 23, and/or 31 in W_{22}. The trail is briefly depicted in Table 6 with references to the sufficient conditions (which work out for all the 7 possible differences) in Table 8.

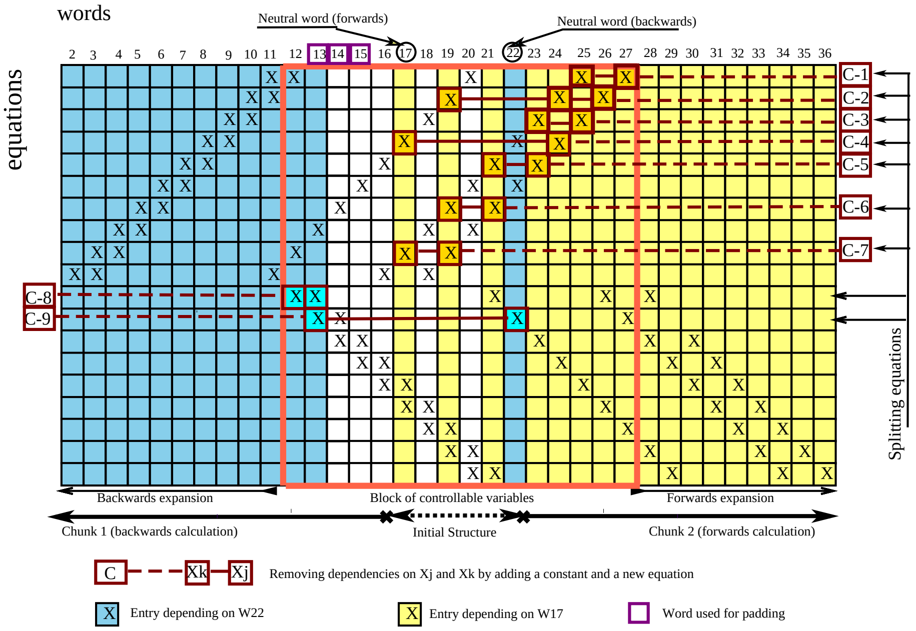


Fig. 4. Message dependencies after message compensation in SHA-256

Round	A	B	C	D	E	F	G	H	W	Cond-s
17	-	-	22,23,31	-	-	Λ'	-	*	-	1
18	-	-	-	22,23,31	-	-	Λ'	-	-	3,4
19	-	-	-	-	22,23,31	-	-	Λ'	-	7-11
20	-	-	-	-	-	22,23,31	-	-	-	12
21	-	-	-	-	-	-	22,23,31	-	-	13
22	-	-	-	-	-	-	-	22,23,31	-	
23	-	-	-	-	-	-	-	-	22,23,31	

Table 6. Details for biclique in SHA-256. Differential ∇-trail (active bits). $\Lambda' = \{6, 11, 12, 16, 17, 20, 23, 24, 29, 30\}$

We also use bits 25, 26, 27 as neutral in W_{17}. To prevent this difference to interleave with the backward trail difference in round 19, we restrict the behavior of the forward trail as specified in Table 7. The aggregated conditions, which make each forward trail keep the backward ones unaffected, are given in Table 8.

With three neutral bits we construct a biclique with 8 starting points for chunks in each direction. First, we choose the initial state $A_{17}, \ldots, H_{17}$ so that the conditions 1 and 5 are fulfilled. Then we proceed with a standard trail backtracking procedure modifying the starting state if needed. Here we are free to use all the tools from the collision search like message modification or tunnels. Next, in round 18 we further check whether the value of E stops carries in the forward trail. If not, we change the value of D in the starting state accordingly. Then we sequentially modify the initial state in order to fulfill the conditions 2-11.

The last two conditions are affected by the message words W_{19} and W_{20}. We need to fulfill three bit conditions for every W_{17}, used in the attack. Therefore, we spend $3 \cdot 8 \cdot 2 = 48$ degrees of freedom in message words $W_{17}, W_{18}, W_{19}, W_{20}, W_{21}$. Note that there is a difference in W_{19} determined by the difference in W_{17} due to the message compensation. We have fixed the constants c_6 and c_7 from Eq. 14 while defining W_{19} and W_{21}. In total, we construct the biclique in about 2^{32} time required to find proper W_{19} and W_{20}.

Round	A	B	C	D	E	F	G	H	Cond-s
18	*	-	-	-	25,26,27	-	-	-	2
19	*	*	-	-	Φ	25,26,27	-	-	5,6

Table 7. Details for biclique in SHA-256. Differential Δ-trail (active bits). $\Phi = \Sigma_1\{25, 26, 27\} = \{0, 1, 2, 14, 15, 16, 19, 20, 21\}$, $*$ refers to an arbitrary difference.

Amount of freedom used. In total, we have 512 degrees of freedom in the message and 256 degrees of freedom in the state. The biclique is determined by the state in round 17 and message words W_{17}–W_{21}. The choice of W_{19} and W_{21} is equivalent to the choice of constants c_6, c_7 in Eq. 14. Therefore, we spend $256 + 5 \cdot 32 = 416$ degrees of freedom for the biclique fulfilling as few as $47 + 42$ (Table 8) conditions. We note that we have more than 300 degrees of freedom left in the construction of a biclique. After the biclique is fixed, there are $768 - 416 = 352$ degrees of freedom left. We spend $32 + 32 + 2 = 66$ for the padding, thus leaving with 286 degrees of freedom. Therefore, one biclique is enough for the full attack.

Round	Conditions	Purpose	F	C	D_W
17	1: $A^{22,23,31} = B^{22,23,31}$	Absorption (MAJ)	IC	3	0
	2 : $(W^{\oplus} E_{18})^{25,26,27} = 0$	Stop forw. carry	SM	6	0
18	3 : $E^{\Lambda'} = 1$,	Absorption (IFF)	SM	9	0
	4 : $(D \oplus E_{19})^{22,23,31} = 0$	Stop carry	SM	3	0
	5 : $F^{25,26,27} = G^{25,26,27}$,	Absorption (IFF)	IC	9	0
	6 : $(S1 \oplus E_{19})^{\Phi} = 0$	Stop forw. carry	SM	2	0
19	7: $F^{22,31} = G^{22,31}$	Absorption (IFF)	SM	2	0
	8: $F^{23} \neq G^{23}$	Pass (IFF)	SM	1	0
	9: $CH^{25} \neq S1^{25}$	Force carry (H)	SM	1	0
	10: $(S1 \oplus H)^{\Lambda} = 1$	Stop carry (H)	SM	9	0
	11: $(CH \oplus H)^{24} = 0$	Force carry (H)	SM	1	0
	11': $(CH \oplus H)^{23} = 0$	Force carry (H)	SM	1	0
20	12 : $E^{22,23,31} = 0$	Absorption (IFF)	W^{19}	21	21
21	13 : $E^{22,23,31} = 1$	Absorption (IFF)	W^{20}	21	21

Table 8. Sufficient conditions for the ∇-trails in SHA-256.

A^i – i-th bit of A. F – how the conditions are fulfilled (IC – initial configuration, SM – state modification).
C – total number of independent conditions. D_W – conditions fulfilled by message words.
$\Lambda = \Sigma_1\{22, 23, 31\} = \{6, 11, 12, 16, 17, 20, 25, 29, 30\}$

D Details on the 50-round SHA-512 attack

D.1 Biclique construction

Attack layout The basic parameters of the pseudo-preimage attack are given in Table 9. More details:

1. Fix a group of 6-round differential trails (Table 10) for the differential

$$\nabla_i \xrightarrow{\nabla^M} 0.$$

Derive the set of sufficient conditions on the internal states.

2. Fix the message compensation equations with 9 constants (Appendix D.2).
3. Fix an arbitrary Q_0 and modify it so that the most of conditions in the computation $Q_0 \to P_0$ are fulfilled. Derive Q_i out of Q_0 by applying ∇_i.
4. Fix a group of 3-round trails (Table 11) ($\Delta W^{21} \to \Delta S^{23}$) as Δ-trails (Equation (6)) in rounds 21-23.
5. Choose $W^{21}, W^{22}, \ldots, W^{26}$ and constants c_8, c_9 so that the conditions in the computations $Q_0 \to P_j, j = 0, \ldots, 7$ are fulfilled. Produce all P_j.

Trail details for the biclique are detailed further. Finally, we produce $Q_0, \ldots, Q_7$ and $P_0, \ldots, P_7$ that conform to the biclique equations.

Biclique					
Rounds	Dimension	Δ^M bits	∇^M bits	Complexity	Freedom used
21-26	3	$W^{21}_{60,61,62}$	$W^{26}_{53,54,55}$	2^{32}	96
Equations		**Message compensation**		Constants used in the biclique	
9				2	
Chunks		**Matching**			
Forward	Backward	Partial matching	Matching bits	Complexity per match	
6-20	27-40	41 → 43 ← 5	$A^{43}_{0,1,2}$	2^3	

Table 9. Parameters of the preimage attack on the 50-round SHA-512

The complexity of building a single biclique is estimated to be 2^{32} units. However, the amortized cost is again negligible, since we have much freedom in unused message words. The complexity of getting 2^3 matches on 3 bits is 2^3 calls of the compression function. Since we need 2^{509} matches, the complexity of the pseudo-preimage search is 2^{509}. Therefore, a full preimage can be found with complexity approximately $2^{1+(509+512)/2} \approx 2^{511.5}$ by restarting the attack procedure $2^{\frac{512-509}{2}} = 2^{1.5}$ times. Memory requirements are approximately $2^{1.5} \times 24$ words.

D.2 Message compensation

The system of compensation equations is defined similarly to the attack on SHA-256:

$$\begin{array}{rrr} -\sigma_1(W^{29}) + W^{31} = c_1; & -W^{23} - \sigma_1(W^{28}) + W^{30} = c_2; & -\sigma_1(W^{27}) + W^{29} = c_3 \\ -\boxed{W^{21}} + W^{28} = c_4; & -\sigma_1(W^{25}) + W^{27} = c_5; & -\sigma_1(W^{23}) + W^{25} = c_6 \\ -\sigma_1(\boxed{W^{21}}) + W^{23} = c_7; & W^{16} + \sigma_0(W^{17}) = c_8; & W^{17} + \boxed{W^{26}} = c_9 \end{array}$$

To satisfy padding rules, we need to use 1 LSB of W^{13} and 10 LSB of W^{15}.The choice of constants c_8, c_9 and fixed lower 53 bits of W^{26} provide us with sufficient freedom. Indeed, by choosing c_9 we define lower 53 bits of W^{17}. Having c_8 chosen, we derive 45 lower bits of W^{16} fixed due to σ_0 in message schedule. Further, we get lower 37 bits of W^{15}, 29 bits of W^{14} and 21 bit of W^{13} fixed. As we need only one LSB of W^{13} and 10 LSB of W^{15} to be fixed, we use only lower 33 bits of W^{26}, lower 33 bits of c_9, and lower 25 bits of c_8. For simplicity, details of message compensation are depicted in Fig. 5.

D.3 Trails

The basic differential trail for the biclique is a 6-round trail in the backward direction ($\Delta_Q \leftarrow \nabla M$) that starts with the difference in bits 53, 54, and/or 55 in W^{26}. The trail is depicted in

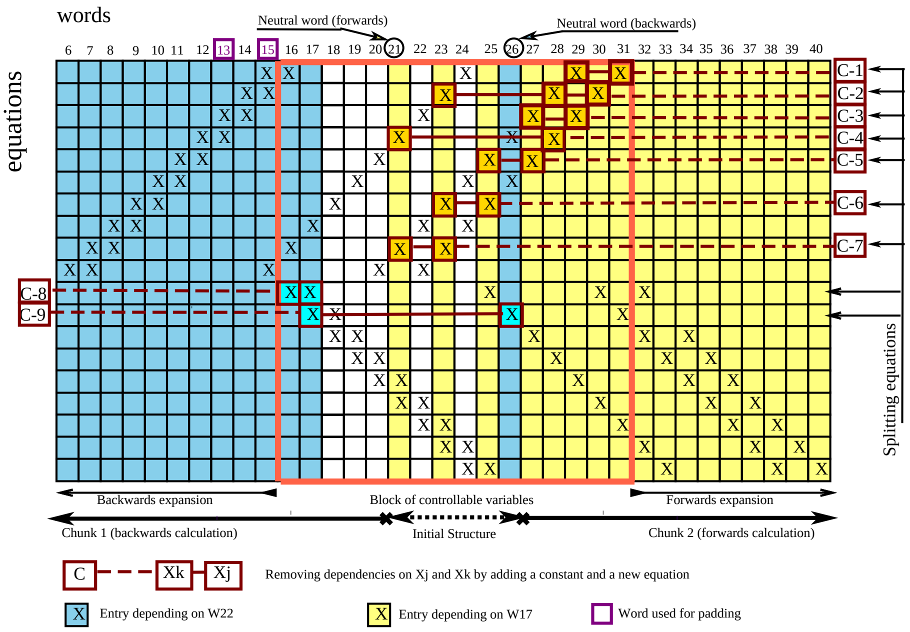


Fig. 5. Message dependencies after message compensation in SHA-512

Table 10 with the number of independent sufficient conditions.We also use bits 60, 61, 62 as neutral in W^{21}. To prevent this difference to interleave with the backward trail difference in round 19, we restrict the behavior of the forward trail as specified in Table 11. Note that the trails based on the linearized version are now compatible with our choice of neutral bits. The biclique is basically the same as in SHA-256, with a small difference that we spend $48 + 48 = 96$ degrees of freedom inside.

Round	A	B	C	D	E	F	G	H	Indep. cond-s
21	-	-	53,54,55	-	-	Λ	-	*	3
22	-	-	-	53,54,55	-	-	Λ	-	12
23	-	-	-	-	53,54,55	-	-	Λ	12
24	-	-	-	-	-	53,54,55	-	-	24
25	-	-	-	-	-	-	53,54,55	-	24
26	-	-	-	-	-	-	-	53,54,55	

Table 10. Biclique in SHA-512. Differential ∇-trail (active bits). $\Lambda = \Sigma_1\{53, 54, 55\} = \{12, 13, 14, 35, 36, 37, 39, 40, 41\}$

Complexity estimate. We get a pseudo-preimage with complexity approximately $2^{506} \times 2^3 = 2^{509}$ compression function operations. Therefore, a full preimage can be found with complexity approximately $2^{1+(509+512)/2} \approx 2^{511.5}$ by restarting the attack procedure $2^{\frac{512-509}{2}} = 2^{1.5}$ times from step 2. Memory requirements are approximately 4 message words (2 message words for storing the fixed parts of neutral bits, 2^3 entries of 3 neutral bits difference and 3 bits for matching in each list). For finding a preimage, we need to store $2^{1.5}$ pseudo-preimages, i.e. the memory requirement is $2^{1.5} \times 24$ words.

Round	A	B	C	D	E	F	G	H	Cond.
22	*	-	-	-	60,61,62	-	-	-	3
23	*	*	-	-	Φ	60,61,62	-	-	18

Table 11. Biclique in SHA-512. Differential Δ-trail. $\Phi = \Sigma_1\{60,61,62\} = \{17,20,21,42,43,44,46,47,48\}$, $*$ refers to an arbitrary difference.

E Comparison with more generic brute-force optimizations

Due to the nature of our results being close to brute force in terms of time complexity, we here discuss efficiency optimizations of otherwise naive brute-force search. This serves as a benchmark for our results that use bicliques.

E.1 Second preimage search for full Skein-512

Skein-512 appears to be the only hash function in the finalist selection of the SHA-3 competition that allows for time-complexity gains over brute-force search using cryptanalytic meet-in-the-middle strategies[1]. In here we explore this further, while noting that the MITM strategy pursued here is not using bicliques.

We start with the simple observation that when looking through a message space the first rounds need only be partially computed. When using the 64-bit modular addition is the cost metric, the first round can be for free, the second rounds needs only one instead of 8 computations, etc, saving more than 3 round computations in total. Likewise, if only a few output bits instead of all need to be computed, e.g. for matching with the target hash value, a similar number of computations can be saved. Similar observations can and have been made for other primitives, however as Skein-512 is the only narrow-pipe SHA-3 finalist, we can go further. Those bits on which the check is performed need not be at the end of the compression function call, but can be at any point in the internal state. This allows to also have savings in another chunk. One simple example of such savings are neutral bit effects. Similar to [17], in experiments we found that for 7 rounds in the forwards direction, many neutrals bits to not affect a number of state bits with probability close to 1. These neutral bits can in turn be used as the inner loop of the search space. The total number of rounds that are saved are hence at least 13. For full 72-round Skein, this leads to a preimage search complexity of no more than $2^{511.71}$. For 22-round Skein, the effort would be $2^{510.71}$. By spending some computation (that is later amortized) to find suitable chaining values that allow for longer neutral bits, or by simply choosing a suitable CV in the compression function setting, these results can be improved further.

[1] The narrow pipe Skein-256-256 also allows for the approach, but all versions of Keccak, Grøstl, and JH are wide-pipe and do not allow this. For Blake the situation is less clear.

Converting Meet-in-the-Middle Preimage Attack into Pseudo Collision Attack: Application to SHA-2

Ji Li[1], Takanori Isobe[2], and Kyoji Shibutani[2]

[1] Sony China Research Laboratory, China
Ji.Li@sony.com.cn
[2] Sony Corporation, Japan
{Takanori.Isobe,Kyoji.Shibutani}@jp.sony.com

Abstract. In this paper, we present a new technique to construct a collision attack from a particular preimage attack which is called a partial target preimage attack. Since most of the recent meet-in-the-middle preimage attacks can be regarded as the partial target preimage attack, a collision attack is derived from the meet-in-the-middle preimage attack. By using our technique, pseudo collisions of the 43-step reduced SHA-256 and the 46-step reduced SHA-512 can be obtained with complexities of 2^{126} and $2^{254.5}$, respectively. As far as we know, our results are the best pseudo collision attacks on both SHA-256 and SHA-512 in literature. Moreover, we show that our pseudo collision attacks can be extended to 52 and 57 steps of SHA-256 and SHA-512, respectively, by combined with the recent preimage attacks on SHA-2 by bicliques. Furthermore, since the proposed technique is quite simple, it can be directly applied to other hash functions. We apply our algorithm to several hash functions including Skein and BLAKE, which are the SHA-3 finalists. We present not only the best pseudo collision attacks on SHA-2 family, but also a new insight of relation between a meet-in-the-middle preimage attack and a pseudo collision attack.

Keywords: hash function, narrow-pipe, SHA-2, Skein, BLAKE, meet-in-the-middle attack, preimage attack, pseudo collision attack

1 Introduction

Cryptographic hash functions play a central role in the modern cryptography. A secure hash function, which produces a fixed length hash value from an arbitrary length message, is required to satisfy at least three security properties: preimage resistance, second preimage resistance and collision resistance.

While there has not been a generic method to convert a collision attack into a preimage attack, it has been known that the preimage attack that can find at least two distinct preimages from the same target can be directly converted into a collision attack. However, the converted collision attack is often not efficient due to that the birthday bound of a collision attack ($2^{n/2}$) is far lower than the generic bound of the preimage attack (2^n), where n is the bit size of the hash value. Thus, it is left as open question that how to convert an efficient preimage attack into an efficient collision attack. In the case of the reduced SHA-256 regarding the number of attacked rounds, a preimage attack, covering 43 steps [4], is much better than the best known collision attack, with only 27 steps [17]. Moreover, basically, a collision attack and a preimage attack require quite different techniques. In other words, in general, the techniques used for the collision attack do not work well for a preimage attack, and vice versa. In fact, most of the recent collision attacks are based on a differential attack [32, 31], in contrast to that most of the recent preimage attacks are based on a meet-in-the-middle (MITM) attack [2]. Though converting the differential collision attack to a (pseudo) preimage attack was discussed in [8], there is no generic way to construct a collision attack from a MITM preimage attack.

In this paper, we give a generic method to convert a particular preimage attack into a collision attack. By using our technique, an efficient collision attack which works faster than a generic collision attack can be constructed from a partial target preimage attack even if the complexity of the preimage attack is more than the birthday bound ($2^{n/2}$). Our method is especially fit for converting a MITM preimage attack into a pseudo collision attack, since most of the recent MITM preimage attacks can be considered as the partial target preimage attack as long as its matching point is located in the end of the compression function. We first apply our algorithm to SHA-256 and SHA-512 and show the best pseudo collision attacks on them in literature. Specifically, pseudo collisions of the 43-step (out of 64-step) reduced SHA-256 and the 46-step (out of 80-step) reduced SHA-512 can be derived faster than a generic attack. Combined with the recent preimage attacks on SHA-2 [14], these attacks are extended to the 52-step and 57-step reduced SHA-256 and SHA-512, respectively. Then we show some other applications of our conversion techniques including a pseudo collision attack on the 37-round reduced Skein-512 and pseudo collision attacks on the 4-round reduced BLAKE-256/512 without the initialization function. While it seems hard to extend our pseudo collision attacks to collision attacks, the proposed conversion technique is a generic, and thus it is expected to be widely used for security evaluations of hash functions.

This paper is organized as follows. Some security notions and a meet-in-the-middle preimage attack are introduced in Section 2. Section 3 introduces our approach for constructing a pseudo collision attack. Then, applications of our technique to SHA-256 and SHA-512 are presented in Section 4. The result on Skein is described in Section 5. Finally, we conclude in Section 6.

2 Preliminaries

In this section, we first give security notions used throughout this paper, then briefly refer a meet-in-the-middle (MITM) preimage attack.

2.1 Security Notions

Let f be a compression function which outputs an n-bit chaining variable h_i from an n-bit input chaining variable h_{i-1} and a k-bit input message m_i, i.e., $h_i = f(h_{i-1}, m_i)$. Similarly, let H be an iterated hash function consisting of f, which produces an n-bit hash value d from an initial value $IV (= h_0)$ and an arbitrary length message M, i.e., $d = H(IV, M) = f(\cdots f(f(IV, m_1), m_2), \cdots, m_t)$, where $pad(M) = (m_1|m_2|\cdots|m_t)$ and pad denotes a padding function. This type of hash function, in which the size of an intermediate chaining variable is the same as that of a hash value, is called a *narrow-pipe* hash function. On the other hand, a hash function having a larger internal state size is called a *wide-pipe* hash function, i.e., the size of a final hash value is smaller than that of a chaining variable. We use the terminology introduced in [15] for a collision attack and a pseudo (or free-start) collision attack on hash functions as follows.

Definition 1 (Collision attack). *Given IV, find (M, M') such that $M \neq M'$ and $H(IV, M)$ $= H(IV, M')$.*

Definition 2 (Free-start or pseudo collision attack). *Find (IV, IV', M, M') such that $H(IV, M) = H(IV', M')$ and $(IV, M) \neq (IV', M')$.*

Additionally, we give several definitions for (pseudo) preimage attacks on hash functions and (pseudo) preimage attacks on compression functions.

Definition 3 (Preimage attack). *Given IV and $d(= H(IV, M))$, find M' such that $H(IV,$ $M') = d$.*

Definition 4 (Pseudo preimage attack). *Given $d(= H(IV, M))$, find (IV', M') such that $H(IV', M') = d$.*

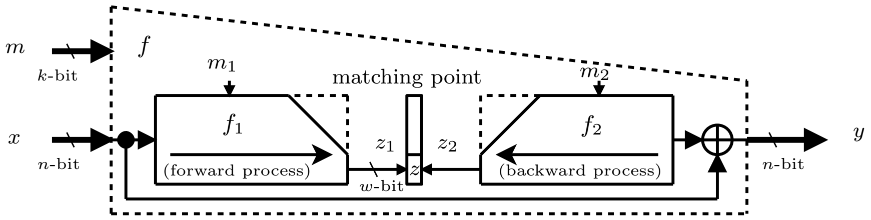


Fig. 1. Meet-in-the-middle preimage attack

Definition 5 ((*t*-bit) partial target preimage attack). *Given IV and t-bit partial target of $d(= H(IV, M))$, find M' such that t-bit of $d'(= H(IV, M'))$ is the same as the t-bit of d at the same position, and the other part of d' is randomly obtained.*

Definition 6 (Preimage attack on compression function). *Given h_{i-1} and $h_i(= f(h_{i-1}, m_i))$, find m'_i such that $f(h_{i-1}, m'_i) = h_i$.*

Definition 7 (Pseudo preimage attack on compression function). *Given $h_i(= f(h_{i-1}, m_i))$, find (h'_{i-1}, m'_i) such that $f(h'_{i-1}, m'_i) = h_i$.*

Definition 8 ((*t*-bit) partial target preimage attack on compression function). *Given h_{i-1} and t-bit partial target of $h_i(= f(h_{i-1}, m_i))$, find m'_i such that t-bit of $h'_i(= f(h_{i-1}, m'_i))$ is the same as the t-bit of h_i at the same position, and the other part of h'_i is randomly obtained.*

Definition 9 ((*t*-bit) pseudo partial target preimage attack on compression function). *Given t-bit partial target of $h_i(= f(h_{i-1}, m_i))$, find (h'_{i-1}, m'_i) such that t-bit of $h'_i(= f(h'_{i-1}, m'_i))$ is the same as the t-bit of h_i at the same position, and the other part of h'_i is randomly obtained.*

2.2 Meet-in-the-Middle Preimage Attack

The basic concept of the MITM preimage attack was introduced in [22, 16]. Since then, the MITM preimage attacks have been drastically improved and applied to several hash functions [2, 28, 27, 3, 13, 4, 10]. Also, the techniques for the MITM preimage attacks on hash functions have been extended to the attacks on several block ciphers [7, 12].

As shown in Fig. 1,[3] in the MITM preimage attack on a compression function, the compression function f is assumed to be divided into two sub-functions: f_1 (forward process) and f_2 (backward process) so that the w-bit matching point z calculated by f_1 does not depend on m_2 which is some message bits of m, and z calculated by f_2 does not depend on m_1 which is other message bits of m. Such m_1 and m_2 are called neutral bits of f_2 and f_1, respectively. Then, the MITM preimage attack finds a preimage m' such that $f(x, m') = y$ from a given x and $y(= f(x, m))$ as follows.

Step 1. Choose a random m except for m_1 and m_2.

Step 2. For all possible m_1, calculate w-bit $z_1(= f_1(x, m_1))$, and add a pair of $(z_1^{(i)}, m_1^{(i)})$ to a list, where $(1 \leq i \leq 2^{|m_1|})$, and $| * |$ denotes the bit size of $*$.

Step 3. For all possible m_2, calculate w-bit $z_2(= f_2^{-1}(x \oplus y, m_2))$, and add a pair of $(z_2^{(j)}, m_2^{(j)})$ to a list, where $(1 \leq j \leq 2^{|m_2|})$.

Step 4. Compare two lists to find pairs satisfying $z_1^{(p)} = z_2^{(q)}$. If such pair is found, then check if the other bits of the matching point derived from $m_1^{(p)}$ and $m_2^{(q)}$ are the same value.

[3] Here, we show the MITM preimage attack on Davies-Meyer mode as an example. MITM preimage attacks on other modes like Matyas-Meyer-Oseas mode can be performed in a similar way.

Step 5. If the other parts are also the same, then outputs such m including $m_1^{(p)}$ and $m_2^{(q)}$. Otherwise, go back to Step 1 and repeat the computation.

From Steps 2 and 3, we have $2^{|m_1|}$ and $2^{|m_2|}$ values of w-bit z_1 and z_2, i.e., we have $2^{|m_1|+|m_2|}$ values of $(z_1 \oplus z_2)$. Since the probability of $(z_1 \oplus z_2 = 0)$ is 2^{-w}, we have $2^{|m_1|+|m_2|} \cdot 2^{-w}$ pairs such that $z_1 = z_2$ in Step 4. Thus, by repeating this algorithm about $2^{n-w} \cdot 2^{-(|m_1|+|m_2|)} \cdot 2^w$ times, we expect to obtain a desired preimage. The required computation for the one process from Step 1 to 5 is at most $\max(2^{|m_1|}, 2^{|m_2|})$ calls of the compression function. Thus, the total computation to find a preimage of the compression function is about $2^n \cdot 2^{-(|m_1|+|m_2|)} \cdot \max(2^{|m_1|}, 2^{|m_2|})$.[4]

For a narrow-pipe hash function, by replacing x and y by IV and d, this MITM preimage attack on a compression function can be directly converted into a preimage attack on a hash function. However, for an attack on a hash function, some of the message bits related to the padding bits are required to be controlled by the attacker to set appropriate padding data.

3 Method to Convert Preimage Attack into Collision Attack

In this section, we present how to efficiently convert a particular preimage attack into a pseudo collision attack. First, we introduce a generic technique to construct a pseudo collision attack from a partial target preimage attack. Then, we introduce the MITM preimage attack whose matching point is located at the end of the compression function. We show that such class of the MITM preimage attack is regarded as the partial target preimage attack. Finally, we show that a pseudo collision attack can be efficiently constructed from the MITM preimage attack whose matching point is at the end by showing how to efficiently obtain many partial target preimages.

3.1 Generic Conversion of Partial Target Preimage Attack into Collision Attack

We consider the oracle $\mathcal{A}$ that can find a t-bit partial target preimage with a complexity of 2^s. Also, $\mathcal{A}$ is assumed to return different M' for each call. Obviously, we can construct a collision attack with a complexity of $2^s \cdot 2^{(n-t)/2}$ by iteratively calling $\mathcal{A}$ as follows.

- Set t-bit random data as d'
- Call $\mathcal{A}$ with the parameter IV and d' in $2^{(n-t)/2}$ times

After this procedure, we have $2^{(n-t)/2}$ of $(n-t)$-bit random data, and thus there exists a colliding data with a high probability. Once the colliding data are found, we have a collision of the hash function since the rest of the hash value d' is fixed. The total complexity is $2^{(n-t)/2} \cdot 2^s$. The memory requirement can be reduced to the memory requirement of finding a partial target preimage by using memory free birthday attack [29, 21]. This conversion itself can be applied to not only a narrow-pipe hash but also a wide-pipe hash, since the required complexity depends only on the size of the digest. The basic concept of this attack that fixes t-bit of the target with the complexity of 2^s has been used to find a collision of (new) FORK-256 in [22] and a collision and a second preimage of LUX in [33]. However, the method does not work if the partial target preimage attack is not efficient, i.e., $(s \geq t/2)$. In this case, the required complexity in total will be higher than $2^{n/2}$.

3.2 Meet-in-the-Middle Attack with Matching Point in Last Step

We consider a similar model explained in Section 2.2. The difference from the model shown in Fig. 1 is that the matching point is restricted to be in the last step as shown in Fig. 2. In this

[4] The estimated complexity does not contain the size of the matching point w. However, as discussed in [10], if w is extremely small like $w = 1$, the total complexity is dominated by the recomputations in Step 4 which is ignored in our estimation. Thus, in our evaluation, we assume that w is sufficiently large.

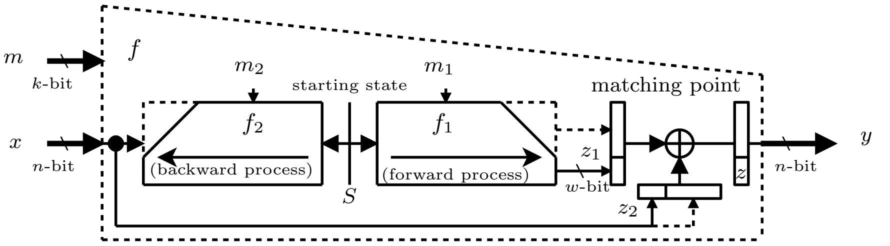


Fig. 2. MITM preimage attack with the matching point in the last step

scenario, the MITM pseudo preimage attack on a compression function finds a preimage m' and a random x' such that $f(x', m') = y$ from a given $y(= f(x, m))$ as follows.

Step 1. Choose a random m except for m_1 and m_2, and a random starting state S.

Step 2. For all possible m_1, calculate w-bit $z_1(= f_1(S, m_1))$, and add a pair of $(z_1^{(i)}, m_1^{(i)})$ to a list, where $(1 \le i \le 2^{|m_1|})$.

Step 3. For all possible m_2, calculate w-bit $z_2(= f_2^{-1}(S, m_2))$, and add a pair of $(z_2^{(j)}, m_2^{(j)})$ to a list, where $(1 \le j \le 2^{|m_2|})$.

Step 4. Compare two lists to find pairs satisfying that $z_1^{(p)} \oplus z_2^{(q)}$ equals the t-bit of y. If such pair is found, then check if the XORed other bits of the matching point derived from $m_1^{(p)}$ and $m_2^{(q)}$ is the same as the rest of y.

Step 5. If the XORed other bits are also the same as y, then output such m including $m_1^{(p)}$ and $m_2^{(q)}$, and x' calculated from the data of the matching point. Otherwise, go back to Step 1 and repeat the computation.

Note that, this attack basically cannot obtain a preimage from the given x unlike the attack described in Section 2.2, since x' will be randomly derived. Thus, this attack is considered as a pseudo preimage attack on a compression function. However, for a narrow-pipe hash, it has been known that a pseudo preimage attack on a compression function can be converted into a preimage attack on a hash function assuming that the attacker can set valid padding bits [19, 10]. The estimated complexity to find a desired pseudo preimage is the same as that presented in Section 2.2, i.e., $2^n \cdot 2^{-(|m_1|+|m_2|)} \cdot \max(2^{|m_1|}, 2^{|m_2|})$.

3.3 Conversion of MITM Preimage Attack into Pseudo Collision Attack

If we can construct the MITM pseudo preimage attack whose matching point is located at the end of the compression function, we can control part of the output variables as explained in the previous subsection. In other words, the MITM pseudo preimage attack described in the previous subsection can be regarded as the pseudo partial target preimage attack on a compression function. For the MITM preimage attack, at least $2^{t/2}$ computations are required to derive a preimage of an t-bit partial target. Thus, the directly converted pseudo collision attack will at least have the complexity of $2^{(n-t)/2+t/2} = 2^{n/2}$, that is not an efficient pseudo collision attack.

In order to overcome this problem, we exploit extra freedom of a neutral word after finding a partial target preimage. For example, in the case of $t = 10$ and $|m_1| = |m_2| = 8$ $(> t/2)$, we can find $2^6(= 2^{8+8}/2^{10})$ 10-bit partial target preimages with the complexity of 2^8. It essentially means that a 10-bit partial target preimage is found with the complexity of $2^2(= 2^8/2^6) < 2^5(= 2^{10/2})$. When $t \le w$, the required complexity to find a partial target preimage from a given t-bit partial target is estimated as

$$2^{t-(|m_1|+|m_2|)} \cdot \max(2^{|m_1|}, 2^{|m_2|}),$$

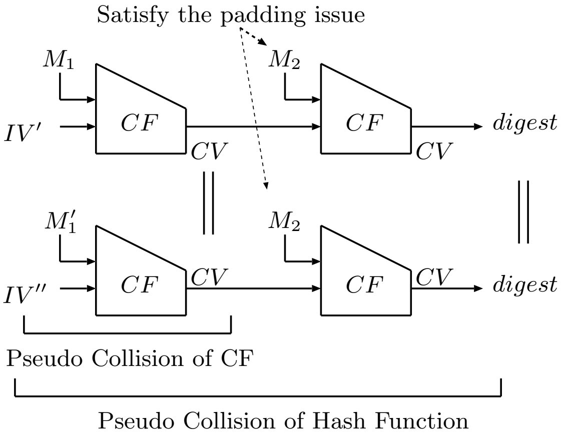


Fig. 3. Multi-block pseudo collision

where recall that w denotes the bit size of the matching point. In particular, $s < t/2$, which is the condition for a successful attack as mentioned in Section 3.1, holds when $\min(|m_1|, |m_2|) > t/2$, where recall that 2^s represents the required complexity to find a t-bit partial target preimage. Therefore, if we can move the matching point of the MITM attack to the end of the compression function and there is enough freedom in neutral words, we can construct an efficient pseudo collision attack on a compression function.

Moreover, for a narrow-pipe hash function, it has been known that a (pseudo) collision attack on a compression function can be directly converted to a (pseudo) collision attack on a hash function by appending another message block illustrated in Fig. 3, which is called multi-block message technique. By using the multi-block message technique, an attacker can append arbitrary messages. Thus, unlike the conversion to a (pseudo) preimage attack on a hash function, for the conversion to a pseudo collision attack on a hash function, there is no restriction on controllability of message bits for a MITM pseudo preimage attack on a compression function. This will relax conditions on the position of the matching point for the MITM pseudo preimage attack on a compression function, and thus may allow us to attack larger number of steps. Note that, for a wide-pipe hash function, even though a (pseudo) collision attack on a compression function can not be directly converted to a (pseudo) collision attack on a hash function by using multi-block message, we still can convert a MITM pseudo preimage attack on a hash function to a pseudo collision attack on a hash function since the conversion of a partial target preimage attack into a collision attack is generic.

4 Pseudo Collision Attacks on SHA-2

In this section, we apply our conversion technique to SHA-2. At first, we briefly describe the algorithm of SHA-2. Then, we review the previous collision attacks on SHA-2. After that, we introduce the known MITM preimage attack on the 43-step SHA-256 presented in [4]. After we modify these results in order to fit our conversion technique, i.e., moving the matching point to the end of the compression function, we show the pseudo collision attack on the 43-step SHA-256. Moreover, we present the pseudo collision attack on the 46-step SHA-512 based on the MITM preimage attack on the 46-step SHA-512 [4]. Furthermore, pseudo collision attacks on the 40-step reduced SHA-224 and SHA-384 are demonstrated as well. Finally, we discuss pseudo collision attacks based on the recent MITM preimage attacks [14], which significantly improve the results of [4] in terms of the number of attacked steps by using *bicliques*. These results on SHA-2 are summarized in Table 1.

Table 1. Summary of collision attacks on the reduced SHA-2

algorithm	type of attack	steps	complexity	based attack	paper
SHA-256	collision	24	$2^{28.5}$	-	[11]
	collision	27	(practical)	-	[17]
	semi-free-start-collision^{*1}	24	2^{17}	-	[11]
	semi-free-start-collision^{*1}	32	(practical)	-	[17]
	pseudo-near-collision	31	2^{32}	-	[11]
	pseudo collision	42	2^{123}	[4]	Our (Section 4.7)
	pseudo collision	43	2^{126}	[4]	Our (Section 4.4)
	pseudo collision	45	$2^{126.5}$	[14]	Our (Section 4.9)
	pseudo collision	52	$2^{127.5}$	[14]	Our (Section 4.9)
SHA-224	pseudo collision	40	2^{110}	[4]	Our (Section 4.8)
SHA-512	collision	24	$2^{28.5}$	-	[11]
	pseudo collision	42	2^{244}	[4]	Our (Section 4.7)
	pseudo collision	46	$2^{254.5}$	[4]	Our (Section 4.6)
	pseudo collision	50	$2^{254.5}$	[14]	Our (Section 4.9)
	pseudo collision	57	$2^{255.5}$	[14]	Our (Section 4.9)
SHA-384	pseudo collision	40	2^{183}	[4]	Our (Section 4.8)

$*1$: semi-free-start-collision attack finds (IV', M, M') such that $H(IV', M) = H(IV', M')$ and $M \neq M'$.

4.1 Description of SHA-2

While our target is both SHA-256 and SHA-512, we only explain the structure of SHA-256, since SHA-512 is structurally equivalent to SHA-256 except for the number of steps, the amount of rotations and the word size. The compression function of SHA-256 consists of a message expansion function and a state update function. The message expansion function expands a 512-bit message block into 64 32-bit message words $(W_0, \cdots, W_{63})$ as follows:

$$W_i = \begin{cases} M_i & (0 \leq i < 16), \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} & (16 \leq i < 64), \end{cases}$$

where the functions $\sigma_0(X)$ and $\sigma_1(X)$ are defined by

$$\begin{aligned} \sigma_0(X) &= (X \ggg 7) \oplus (X \ggg 18) \oplus (X \gg 3), \\ \sigma_1(X) &= (X \ggg 17) \oplus (X \ggg 19) \oplus (X \gg 10). \end{aligned}$$

The state update function updates eight 32-bit chaining variables, $A, B, \cdots, G, H$ in 64 steps as follows:

$$\begin{aligned} T_1 &= H_i + \Sigma_1(E_i) + Ch(E_i, F_i, G_i) + K_i + W_i, \\ T_2 &= \Sigma_0(A_i) + Maj(A_i, B_i, C_i), \\ A_{i+1} &= T_1 + T_2,\ B_{i+1} = A_i,\ C_{i+1} = B_i,\ D_{i+1} = C_i, \\ E_{i+1} &= D_i + T_1,\ F_{i+1} = E_i,\ G_{i+1} = F_i,\ H_{i+1} = G_i, \end{aligned}$$

where K_i is the i-th step constant and the functions Ch, Maj, Σ_0 and Σ_1 are given as follows:

$$\begin{aligned} Ch(X, Y, Z) &= XY \oplus \overline{X}Z, \\ Maj(X, Y, Z) &= XY \oplus YZ \oplus XZ, \\ \Sigma_0(X) &= (X \ggg 2) \oplus (X \ggg 13) \oplus (X \ggg 22), \\ \Sigma_1(X) &= (X \ggg 6) \oplus (X \ggg 11) \oplus (X \ggg 25). \end{aligned}$$

After 64 steps, a feed-forward process is executed with initial state variables by using word-wise addition modulo 2^{32}.

4.2 Known Collision Attacks on SHA-2

The first collision attack on reduced SHA-256 was presented in [18] which is a 19-step near collision attack. Since then, the collision attacks on SHA-2 have been improved [20, 23, 25, 24, 26, 11, 17]. The previously published best collision attacks in terms of the number of attacked steps are the 27 steps on SHA-256 [17] and the 24 steps on SHA-512 [11, 25]. A non-random property, which is a second-order differential collision, of the 47-step reduced SHA-256 compression function was reported in [6].

4.3 Known MITM Preimage Attack on 43-step SHA-256 [4]

The MITM preimage attack on the 43-step SHA-256 presented in [4] uses the 33-step two chunks $W_j, \ldots, W_{j+32}$ including the 4-step initial structure (IS), the 2-step partial fixing (PF), the 7-step partial matching (PM) and the 1-step indirect partial matching (IPM). In the following, we review the details of these techniques.

33-step Two Chunks with the 4-step IS. The message words of length 33 is divided into two chunks as $\{W_j, \ldots, W_{j+14}, W_{j+18}\}$ and $\{W_{j+15}$, W_{j+16}, W_{j+17}, $W_{j+19}, \ldots, W_{j+32}\}$. Using message compensation technique [4], the first chunk and the second chunk are independent from W_{j+15} and W_{j+18}, respectively. In particular, the following constraints ensure the above message words to be neutral words with respect to each chunk;

$$\begin{array}{lll} W_{j+17} = \sigma_1(W_{j+15}), & W_{j+19} = \sigma_1^2(W_{j+15}), & W_{j+21} = \sigma_1^3(W_{j+15}), \\ W_{j+22} = W_{z+5}, & W_{j+23} = \sigma_1^4(W_{j+15}), & W_{j+24} = 2\sigma_1(W_{j+15}), \\ W_{j+25} = \sigma_1^5(W_{j+15}), & & \end{array} \tag{1}$$

where $\sigma_1^2(X)$ means $\sigma_1 \circ \sigma_1(X)$.

These two chunks include the 4-step IS, which essentially exchanges the order of the words W_i and W_{i+3} by exploiting the absorption property of the function Ch. After the swapping, the final output after the step $(i+3)$ still keeps unchanged. Here, W_{j+18} is moved to the first chunk and W_{j+15}, W_{j+16} and W_{j+17} are moved to the second chunk.

In the forward direction, a state value of $p_{j+33} = A_{j+33}||\ldots||H_{j+33}$ can be computed independently of the first chunk. In the backward direction, a state value of $p_j = A_j||\ldots||H_j$ can be computed independently of the second chunk. Note that the 33-step two-chunk is valid regardless of the choice of j for $j > 0$.

7-step PM. In the backward computation, A_j can be computed from p_{j+7} without knowing $\{W_j, \cdots, W_{j+6}\}$ for any j as used in [13].

2-step PF. PF is a technique to enhance PM by fixing a part of a neutral word. The equation for H_{j-1} is as follows:

$$\begin{cases} H_{j-1} = A_j - \Sigma_0(B_j) - Maj(B_j, C_j, D_j) - \Sigma_1(F_j) \\ \qquad\quad -Ch(F_j, G_j, H_j) - K_{j-1} - W_{j-1}, \\ W_{j-1} = W_{j+15} - \sigma_1(W_{j+13}) - W_{j+8} + \sigma_0(W_j). \end{cases}$$

If we fix the lower ℓ bits of W_{j+15}, which is assumed to be a neutral word for the other chunk, the lower ℓ bits of H_{j-1} can be computed without using the value of the higher $(32-\ell)$ bits of W_{j+15}. Furthermore, the equation for H_{j-2} is expressed as follows:

$$\begin{cases} H_{j-2} = A_{j-1} - \Sigma_0(B_{j-1}) - Maj(B_{j-1}, C_{j-1}, D_{j-1}) - \Sigma_1(F_{j-1}) \\ \qquad\quad -Ch(F_{j-1}, G_{j-1}, H_{j-1}) - K_{j-2} - W_{j-2}, \\ W_{j-2} = W_{j+14} - \sigma_1(W_{j+12}) - W_{j+7} + \sigma_0(W_{j-1}). \end{cases}$$

The lower $(\ell - 18)$ bits of H_{j-2} can be computed if we can obtain the lower ℓ bits of $Ch(F_{j-1}, G_{j-1}, H_{j-1})$ and the lower $(\ell - 18)$ bits of $\sigma_0(W_{j-1})$. Note that these values can be computed by using only the lower ℓ bits of W_{j+15}. Thus, when we fix the lower ℓ bits of W_{j+15}, the lower $(\ell - 18)$ bits of H_{j-2} can be computed without knowing the higher $(32 - \ell)$ bits of W_{j+15}. Therefore, by combining the 7-step PM with the 2-step PF, 9 steps can be skipped in the backward computation.

1-step IPM. For the forward computation, A_{j+34} can be expressed as a sum of two independent functions ψ_F, ξ_F of each neutral word as follows;

$$\begin{cases} A_{j+34} = \Sigma_0(A_{j+33}) + Maj(A_{j+33}, B_{j+33}, C_{j+33}) + H_{j+33} + \Sigma_1(A_{j+33}) \\ \qquad\quad +Ch(A_{j+33}, B_{j+33}, C_{j+33}) + K_{j+33} + W_{j+33}, \\ W_{j+33} = \sigma_1(W_{j+31}) + W_{j+26} + \sigma_0(W_{j+18}) + W_{j+17}, \end{cases}$$
$$\Rightarrow A_{j+34} = \psi_F(W_{j+15}) + \xi_F(W_{j+18}).$$

Then, we can compute $\psi_F(W_{j+15})$ and $\xi_F(W_{j+18})$ independently. It is equivalent to move the computation of $\xi_F(W_{j+18})$ to the backward chunk. In this case, $\xi_F(W_{j+18}) = \sigma_0(W_{j+18})$.

Attack Overview. These techniques enable us to construct the 43 (= 33 + 7 + 2 + 1)-step attack on SHA-256. Here, we have the freedom of choice of j as long as 36 steps (W_{j-2} to W_{j+34}) is located sequentially.

For the actual attack in [4], j is chosen as $j = 3$, because W_{13}, W_{14} and W_{15} can be freely chosen to satisfy the message padding rule. The matching state is the lower 4 bits of A_{37}. In addition, the number of fixed bits ℓ for PF is chosen as $\ell = 23$. Then, neutral words of W_{18} and W_{21} have 5- and 4-bit freedom degrees, respectively. As a result, a pseudo preimage is found with the complexity of $2^{251.9}$. After that, pseudo preimages are converted into a preimage with the complexity of $2^{254.9}$. See [4] for more details about this attack.

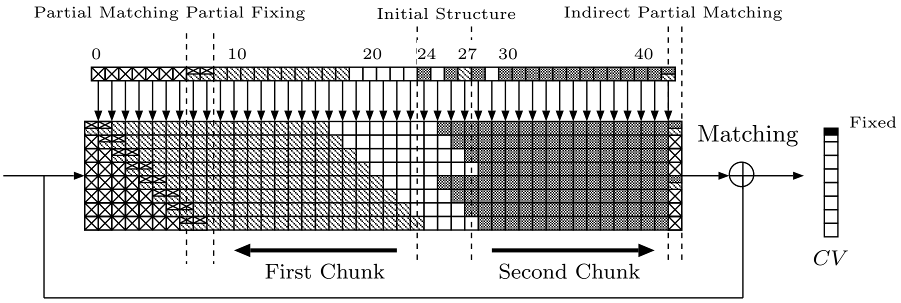
4.4 Pseudo Collision Attack on 43-step SHA-256

As discussed in Section 3.3, to convert a MITM preimage attack into a pseudo collision attack, the matching point is located into the end of the compression function, i.e., the addition of the feed-forward. As mentioned in section 4.3, the matching point of the 43-step MITM preimage attack is selected at the state after the step 37 ($j = 3$) due to the padding bits.

However, for a (pseudo) collision attack, we do not need to control message words for satisfying the padding rules, since we can generate correct padding by simply adding another message block as discussed in Section 3.3. It means that the last block of a compression function is used only for satisfying the padding condition in the collision attack when pseudo collision can be found before the last compression function as shown in Fig. 3. As a result, for a (pseudo) collision attack, we can move the matching point to the state after the step 43 ($j = 9$) that is the end of the compression function. [5]

Let a 256-bit output of the compression function be $CV = \{Z_A || \cdots || Z_H\}$, where each word is 32 bits. For $j = 9$, W_{24} and W_{27} are neutral words, and the matching point is the lower 4 bits of $A_{43} (= A_0 \oplus Z_A)$.

In order to construct the pseudo collision attack, we give the efficient method to obtain 4-bit partial target preimages by using the MITM technique [4]. Figure 4 shows the overview of the 43-step pseudo collision attack.



▧ : variables not depending on W_{24} ▩ : variables not depending on W_{27}

☒ : variables depending on both W_{24} and W_{27} ▦ : a few bits of variables depending only on W_{27}

▤ : variables that can be expressed as a sum module 2^{32} of W_{24} and W_{27}

Fig. 4. 43-step pseudo collision attack on SHA-256

Attack Procedure.

1. Choose the lower 4 bits of Z_A, which are target values.
2. Randomly choose the value of p_{25} and message W_{25}. Randomly fix the lower 23 bits of W_{24}. Then we can find 2^5 values of W_{24} on average from 9 free bits that correctly construct the 4-step initial structure and store them in the table T_W.
3. Randomly choose message words not related to the initial structure and the neutral words, i.e., W_{19}, W_{20}, W_{21}, W_{22}, W_{23} and W_{29} (called an initial configuration).
4. For all 2^5 possible W_{24} in T_W, compute W_{26}, W_{28}, W_{30}, W_{31}, W_{32}, W_{33} and W_{34} following Eq. (1). Compute forward and find $\psi_F(W_{24})$. Then, store the pairs (W_{24}, $\psi_F(W_{24})$) in a list L_F.
5. For all 2^4 possible values (the lower 4 bits) of W_{27}, compute backward and find $\xi_F(W_{27})$ and the lower 4 bits of A_0. Then, store the pairs $(W_{27}, Z_A \oplus A_0 - \sigma_0(W_{27}))$ in a list L_B.
6. If a match is found, i.e., $\psi_F(W_{24}) = Z_A \oplus A_0 - \sigma_0(W_{27})$, then compute two group of states $A_{43}, B_{43}, \cdots, H_{43}$ and $A_0, B_0, \cdots, H_0$ with corresponding W_{24} and W_{27}, respectively. Then obtain 2^5 $(= 2^9/2^4)$ CV whose 4-bit are fixed, i.e., the lower 4 bits of Z_A, and store these in a List L_1.
7. Repeat (3)-(6) 2^{121} times with different values of the initial configuration.

After the above procedures, we obtain 2^{126} $(= 2^5 \times 2^{121})$ pairs whose 4 bits are fixed.[6] Thus, there exists a colliding pair with a high probability, because of the equation of $(2^{126} = 2^{(256-4)/2})$.

Evaluation. We assume that the complexity for the 1-step function and the 1-step message expansion is 1/43 compression function operation of the 43-step SHA-256. As estimated in [10], the complexity of Step 2 in the presented attack is 2^9, and that of Steps 3-6 is $2^{4.878}$, which is the complexity for finding 2^5 4-bit partial target preimages. Thus, whole complexity of the pseudo collision attack on the 43-step SHA-256 is estimated as $2^{126} \approx 2^9 + (2^{121} \times 2^{4.878})$.

4.5 Known MITM Preimage Attack on 46-step SHA-512 [4]

The MITM preimage attack on the 46-step SHA-512 presented in [4] uses the 31-step two chunk $W_j, \ldots, W_{j+30}$ including the 2-step IS, the 8-step PF for $W_{j-1}, \ldots, W_{j-6}$ and W_{j+31},W_{j+32} and the 7-step PM. In this attack, we can choose j as long as 39 step (W_{j-6} to W_{j+32}) are located

[5] It is also pointed out in [10] as the matching point can be rotated to the end of the compression function

[6] It is noted that we need a slightly more than 2^{121} times repeated experiments to get 2^{126} pairs that will achieve a probability higher than 2^{-1}. However the difference is so small that we ignore it here.

sequentially. For the actual attack in [10], j is chosen as $j = 6$ to satisfy the padding rule. Then, the neutral words W_{21} and W_{22} have 4 and 3-bit freedom degrees, respectively, and the bit size of the matching point is 3. Thus, a preimage of the 46-step SHA-512 is found with the complexity of $2^{511.5}$. See [4] for more details about this attack.

4.6 Pseudo Collision Attack on 46-step SHA-512

Similarly to the attack on the reduced SHA-256, we can move the matching point to the end of the compression function, because the padding issue can be avoided by using multi-block message technique in the pseudo collision attack. In the case of SHA-512, since the bit size of the matching point is 3, we utilize the 3-bit partial target preimages for the attack. Then, the complexity of the attack is estimated as $2^{254.5} = (2^{(512-3)/2})$.

4.7 Pseudo Collision Attacks on 42-step SHA-256 and 42-step SHA-512

We consider pseudo collision attacks on smaller number of rounds of SHA-2 in order to save the time complexity. For the 42-step reduced SHA-256, we can use 10 bits of freedom in both directions to find a 10-bit partial target preimage as discussed in Section 5.4 of [4]. This implies that a 10-bit partial target preimage is obtained with the complexity 1 ($< 2^5$). Thus, a pseudo collision is found with the complexity of $2^{123}(= 2^{(256-10)/2} \times 2^{10}/2^{10})$. Similarly to this, for the 42-step reduced SHA-512, we can use 24 bits of freedom in both directions to find a 24-bit partial target preimage as discussed in Section 6.5 of [4]. Therefore, a pseudo collision of the 42-step reduced SHA-512 is found with the complexity of $2^{244}(= 2^{(512-24)/2} \times 2^{24}/2^{24})$.

4.8 Pseudo Collision Attacks on Reduced SHA-224 and SHA-384

The pseudo collision attack on the 43-step SHA-256 described in Section 4.4 is applicable to the 43-step SHA-224 in the similar manner. However, we can not use the multi-block message technique straightforwardly, because the pseudo collision attack on SHA-224 needs to be done in the last compression function whose output Z_H is disregarded. Thus, due to the padding issue, we can mount only pseudo collision attack on a compression function of 43-step, not a hash function. The estimated complexity is 2^{110} for this attack.

However, the smaller number of rounds of SHA-224 hash function can be attacked by using another MITM attack. The 40-step SHA-224 hash function can be attacked by using the same two chunks for the 43-step preimage attack on SHA-256 in [4], i.e., the case of $j = 3$. The 7-step partial matching for backward computation are replaced by the 4-step one. Then the message words W_{13}, W_{14} and W_{15} are left as free message words to satisfy the padding rule. Instead of the lower 4 bits of Z_A, we use the lower 4 bits of Z_D as the target value. Here, we need additional one step: when finding matches at the lower 4 bits of A_{37}, we compute forward from the matching point to the end of the compression function (40-th step) by using these values that are computed forward from the starting point. Since $A_{37} = D_{40} = D_0 \oplus Z_D$ for the 40-step SHA-224, the lower 4 bits of Z_D will keep unaffected by the additional step. Thus, we can still get a partial target preimage. It can be converted into a pseudo collision attack on a hash function, because we can set W_{13}, W_{14} and W_{15} to follow the padding rule.

The detail of the attack procedure is as follows.

1. Choose the lower 4 bits of Z_D, which are target values.
2. Randomly choose the value of p_{19} and message W_{19}. Randomly fix the lower 23 bits of W_{18}. Then we can find 2^5 values of W_{18} on average from 9 free bits that correctly construct the 4-step initial structure and store them in the table T_W.

3. Randomly choose message words not related to the initial structure and the neutral words, i.e., W_{13}, W_{14}, W_{15}, W_{16}, W_{17}, W_{23} (called an initial configuration [4]).
4. For all 2^5 possible W_{18} in T_W, compute W_{20}, W_{22}, W_{24}, W_{25}, W_{26}, W_{27}, W_{28} following Eq, (1). Compute forward and find $\psi_F(W_{18})$. Store the pairs (W_{18}, $\psi_F(W_{18})$) in a list L_F.
5. For all 2^4 possible values (the lower 4 bits) of W_{21}, compute backward and find $\xi_F(W_{21})$ and the lower 4 bits of A_{37} $(= D_{40} = Z_D \oplus D_0)$. Store the pairs $(W_{21}, Z_D \oplus D_0 - \sigma_0(W_{27}))$ in a list L_B.
6. If a match is found, i.e., $\psi_F(W_{24}) = Z_D \oplus D_0 - \sigma_0(W_{27})$, then compute forward to get the states $A_{40}, B_{40}, \cdots, H_{40}$ with corresponding W_{24} and W_{27}, respectively. D_{40} will keep unaffected in this step. Then obtain 2^5 $(= 2^9/2^4)$ CV whose 4 bits are fixed, i.e., the lower 4 bits of Z_D, and store these in a List.
7. Repeat (3)-(6) 2^{105} times with different values of the initial configuration.

The complexity of the attack is estimated as 2^{110}.

Similarly, the pseudo collision attack on the 46-step SHA-512 hash function described in 4.6 can also be applied to the 46-step SHA-384 compression function with the complexity of $2^{190.5} = (2^{(384-3)/2})$. For a pseudo collision attack on the reduced SHA-384 hash function, we use the 43-step preimage attack on SHA-384 [4]. Combining the result in [4] with our conversion technique, a pseudo collision attack on the 40-step SHA-384 hash function can be constructed. The matching bit is 18 when chosen parameter of partial matching as $\ell = 27$. The complexity of the pseudo collision attack on the 40-step SHA-384 is estimated as $2^{(384-18)/2} = 2^{183}$. These 40-step pseudo collision attacks give examples that the matching point is not at but near the end of compression function. That is compatible to solve padding problem.

4.9 Application to Other Results of SHA-2

Recently, the MITM preimage attacks on the reduced SHA-2 are improved by using "bicliques" technique which is considered as generalized initial structure [14]. This technique enables us to construct longer initial structures than those of the attacks [4]. In the following, let us consider pseudo collision attacks based on [14].

For SHA-256, the 36-step two independent chunks including the 6-step IS based on bicliques are constructed. Combining the 2-step PM with the 7-step PM and the 1-step IPM, the MITM preimage attack on the 45-step SHA-2 is derived. In this attack, both neutral words have 3-bit freedom degrees, and the matching point is 4-bit. Since our conversion technique does not need to consider the padding issue, the matching point can be moved to the end of the compression function similar to the 43-step attack. Then, we can convert it into the 45-step pseudo collision attack on SHA-256 with the complexity of $2^{126.5}$ $(= 2^{(256-3)/2})$ [7]. Similarly, we can construct the 50-step pseudo collision attack on SHA-512 based on the 50-step MITM preimage attack [14]. In this attack, both neutral words have 3-bit freedom degrees, and the bit size of the matching point is 3. Thus, the complexity of the attack is estimated as $2^{254.5}$ $(= 2^{(512-3)/2})$.

In addition, [14] showed pseudo preimage attacks on the 52-step SHA-256 and the 57-step SHA-512. For the setting of a pseudo preimage attack, the cost of converting a pseudo preimage to a preimage is omitted. Thus, larger number of rounds can be attacked. Note that in these attacks, the amount of freedom degrees for both neutral words are only 1-bit, and the bit size of the matching point is 1. In order to construct a pseudo collision attack by using our conversion technique, it is sufficient to obtain a pseudo preimage on a compression function, i.e., a preimage on a hash function is not needed. Therefore, the above explained pseudo preimage attacks can also be converted into pseudo collision attacks in a similar way. The complexities of

[7] Our attack uses only 3 bits for the matching and find 3-bit partial target preimages, because this setting is optimal with respect to the time complexity.

the pseudo collision attacks on the 52-step SHA-256 and the 57-step SHA-512 are estimated as $2^{127.5}$ $(= 2^{(256-1)/2})$ and $2^{255.5}$ $(= 2^{(512-1)/2})$, respectively.

5 Application to Skein

In this section, we show pseudo collision attacks on the reduced Skein-512 [9] based on the preimage attacks presented in [14].

5.1 Description of Skein

Skein is built from the tweakable block cipher Threefish $E_{K,T}(P)$, where K, T and P denote a key, a tweak and a plaintext message, respectively. The compression function $F(CV, T, M)$ of Skein outputs the next chaining variable as $F(CV, T, M) = E_{CV,T}(M) \oplus M$, where CV is the previous chaining variable and M is an input message block.

Threefish-512 supports a 512-bit block and a 512-bit key, and operates on 64-bit words. The subkey $K^s = (K^s_0, K^s_1, \ldots, K^s_7)$ injected every four rounds is generated from the secret key $K = K[0], K[1], \ldots, K[7]$ as follows:

$$K^s_j = K[(s + j) \bmod 9], (0 \leq j \leq 4); \qquad K^s_5 = K[(s + 5) \bmod 9] + T[s \bmod 3];$$
$$K^s_6 = K[(s + 6) \bmod 9] + T[(s + 1) \bmod 3]; \quad K^s_7 = K[(s + 7) \bmod 9] + s,$$

where s denotes a round counter, $T[0]$ and $T[1]$ denote tweak words, $T[2] = T[0] + T[1]$, and $K[8] = C_{240} \oplus \bigoplus_{j=0}^{7} K[j]$ with a constant C_{240}. Each Threefish-512 round consists of four MIX functions followed by a permutation of the eight 64-bit words. The 128-bit function MIX processes the pairs of eight words of internal state $I^0, I^1, \ldots, I^7$ after key addition.

5.2 Known Pseudo Preimage Attacks on Skein [14].

We briefly review two MITM preimage attacks on Skein-512 presented in [14]: one is a preimage attack on the 22-round reduced Skein-512 hash function starting from the 3rd round, and the other is a preimage attack on the 37-round reduced Skein-512 compression function starting from the 2nd round.

For the 22-round attack, the 3-dimension biclique at rounds 12-15 is obtained with the complexity of 2^{200}. Since many bicliques can be produced out of one, the cost of constructing the bicliques is negligible in the total complexity of the attack. In this attack, we can obtain 2^3 pairs matched in 3 bits by $2^{2.3}$ calls of the 22-round Skein-512 compression function. As a result, a preimage of the 22-round reduced Skein is found with the complexity of $2^{511.2}$.

Table 2. Parameters of the (pseudo) preimage attacks on the reduced Skein-512 [14]

Parameters of the preimage attack on the 22-round Skein-512 hash function

Chunks			Matching			
Forward	Backward	Biclique	Partial matching	Matching bits	Total matching pairs	Complexity
8-11	16-19	12-15	$20 \rightarrow 24 = 3 \leftarrow 7$	$I^1_{30,31,53}$	2^3	$2^{2.3}$

Parameters of the pseudo preimage attack on the 37-round Skein-512 compression function

Chunks			Matching			
Forward	Backward	Biclique	Partial matching	Matching bits	Total matching pairs	Complexity
8-15	24-31	16-23	$32 \rightarrow 38 = 2 \leftarrow 7$	I^3_{25}	2	$2^{1.2}$

Considering a pseudo preimage attack on the compression function, it is natural to assume that tweak bits T can also be controlled by the attacker. Due to additional freedom, the pseudo preimage attack on the 37-round reduced Skein-512 is feasible by using the 1-dimension biclique at rounds 16-23. In this attack, we can obtain 2 pairs matched in 1 bit by $2^{1.2}$ calls of the 37-round Skein-512 compression function. Consequently, a pseudo preimage of the 37-round reduced Skein is found with the complexity of $2^{511.2}$.

The parameters for the preimage attacks on the 22-round and the 37-round reduced Skein-512 hash function and compression function are summarized in Table 2. See [14] for more details about this attack.

5.3 Pseudo Collision Attacks on Skein.

Since the matching point used in the MITM preimage attack on the 22-round reduced Skein-512 hash function [14] is located in the end of the compression function, our conversion technique can directly convert it to the pseudo collision attack on the 22-round reduced Skein-512. In this attack, the neutral words have 3-bit freedom degrees, and the bit size of the matching point is 3. As reported in [14], a 3-bit matching candidate can be found with the complexity of $2^{2.3}/2^3$. Thus, the complexity of the pseudo collision attack on the 22-round reduced Skein-512 hash function is estimated as $2^{253.8}$ ($= 2^{(512-3)/2} \times 2^{2.3}/2^3$).

The pseudo preimage attack on the 37-round reduced Skein compression function can be converted into a pseudo collision attack on a hash function in a similar way. The required complexity for the pseudo collision attack on the 37-round reduced Skein hash function is estimated as $2^{255.7}$ ($= 2^{(512-1)/2} \times 2^{1.2}/2$).

6 Conclusion

In this paper, we gave a generic method to convert preimage attacks to pseudo collision attacks. It provides a new insight to evaluate the security of hash functions. The essence of the method is converting a partial target preimage attack to a pseudo collision attack. That is especially compatible to meet-in-the-middle preimage attacks since it can be converted into a partial target preimage attack if the matching point can be moved to the end of a hash function or a compression function and enough freedom on neutral bits are left.

Using the proposed approach, we presented the best pseudo collision attacks on SHA-2 based on the known preimage attacks, which has been left as open question. We showed pseudo collision attacks on the 43- and 46-step reduced SHA-256 and SHA-512 based on the MITM preimage attacks presented in [4]. Also, pseudo collision attacks on the 52- and 57-step reduced SHA-256 and SHA-512 based on the more advanced MITM preimage attacks in [14] were demonstrated. We also applied the conversion technique to other hash functions including Skein and BLAKE with the meet-in-the-middle preimage attacks, that showed the widely usage of this method. The pseudo collision attacks on the 22- and 37-round reduced Skein-512 were presented. The 4-round reduced BLAKE-256/512 without the initialization function can be attacked by the converted pseudo collision attack (see Appendix A). Our technique can also apply to other hash functions, such as Tiger [1]. Based on the MITM preimage attack on the full Tiger [10], we might construct the pseudo collision attack on the full Tiger. We believe that the technique can be used for more hash algorithms once their preimage or pseudo preimage attacks are found.

By this method, now we only can get pseudo collision attacks. It is left as future works that how to construct collision attacks from known preimage attacks.

Acknowledgments The author would like to thank the anonymous reviewers for their helpful comments.

References

1. R. J. Anderson and E. Biham, "Tiger: A fast new hash function." in *FSE* (D. Gollmann, ed.), vol. 1039 of *Lecture Notes in Computer Science*, pp. 89–97, Springer, 1996.
2. K. Aoki and Y. Sasaki, "Preimage attacks on one-block MD4, 63-step MD5 and more." in *Selected Areas in Cryptography* (R. Avanzi, L. Keliher, and F. Sica, eds.), vol. 5381 of *Lecture Notes in Computer Science*, pp. 103–119, Springer, 2009.
3. K. Aoki and Y. Sasaki, "Meet-in-the-middle preimage attacks against reduced SHA-0 and SHA-1." in *CRYPTO* (S. Halevi, ed.), vol. 5677 of *Lecture Notes in Computer Science*, pp. 70–89, Springer, 2009.
4. K. Aoki, J. Guo, K. Matusiewicz, Y. Sasaki, and L. Wang, "Preimages for step-reduced SHA-2." in *ASIACRYPT* (M. Matsui, ed.), vol. 5912 of *Lecture Notes in Computer Science*, pp. 578–597, Springer, 2009.
5. J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan, "SHA-3 proposal BLAKE (version 1.3)." Submission to NIST, Dec. 2010. Available at http://131002.net/blake/blake.pdf.
6. A. Biryukov, M. Lamberger, F. Mendel, and I. Nikolic, "Second-order differential collisions for reduced SHA-256." in *ASIACRYPT* (D. H. Lee and X. Wang, eds.), vol. 7073 of *Lecture Notes in Computer Science*, pp. 270–287, Springer, 2011.
7. A. Bogdanov and C. Rechberger, "A 3-subset meet-in-the-middle attack: Cryptanalysis of the lightweight block cipher KTANTAN." in *Selected Areas in Cryptography* (A. Biryukov, G. Gong, and D. R. Stinson, eds.), vol. 6544 of *Lecture Notes in Computer Science*, pp. 229–240, Springer, 2010.
8. C. D. Cannière and C. Rechberger, "Preimages for reduced SHA-0 and SHA-1." in *CRYPTO* (D. Wagner, ed.), vol. 5157 of *Lecture Notes in Computer Science*, pp. 179–202, Springer, 2008.
9. N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker, "The Skein hash function family.".
10. J. Guo, S. Ling, C. Rechberger, and H. Wang, "Advanced meet-in-the-middle preimage attacks: First results on full Tiger, and improved results on MD4 and SHA-2." in *ASIACRYPT* (M. Abe, ed.), vol. 6477 of *Lecture Notes in Computer Science*, pp. 56–75, Springer, 2010.
11. S. Indesteege, F. Mendel, B. Preneel, and C. Rechberger, "Collisions and other non-random properties for step-reduced SHA-256." in *Selected Areas in Cryptography* (R. Avanzi, L. Keliher, and F. Sica, eds.), vol. 5381 of *Lecture Notes in Computer Science*, pp. 276–293, Springer, 2009.
12. T. Isobe, "A single-key attack on the full GOST block cipher." in *FSE* (A. Joux, ed.), vol. 6733 of *Lecture Notes in Computer Science*, pp. 290–305, Springer, 2011.
13. T. Isobe and K. Shibutani, "Preimage attacks on reduced Tiger and SHA-2." in *FSE* (O. Dunkelman, ed.), vol. 5665 of *Lecture Notes in Computer Science*, pp. 139–155, Springer, 2009.
14. D. Khovratovich, C. Rechberger, and A. Savelieva, "Bicliques for preimages: Attacks on Skein-512 and the SHA-2 family." in *FSE'12 (to appear)*, Lecture Notes in Computer Science, Springer-Verlag, 2012.
15. X. Lai and J. L. Massey, "Hash function based on block ciphers." in *EUROCRYPT*, pp. 55–70, 1992.
16. G. Leurent, "MD4 is not one-way." in *Fast Software Encryption* (K. Nyberg, ed.), vol. 5086 of *Lecture Notes in Computer Science*, pp. 412–428, Springer, 2008.
17. F. Mendel, T. Nad, and M. Schläffer, "Finding SHA-2 characteristics: Searching through a minefield of contradictions." in *ASIACRYPT* (D. H. Lee and X. Wang, eds.), vol. 7073 of *Lecture Notes in Computer Science*, pp. 288–307, Springer, 2011.
18. F. Mendel, N. Pramstaller, C. Rechberger, and V. Rijmen, "Analysis of step-reduced SHA-256." in *FSE* (M. J. B. Robshaw, ed.), vol. 4047 of *Lecture Notes in Computer Science*, pp. 126–143, Springer, 2006.
19. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1997.
20. I. Nikolić and A. Biryukov, "Collisions for step-reduced SHA-256." in *Fast Software Encryption* (K. Nyberg, ed.), vol. 5086 of *Lecture Notes in Computer Science*, pp. 1–15, Springer, 2008.
21. J.-J. Quisquater and J.-P. Delescaille, "How easy is collision search? application to DES (extended summary)." in *EUROCRYPT*, pp. 429–434, 1989.
22. M.-J. O. Saarinen, "A meet-in-the-middle collision attack against the new FORK-256." in *INDOCRYPT* (K. Srinathan, C. P. Rangan, and M. Yung, eds.), vol. 4859 of *Lecture Notes in Computer Science*, pp. 10–17, Springer, 2007.
23. S. K. Sanadhya and P. Sarkar, "22-step collisions for SHA-2." *CoRR*, vol. abs/0803.1220, 2008.
24. S. K. Sanadhya and P. Sarkar, "Attacking reduced round SHA-256." in *ACNS* (S. M. Bellovin, R. Gennaro, A. D. Keromytis, and M. Yung, eds.), vol. 5037 of *Lecture Notes in Computer Science*, pp. 130–143, 2008.
25. S. K. Sanadhya and P. Sarkar, "New collision attacks against up to 24-step SHA-2." in *INDOCRYPT* (D. R. Chowdhury, V. Rijmen, and A. Das, eds.), vol. 5365 of *Lecture Notes in Computer Science*, pp. 91–103, Springer, 2008.
26. S. K. Sanadhya and P. Sarkar, "Non-linear reduced round attacks against SHA-2 hash family." in *ACISP* (Y. Mu, W. Susilo, and J. Seberry, eds.), vol. 5107 of *Lecture Notes in Computer Science*, pp. 254–266, Springer, 2008.
27. Y. Sasaki and K. Aoki, "Finding preimages in full MD5 faster than exhaustive search." in *EUROCRYPT* (A. Joux, ed.), vol. 5479 of *Lecture Notes in Computer Science*, pp. 134–152, Springer, 2009.

28. Y. Sasaki and K. Aoki, "Preimage attacks on 3, 4, and 5-pass HAVAL." in *ASIACRYPT* (J. Pieprzyk, ed.), vol. 5350 of *Lecture Notes in Computer Science*, pp. 253–271, Springer, 2008.
29. R. Sedgewick, T. G. Szymanski, and A. C.-C. Yao, "The complexity of finding cycles in periodic functions." *SIAM J. Comput.*, vol. 11, no. 2, pp. 376–390, 1982.
30. L. Wang, K. Ohta, and K. Sakiyama, "Free-start preimages of round-reduced Blake compression function." Rump session at ASIACRYPT 2009.
31. X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full SHA-1." in *CRYPTO* (V. Shoup, ed.), vol. 3621 of *Lecture Notes in Computer Science*, pp. 17–36, Springer, 2005.
32. X. Wang and H. Yu, "How to break MD5 and other hash functions." in *EUROCRYPT* (R. Cramer, ed.), vol. 3494 of *Lecture Notes in Computer Science*, pp. 19–35, Springer, 2005.
33. D. Watanabe, "OFFICIAL COMMENT: LUX." NIST mailing list, 2009. Available at `http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/documents/LUX_Comments.pdf`.

Appendix

A Application to BLAKE

We apply our technique to BLAKE hash family consisting of BLAKE-224, BLAKE-256, BLAKE-384 and BLAKE-512 [5]. We utilize the result presented in [30] which showed a pseudo preimage attack on the 4-round reduced BLAKE compression function without the initialization function. While the practical impact on the attack on this reduced BLAKE compression function is debatable, a pseudo collision on the reduced BLAKE can be directly derived by using our conversion technique. We can find pseudo collision of BLAKE-256 compression function for reduced 4 rounds with the complexity of 2^{112}. For BLAKE-512, the complexity is 2^{224} for reduced 4 rounds compression function.

A.1 Description of BLAKE

The compression function of BLAKE-256 consists of $initialization$, round function and $finalization$.

Initialization : 8 words of chaining value $h_0, \ldots, h_7$ are transformed into 16 words of an initial state $v_0, \ldots, v_{15}$ as $v_i = h_i$ for $0 \leq i \leq 7$. While v_i $(8 \leq i \leq 15)$ are obtained from the salts and the counter, we ignore the details for the simplicity.

Round function : An initial state v is updated by 14 round functions. Each round function includes the following steps, $G_0(v_0,\ v_4,\ v_8,\ v_{12})$, $G_1(v_1,\ v_5,\ v_9,\ v_{13})$, $G_2(v_2,\ v_6,\ v_{10},\ v_{14})$, $G_3(v_3,\ v_7,\ v_{11},\ v_{15})$, $G_4(v_0,\ v_5,\ v_{10},\ v_{15})$, $G_5(v_1,\ v_6,\ v_{11},\ v_{12})$, $G_6(v_2,\ v_7,\ v_8,\ v_{13})$, $G_7(v_3,\ v_4, v_7,\ v_{14})$. The function $G_i(a, b, c, d)$ is defined as:

$$\begin{array}{ll} a \leftarrow a + b + (m_{\sigma_r(2i)} \oplus c_{\sigma_r(2i+1)}), & d \leftarrow (d \oplus a) \ggg 16, \\ c \leftarrow c + d, & b \leftarrow (b \oplus c) \ggg 12, \\ a \leftarrow a + b + (m_{\sigma_r(2i+1)} \oplus c_{\sigma_r(2i)}), & d \leftarrow (d \oplus a) \ggg 8, \\ c \leftarrow c + d, & b \leftarrow (b \oplus c) \ggg 7, \end{array}$$

where permutations $\sigma_r(j)$ $(0 \leq j < 16)$ of the first 4 rounds refer to Table 3. The functions G_0 to G_3 and G_4 to G_7 denote the column transfroms and the diagonal transforms, respectively.

Table 3. Message and Constants Permutation

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
σ_0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
σ_1	14	10	4	8	9	15	13	6	1	12	0	2	11	7	5	3
σ_2	11	8	12	0	5	2	15	13	10	14	3	6	7	1	9	4
σ_3	7	9	3	1	13	12	11	14	2	6	5	10	4	0	15	8

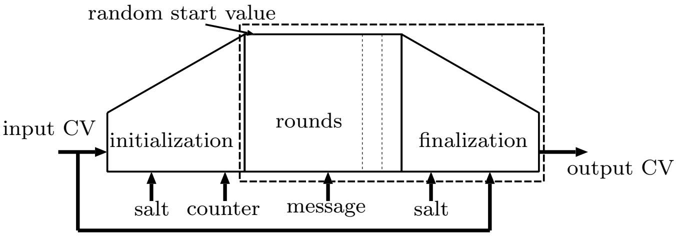


Fig. 5. MITM preimage attack for finalization

Finalization : After the round functions, the new chaining value is extracted with the updated state, the salt and the feed-forward of the initial chaining value as follows.

$$
\begin{array}{ll}
h_0^{'} \leftarrow h_0 \oplus s_0 \oplus v_0 \oplus v_8 & h_1^{'} \leftarrow h_1 \oplus s_1 \oplus v_1 \oplus v_9 \\
h_2^{'} \leftarrow h_2 \oplus s_2 \oplus v_2 \oplus v_{10} & h_3^{'} \leftarrow h_3 \oplus s_3 \oplus v_3 \oplus v_{11} \\
h_4^{'} \leftarrow h_4 \oplus s_0 \oplus v_4 \oplus v_{12} & h_5^{'} \leftarrow h_5 \oplus s_1 \oplus v_5 \oplus v_{13} \\
h_6^{'} \leftarrow h_6 \oplus s_2 \oplus v_6 \oplus v_{14} & h_7^{'} \leftarrow h_7 \oplus s_3 \oplus v_7 \oplus v_{15}
\end{array}
$$

BLAKE-512 operates on 64-bit words and outputs 512 bits. The compression function of BLAKE-512 is similar to that of BLAKE-256 except the number of rounds (16 instead of 14), and the constants and the amount of rotation used in G functions.

A.2 Known MITM Preimage Attacks on 4-round Compression Function of BLAKE [30].

In the setting of the pseudo preimage of the compression function presented in [30], the initialization step is disregarded, and selected a random start value from the start of round functions (the end of initialization step) as shown in Fig. 5.

Figure 6 shows the overview of the pseudo preimage attacks on the 4-round reduced BLAKE compression function without the initialization. Let an input state of the round i be $\{v_0^{i-1}, \dots, v_{15}^{i-1}\}$. In this attack, message words m_4 and m_6 are used as the neutral words, and the starting point of the attack is the state after the column transformation of the round 3. In the forward computation from the starting point, v_6^4, v_{14}^4 can be computed without using m_6. Similarly, in the backward computation, v_6^0 can be computed without using m_4. Therefore, stroing m_4, v_6^4, v_{14}^4 in a list L_F, and m_6, v_6^0 in a list L_B, we expect to find matching pairs satisfying $h_6' = v_6^0 \oplus v_6^4 \oplus v_{14}^4$. As a result, a pseudo preimage of the 4-round reduced BLAKE without the initialization is found with the complexity of 2^{224}.

A.3 Pseudo Collision Attacks on BLAKE Compression Function.

Since the matching point of the known pseudo preimage attack is at the end of the compression function, we can directly use it to construct a pseudo collision attack.

Attack Procedure.

1. Random choose the 7-th word words of the output value h_6', which is the target value.
2. Random choose the values of state words and message words except of m_4 and m_6.
3. For all 2^{32} possible m_4, compute forward and find v_6^4 and v_{14}^4. Store the pairs $(m_4,\ v_6^4 \oplus v_{14}^4)$ in a list L_F
4. For all 2^{32} possible m_6, compute forward and find v_6^0. Store the pairs $(m_4,\ h_6' \oplus v_6^0)$ in a list L_B.

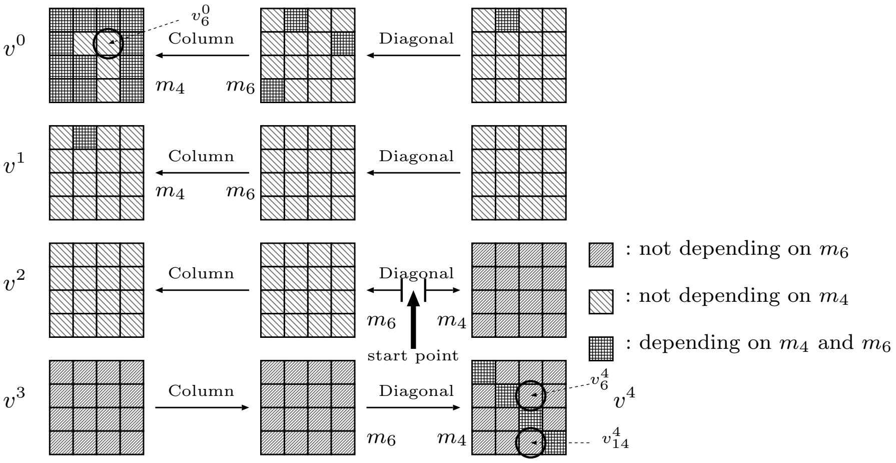


Fig. 6. Pseudo preimage attacks on reduced BLAKE compression function

5. Compare the value $v^4_6 \oplus v^4_{14}$ and $h'_6 \oplus v^0_6$ in two lists L_F and L_B.
6. Once matching, compute states $v^0_0, v^0_1, \cdots, v^0_{15}$ and $v^4_0, v^4_1, \cdots, v^4_{15}$. Compute output values $h'_0, h'_1, \ldots, h'_{15}$ according to finalization steps and store with message words together. Then obtain 2^{32} items in which the value of h'_6 are fixed.
7. Repeat steps (2) - (6) 2^{80} times.

We can obtain 2^{112} items in which the value of h'_6 are fixed. A colliding pair exists with a high probability that the other 224 bits of output values are also same. Finally, we can find a pseudo collision of the compression function for the 4-round reduced BLAKE-256 with the complexity of $2^{112} = 2^{80} \cdot 2^{32}$.

The attack is applicable to the reduced BLAKE-512 in a similar way, since the components of BLAKE-512 are similar to those of BLAKE-256. In BLAKE-224, the variable h'_7 is truncated and discarded. However, the truncation does not affect our convertion, since we use h'_6 as a partial target preimage. Thus, a pseudo collision attack on the 4-round reduced BLAKE-224 without the initialization can be constructed with the complexity of $2^{96}(= 2^{(224-32)/2})$. For BLAKE-512, in contrast to the other variants, the variable h'_6 is discarded by the truncation as well. Therefore, it is hard to straightforwardly apply our conversion to the reduced BLAKE-512, since h'_6 cannot be used as a partial target preimage.

UNAF: A Special Set of Additive Differences with Application to the Differential Analysis of ARX[⋆]

Vesselin Velichkov[1,2,3,⋆⋆], Nicky Mouha[1,3,⋆⋆⋆], Christophe De Cannière[1,3,†], and Bart Preneel[1,3]

[1] Department of Electrical Engineering ESAT/SCD-COSIC, Katholieke Universiteit Leuven.
Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium.
[2] Laboratory of Algorithmics, Cryptology and Security (LACS), University of Luxembourg, Luxembourg.
[3] Interdisciplinary Institute for BroadBand Technology (IBBT), Belgium.
{Vesselin.Velichkov,Nicky.Mouha}@esat.kuleuven.be

Abstract. Due to their fast performance in software, an increasing number of cryptographic primitives are constructed using the operations addition modulo 2^n, bit rotation and XOR (ARX). However, the resistance of ARX-based ciphers against differential cryptanalysis is not well understood. In this paper, we propose a new tool for evaluating more accurately the probabilities of additive differentials over multiple rounds of a cryptographic primitive. First, we introduce a special set of additive differences, called UNAF (unsigned non-adjacent form) differences. Then, we show how to apply them to find good differential trails using an algorithm for the automatic search for differentials. Finally, we describe a key-recovery attack on stream cipher Salsa20 reduced to five rounds, based on UNAF differences.

Keywords: UNAF, ARX, Salsa20, additive differential probability, differential cryptanalysis

1 Introduction

Differential cryptanalysis [4] and linear cryptanalysis [14] have shown to be two of the most powerful techniques in the cryptanalysis of symmetric-key cryptographic primitives. Security against linear and differential cryptanalysis is therefore typically a major design criterion for modern ciphers. An example of this is the wide-trail design strategy, used to provide provable resistance against linear and differential cryptanalysis for the AES block cipher [6].

In order to achieve a fast performance in software, an increasing number of cryptographic primitives are built using the operations addition modulo 2^n, rotation and XOR (ARX). Examples include the block cipher FEAL [17], the Salsa20 stream cipher family [3], as well as the SHA-3 finalists BLAKE [2] and Skein [9]. Although ARX-based algorithms are very popular, their resistance to differential cryptanalysis [4] is not well understood.

The probability with which differences propagate through a sequence of operations must be calculated efficiently and accurately, in order to correctly assess the security of a cipher against differential cryptanalysis. Lipmaa et al. studied the xor-differential probability of addition (xdp^{+}) in [12], and the additive differential probability of XOR ($\mathrm{adp}^{\oplus}$) in [13]. These results were generalized using the S-functions framework, introduced by Mouha et al. [15].

As shown by Velichkov et al. [18], the additive differential probability of ARX ($\mathrm{adp}^{\mathtt{ARX}}$) can differ significantly from the multiplication of the differential probability of the separate components – addition, rotation and XOR. Although an algorithm was proposed in [18] for the exact

[⋆] This work was supported in part by the Research Council K.U.Leuven: GOA TENSE, and by the IAP Program P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II.

[⋆⋆] DBOF Doctoral Fellow, K.U.Leuven, Belgium.

[⋆⋆⋆] This author is funded by a research grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

[†] Postdoctoral Fellow of the Research Foundation – Flanders (FWO).

Table 1. Notation.

Symbol	Meaning
n	Number of bits in a word
x	n-bit word
$x[i]$	Select the ($i \mod n$)-th bit (or element) of the n-bit word x, $x[0]$ is the least-significant bit (or element)
$\|x\|$	The absolute value of x
$\overline{x}$	The negation of x i.e. $\overline{x} = -x$ (e.g. $\overline{1} = -1$)
$\#A$	Number of elements in the set A
$+$, -	Addition modulo 2^n, subtraction modulo 2^n
$\oplus$	Exclusive-OR (`XOR`)
$\lll t$	Left bit rotation by t positions
$\alpha \rightarrow \beta$	Input difference α propagates to output difference β
w_i^r	32-bit word i from the input state to round $r+1$ of Salsa20
Δ_i^r	Additive difference in word i of the input to round $r+1$ of Salsa20
0_i^r	Zero difference in word i of the input to round $r+1$ of Salsa20
$\{\Delta^U\}_i^r$	UNAF difference in word i of the input to round $r+1$ of Salsa20
`ARX`	The sequence of the operations: $+, \lll, \oplus$ as a single operation
HW(x)	Hamming weight of x (number of non-zero bits in x)

calculation of adp$^{\texttt{ARX}}$, unfortunately their method does not scale to analyze larger components. The accurate calculation of the probability of a differential characteristic therefore still remains an open problem for `ARX` constructions.

In this paper we take a different approach. Namely, we do not calculate the *exact* differential probability of a component consisting of more than one `ARX` operations. Instead, we multiply the differential probabilities of several `ARX` operations in order to estimate the total probability. As we want to avoid that this calculation differs significantly from the actual probability (e.g. due to dependencies between the inputs as noted in [18]), we propose to use a new type of difference: the UNAF difference, which represents a set of specially chosen additive differences.

We apply UNAF differences to the cryptanalysis of the `ARX`-based stream cipher Salsa20. A general algorithm for automatic search of differentials is briefly discussed. We apply it to find several differentials for three rounds of Salsa20. By multiplying the probabilities adp$^{\texttt{ARX}}$ of separate `ARX` components, we estimate that the best differential has a probability of 2^{-10}. Using UNAF differences, the same probability is evaluated as 2^{-4}. Experimentally, we estimate the probability of this differential to be $2^{-3.39}$. We observe that the probability obtained using UNAF differences is much closer to the experimental value.

Finally, we apply UNAF differences to mount key-recovery attack on a version of Salsa20 reduced to 5 rounds. Note that this is not the best known attack on Salsa20. It is therefore provided only as a demonstration of a practical application of UNAF differences. Furthermore, we expect that our attack can be extended to more rounds.

The outline of the paper is as follows. In Sect. 2, we describe the UNAF framework. It is applied to the differential analysis of stream cipher Salsa20 in Sect. 3. Sect. 4 concludes the paper. Notation is defined in Table 1.

2 The UNAF Framework

In this section, we describe the UNAF framework. We define UNAF differences and state the main UNAF theorem. The UNAF differential probability of `ARX` (udp$^{\texttt{ARX}}$) is defined and a general algorithm for the automatic search for high-probability differentials is briefly discussed.

2.1 Preliminaries

Before we give the formal definition of UNAF differences, we first recall a few related concepts: the binary-signed digit (BSD) difference and the non-adjacent form (NAF) difference.

Definition 1. (BSD difference) *A BSD difference is a difference whose bits are signed and take values in the set* $\{\bar{1},0,1\}$*:*

$$\Delta^{\pm}a : \Delta^{\pm}a[i] = (a_2[i] - a_1[i]) \in \{\bar{1},0,1\}, \quad 0 \le i < n \ . \tag{1}$$

An additive difference $\Delta^{+}a$ can be composed of more than one BSD difference $\Delta^{\pm}a$. From any BSD difference, the corresponding additive difference can be computed as: $\Delta^{+}a = \sum_{i=0}^{n-1} \Delta^{\pm}a[i] \cdot 2^i$.

All BSD differences corresponding to $\Delta^{+}a$ can be obtained by replacing 01 with $1\bar{1}$ and vice versa and by replacing $0\bar{1}$ with $\bar{1}1$ and vice versa [7, 16]. Note also that the number of pairs (a_1, a_2) that satisfy the n-bit difference $\Delta^{+}a$ is 2^n, while the number of pairs that satisfy any of its BSD differences $\Delta^{\pm}a$ is 2^k, where k is the number of zeros in the word $\Delta^{\pm}a$. Therefore, the following inequality holds: $2^k \le 2^n$, $k = n - \mathrm{HW}(\Delta^{\pm}a)$.

The non-adjacent form (NAF) difference is a special BSD difference and is defined as follows:

Definition 2. (NAF) *A NAF (non-adjacent form) difference is a BSD difference in which no two adjacent bits are non-zero:*

$$\Delta^{N}a : \quad \nexists i : \quad (\Delta^{N}a[i] \ne 0) \wedge (\Delta^{N}a[i+1] \ne 0), \quad 0 \le i < n-1 \ . \tag{2}$$

For every additive difference $\Delta^{+}a$, there is exactly one NAF difference $\Delta^{N}a$ (ignoring the sign of the MSB). No other BSD difference has a lower Hamming weight than $\Delta^{N}a$ [16]. We illustrate this with the following example:

Example 1. Let $n = 4$ and $\Delta^{+}a = 3$. Then all possible BSD differences corresponding to $\Delta^{+}a$ are 0011, $010\bar{1}$, $01\bar{1}1$, $1\bar{1}\bar{1}1$, $\bar{1}\bar{1}\bar{1}1$, $1\bar{1}0\bar{1}$ and $\bar{1}10\bar{1}$. Of them, only $010\bar{1}$ is in non-adjacent form (NAF). It also has the lowest Hamming weight among all BSD differences, namely 2.

By enumerating all possible combinations of signs of the non-zero bits of $\Delta^{N}a$, we can construct a special set of additive differences. What is special about this set, is that all of its elements correspond to the same unsigned NAF difference. This set is a UNAF difference and is denoted by $\Delta^{U}a$. More formally:

Definition 3. (UNAF) *A UNAF difference is a set of additive differences that correspond to the same unsigned NAF difference (i.e. a NAF difference with the signs ignored):*

$$\Delta^{U}a = \{\Delta^{+}x : |\Delta^{N}x| = |\Delta^{N}a|\} \ . \tag{3}$$

It is easy to see that the size of the UNAF set $\Delta^{U}a$ is 2^k, where k is the Hamming weight of the n-bit word $\Delta^{N}a$, excluding the MSB. We further clarify the concept of a UNAF difference with the following example:

Example 2. Consider again an example where $n = 4$. Let $\Delta^{+}a = 3$, thus $\Delta^{N}a = 010\bar{1}$. Then, $\Delta^{U}a = \{\Delta^{+}x_1 = 3, \Delta^{+}x_2 = -3, \Delta^{+}x_3 = 5, \Delta^{+}x_4 = -5\}$. This follows from $|\Delta^{N}x_1| = |\Delta^{N}x_2| = |\Delta^{N}x_3| = |\Delta^{N}x_4| = |\Delta^{N}a|$, because $|010\bar{1}| = |0\bar{1}01| = |0101| = |0\bar{1}0\bar{1}| = 0101$.

2.2 Main UNAF Theorem

The main UNAF theorem provides the motivation for applying UNAF differences to the differential analysis of ARX. Before we state it, we define the additive differential probability of `XOR` ($\mathrm{adp}^{\oplus}$).

The differential probability of the operation `XOR`, when differences are expressed using addition modulo 2^n, is denoted by $\mathrm{adp}^{\oplus}$. For fixed additive differences α, β and γ, $\mathrm{adp}^{\oplus}$ is equal to the number of pairs (a_1, b_1) for which the equality $((a_1 + \alpha) \oplus (b_1 + \beta)) - (a_1 \oplus b_1) = \gamma$ holds, divided by the total number of such pairs. More formally, $\mathrm{adp}^{\oplus}(\alpha, \beta \to \gamma)$ is defined as:

Definition 4. ($\mathrm{adp}^{\oplus}$)

$$\begin{aligned}\mathrm{adp}^{\oplus}(\alpha, \beta \to \gamma) &= \frac{\#\{(a_1, b_1) : c_2 - c_1 = \gamma\}}{\#\{(a_1, b_1)\}} \\ &= 2^{-2n} \cdot \#\{(a_1, b_1) : c_2 - c_1 = \gamma\} \ ,\end{aligned} \tag{4}$$

where $c_1 = a_1 \oplus b_1$*,* $c_2 = (a_1 + \alpha) \oplus (b_1 + \beta)$ *and* 2^{2n} *is the total number of pairs* (a_1, b_1)*.*

Efficient algorithms for the computation of $\mathrm{adp}^{\oplus}$ were studied in [13, 15]. Next we state the main UNAF theorem. Its proof is given in Appendix A.

Theorem 1. (Main UNAF theorem) *If the probability with which input additive differences* $\Delta^{+}a$ *and* $\Delta^{+}b$ *propagate to output difference* $\Delta^{+}c$ *through* `XOR` *is non-zero, then the probability with which any of the input additive differences belonging to the corresponding UNAF sets resp.* $\Delta^{U}a$ *and* $\Delta^{U}b$ *propagate to any of the output additive differences belonging to the UNAF set* $\Delta^{U}c$ *is also non-zero:*

$$\begin{aligned}\mathrm{adp}^{\oplus}(\Delta^{+}a, \Delta^{+}b \to \Delta^{+}c) > 0 \implies \mathrm{adp}^{\oplus}(\Delta^{+}a_i, \Delta^{+}b_j \to \Delta^{+}c_k) > 0 \ , \\ \forall i, j, k : \Delta^{+}a_i \in \Delta^{U}a, \Delta^{+}b_j \in \Delta^{U}b, \Delta^{+}c_k \in \Delta^{U}c \ .\end{aligned} \tag{5}$$

Theorem 1 states that if a given additive differential is possible w.r.t. the `XOR` operation, then all additive differentials whose inputs and outputs belong to the same UNAF sets, are also possible. This is illustrated with the following example.

Example 3. Let $n = 4$ and $\Delta^{+}a = 5$, $\Delta^{+}b = 1$, $\Delta^{+}c = 6$. Because $\mathrm{adp}^{\oplus}(5, 1 \to 6) = 0.15625 > 0$, we can use Theorem 1 to show that $\mathrm{adp}^{\oplus}(\Delta^{+}a_i, \Delta^{+}b_j \to \Delta^{+}c_k) > 0$ for any $\Delta^{+}a_i \in \Delta^{U}a = \{3, -3, 5, -5\}$, $\Delta^{+}b_j \in \Delta^{U}b = \{1, -1\}$ and $\Delta^{+}c_k \in \Delta^{U}c = \{6, -6\}$.

In the next section we investigate the probability with which UNAF differences propagate through the `ARX` operation.

2.3 The UNAF Differential Probability of `ARX`

The UNAF differential probability of `ARX` represents the probability with which the sets of input additive differences $\Delta^{U}a$, $\Delta^{U}b$ and $\Delta^{U}d$ propagate to the set of output additive differences $\Delta^{U}e$. It is defined as:

Definition 5. ($\mathrm{udp}^{\mathtt{ARX}}$)

$$\begin{aligned}&\mathrm{udp}^{\mathtt{ARX}}(\Delta^{U}a, \Delta^{U}b, \Delta^{U}d \xrightarrow{t} \Delta^{U}e) = \\ &\frac{\#\{(a_1, b_1, d_1) : \Delta^{+}a \in \Delta^{U}a, \Delta^{+}b \in \Delta^{U}b, \Delta^{+}d \in \Delta^{U}d, \Delta^{+}e \in \Delta^{U}e\}}{\#\{(a_1, b_1, d_1) : \Delta^{+}a \in \Delta^{U}a, \Delta^{+}b \in \Delta^{U}b, \Delta^{+}d \in \Delta^{U}d\}} \ ,\end{aligned} \tag{6}$$

where

$$\Delta^{+}e = e_2 - e_1 = \mathtt{ARX}(a_1 + \Delta^{+}a, b_1 + \Delta^{+}b, d_1 + \Delta^{+}d, t) - \mathtt{ARX}(a_1, b_1, d_1, t),$$

and $\mathtt{ARX}(x, y, z, t) = ((x + y) \lll t) \oplus z$.

The probability $\mathrm{udp}^{\mathtt{ARX}}$ is computed using a method conceptually similar to the one proposed for the computation of $\mathrm{adp}^{\mathtt{ARX}}$ in [18]. The main difference is that in this case we are dealing with *sets* of input and output additive differences. Details on this computation are provided in Appendix B.

2.4 An Algorithm for Finding the Best Output Difference

To demonstrate how the UNAF framework can be used to construct high-probability differential characteristics, we have developed a general algorithm for the automatic search of differentials. It is capable of computing the highest probability output difference from a given operation. The proposed algorithm is applicable to any type of difference and any operation. The only condition is that the propagation of the difference through the operation can be represented as an S-function. The method to find the best output difference is based on the A* search algorithm [11].

Space constraints do not allow us to present the algorithm here in detail. However, a full description of the algorithm accompanied by pseudo-code can be found in Appendix C. Furthermore, a software toolkit that implements this algorithm is available.[4]

In the following sections we describe an application of the algorithm and of UNAF differences to the differential analysis of stream cipher Salsa20.

3 Applications

We describe several applications of the UNAF framework to the differential analysis of stream cipher Salsa20. UNAF differences can be used to obtain more accurate estimations of the probabilities of differentials through multiple rounds of ARX operations. We describe a key-recovery attack using UNAF differentials on a version of Salsa20, reduced to 5 rounds.

3.1 Description of Salsa20

Salsa20 is a stream cipher proposed by Bernstein in [3]. It is one of the finalists of the eSTREAM competition [8]. Salsa20 operates on 32-bit words. The inputs are a 256-bit key $(k_0, k_1, \ldots, k_7)$, a 64-bit nonce (v_0, v_1), a 64-bit counter (t_0, t_1) and four predefined 32-bit constants c_0, c_1, c_2, c_3. These inputs are mapped to a two-dimensional square matrix as follows:

$$\begin{bmatrix} c_0 & k_0 & k_1 & k_2 \\ k_3 & c_1 & v_0 & v_1 \\ t_0 & t_1 & c_2 & k_4 \\ k_5 & k_6 & k_7 & c_3 \end{bmatrix} \to \begin{bmatrix} w_0^0 & w_1^0 & w_2^0 & w_3^0 \\ w_4^0 & w_5^0 & w_6^0 & w_7^0 \\ w_8^0 & w_9^0 & w_{10}^0 & w_{11}^0 \\ w_{12}^0 & w_{13}^0 & w_{14}^0 & w_{15}^0 \end{bmatrix} . \tag{7}$$

The basic operation of Salsa20 is the *quarterround*. One *quarterround* transforms four of the input words to round $r+1$: $w_0^r, w_1^r, w_2^r, w_3^r$ into four output words: $w_0^{r+1}, w_1^{r+1}, w_2^{r+1}, w_3^{r+1}$ by the means of four consecutive ARX operations:

$$w_1^{r+1} = w_1^r \oplus ((w_0^r + w_3^r) \lll 7) = \mathtt{ARX}(w_0^r, w_3^r, w_1^r, 7) \ , \tag{8}$$
$$w_2^{r+1} = w_2^r \oplus ((w_1^{r+1} + w_0^r) \lll 9) = \mathtt{ARX}(w_1^{r+1}, w_0^r, w_2^r, 9) \ , \tag{9}$$
$$w_3^{r+1} = w_3^r \oplus ((w_2^{r+1} + w_1^{r+1}) \lll 13) = \mathtt{ARX}(w_2^{r+1}, w_1^{r+1}, w_3^r, 13) \ , \tag{10}$$
$$w_0^{r+1} = w_0^r \oplus ((w_3^{r+1} + w_2^{r+1}) \lll 18) = \mathtt{ARX}(w_3^{r+1}, w_2^{r+1}, w_0^r, 18) \ . \tag{11}$$

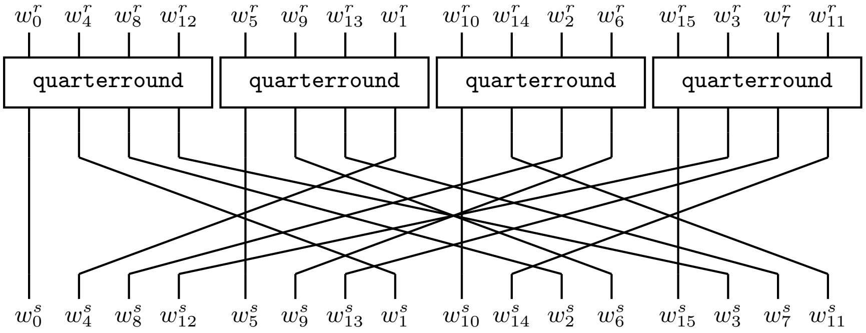

w_0^r w_4^r w_8^r w_{12}^r w_5^r w_9^r w_{13}^r w_1^r w_{10}^r w_{14}^r w_2^r w_6^r w_{15}^r w_3^r w_7^r w_{11}^r
quarterround
quarterround
quarterround
quarterround
w_0^s w_4^s w_8^s w_{12}^s w_5^s w_9^s w_{13}^s w_1^s w_{10}^s w_{14}^s w_2^s w_6^s w_{15}^s w_3^s w_7^s w_{11}^s

Fig. 1. Round $s = r + 1$ of Salsa20.

One *round* of Salsa20 consists of four parallel applications of the *quarterround* transformation. Each transformation is applied to the elements (in permuted order) of one of the four columns of the input state matrix, followed by a permutation of the words, as shown on Fig. 1.

Salsa20 has a total of 20 rounds, although versions with eight and twelve rounds have been proposed, resp. Salsa20/8 and Salsa20/12. The output state after the last round is added to the initial input state by means of a feed-forward operation. This produces sixteen 32-bit words (512 bits) of key stream.

3.2 Estimating the Probability of Differentials Using UNAF Differences

We apply the algorithm of Sect. 2.4 to search for high probability differential characteristics in Salsa20. We use a greedy strategy in which at every ARX operation we select the output UNAF difference with the highest probability, before proceeding with the next ARX operation. In this way we find the following truncated differential for three rounds:

$$\varDelta_8^0 = \texttt{0x80000000} \rightarrow \varDelta_9^3 = \texttt{0x80000000} \ . \tag{12}$$

The expression (12) implies that all words of the input state have zero difference, except for the word at position 8, which has difference 0x80000000. A three round differential characteristic that satisfies (12) is shown on Fig. 2. The probability with which the differential (12) holds, obtained experimentally over 2^{20} chosen plaintexts, is $p_{\text{exper}} = 2^{-3.39}$.

We compute two theoretical estimations of p_{exper}. The first estimation is based on single additive differences and is denoted $\hat{p}_{\text{add}}$. It is computed as a multiplication of $\text{adp}^{\texttt{ARX}}$ probabilities:

$$\hat{p}_{\text{add}} = \prod \text{adp}^{\texttt{ARX}} = 2^{-10} \ . \tag{13}$$

The second estimation of p_{exper} is based on UNAF differences and is denoted $\hat{p}_{\text{unaf}}$. It is computed as a multiplication of $\text{udp}^{\texttt{ARX}}$ probabilities:

$$\hat{p}_{\text{unaf}} = \prod \text{udp}^{\texttt{ARX}} = 2^{-4} \ . \tag{14}$$

The computations (13) and (14) are shown in Table 2 and Table 3 respectively.

Clearly $\hat{p}_{\text{unaf}}$ is a better estimation of p_{exper} than $\hat{p}_{\text{add}}$. The reason is that multiple differential characteristics connect the input and output differences of the differential (12). The estimation $\hat{p}_{\text{add}}$ is based upon a single one among all possible characteristics, while the estimation $\hat{p}_{\text{unaf}}$ takes into account several characteristics at once. This effect is illustrated in Fig. 3. Note that

[4] http://www.ecrypt.eu.org/tools/s-function-toolkit

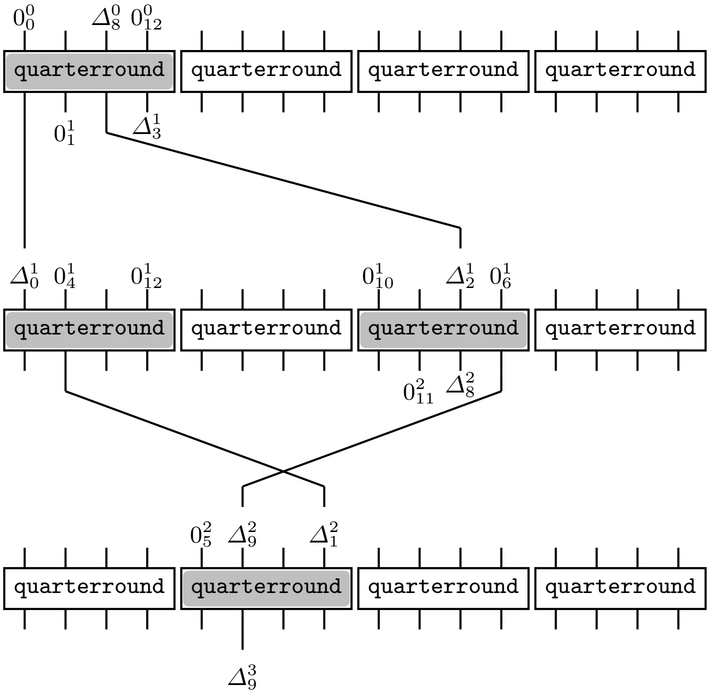


Fig. 2. Three round differential characteristic satisfying the differential $\Delta_8^0 \rightarrow \Delta_9^3$.

the input $\{\Delta^U\}_8^0$ and output $\{\Delta^U\}_9^3$ UNAF sets contain a single element – the additive difference 80000000. Because of that $\{\Delta^U\}_8^0 = \Delta_8^0$ and $\{\Delta^U\}_9^3 = \Delta_9^3$ and therefore the estimations (13) and (14) can be compared to each other.

In the case where the output UNAF set contains more than one element (i.e. $\{\Delta^U\}_9^3 \neq \Delta_9^3$), we propose to divide the resulting probability by the size of the output UNAF set $\#\Delta^U$:

$$\hat{p}_{\text{unaf}} = \frac{\prod \text{udp}^{\texttt{ARX}}}{\#\Delta^U} \ . \tag{15}$$

The estimation (15) is based on the assumption that all additive differences from the output UNAF set Δ^U hold with approximately the same (or very close) probabilities. For the case of Salsa20, our experiments confirm this assumption.

We use (15) to estimate the probabilities with which several differences from the output state after Salsa20/3 hold, given input UNAF difference $\{\Delta^U\}_8^0 = $ 0x80000000. The results are shown in Table 4 and in Fig. 4.

The results presented in Table 4 and Fig. 4 show that although the probability estimations $\hat{p}_{\text{unaf}}/\#\Delta^U$ computed using UNAF differences with (15) deviate from the values obtained experimentally p_{exper}, they are still more accurate than the estimations $\hat{p}_{\text{add}}$ based on single additive differences and computed with (13).

3.3 Key-recovery Attack on Salsa20/5

In this section, we apply UNAF differences to mount a key-recovery attack on a version of stream cipher Salsa20 reduced to 5 rounds, denoted as Salsa20/5. Although its complexity is lower than exhaustive key search, the attack does not improve the best known attack on the cipher. Therefore it is described only as a demonstration of a practical application of UNAF differences.

Table 2. The estimated probability $\hat{p}_{\text{add}}$ (13) of the differential (12); $\text{adp}^{\texttt{ARX}}$ refers to $\text{adp}^{\texttt{ARX}}((\varDelta^+a+\varDelta^+b), \varDelta^+d \xrightarrow{t} \varDelta^+e)$.

$\varDelta$	$\varDelta^+a$	$\varDelta^+b$	$\varDelta^+d$	t	$\varDelta^+e = \varDelta$	$\text{adp}^{\texttt{ARX}}$
$\varDelta_2^1$	0	0	80000000	9	80000000	1
$\varDelta_3^1$	80000000	0	0	13	fffff000	2^{-1}
$\varDelta_0^1$	fffff000	80000000	0	18	40020000	$2^{-2.41}$
$\varDelta_1^2$	40020000	0	0	7	01000020	$2^{-2.99}$
$\varDelta_8^2$	0	0	80000000	9	80000000	1
$\varDelta_9^2$	80000000	0	0	13	fffff000	2^{-1}
$\varDelta_9^3$	0	01000020	fffff000	7	80000000	$2^{-2.58}$

$\hat{p}_{\text{add}} = 2^{-10}$

Table 3. The estimated probability $\hat{p}_{\text{unaf}}$ (14) of the differential (12); $\text{udp}^{\texttt{ARX}}$ refers to $\text{udp}^{\texttt{ARX}}(\varDelta^U a, \varDelta^U b, \varDelta^U d \xrightarrow{t} \varDelta^U e)$.

$\varDelta^U$	$\varDelta^U a$	$\varDelta^U b$	$\varDelta^U d$	t	$\varDelta^U e = \varDelta^U$	$\text{udp}^{\texttt{ARX}}$
$\{\varDelta^U\}_2^1$	0	0	80000000	9	80000000	1
$\{\varDelta^U\}_3^1$	80000000	0	0	13	00001000	1
$\{\varDelta^U\}_0^1$	00001000	80000000	0	18	40020000	$2^{-0.41}$
$\{\varDelta^U\}_1^2$	40020000	0	0	7	01000020	$2^{-0.99}$
$\{\varDelta^U\}_8^2$	0	0	80000000	9	80000000	1
$\{\varDelta^U\}_9^2$	80000000	0	0	13	00001000	1
$\{\varDelta^U\}_9^3$	0	01000020	00001000	7	80000000	$2^{-2.58}$

$\hat{p}_{\text{unaf}} = 2^{-4}$

Using the best-first search algorithm from Sect. 2.4 we find the following UNAF differential for 3 rounds of Salsa20:

$$\{\varDelta^U\}_8^0 = \texttt{0x80000000} \rightarrow \{\varDelta^U\}_{11}^3 = \texttt{0x01000024}\ . \tag{16}$$

The input UNAF set $\{\varDelta^U\}_8^0 = \texttt{0x80000000}$ consists of one element: the additive difference 0x80000000. The output UNAF set $\{\varDelta^U\}_{11}^3 = \texttt{0x01000024}$ contains the following 2^3 additive differences: 0x01000024, 0x0100001c, 0x00ffffe4, 0x00ffffdc, 0xff000024, 0xff00001c, 0xfeffffe4, 0xfeffffdc. The probability that an additive difference $\varDelta_{11}^3$ falls into the set $\{\varDelta^U\}_{11}^3$ was determined experimentally to be $p_{\text{exper}} = 2^{-3.38}$.

In our attack, we first invert the feed-forward operation to compute the differences $\varDelta_5^5$, $\varDelta_6^5, \ldots, \varDelta_{10}^5$ of the state after round 5. Next, we guess 5 of the 8 words of the secret key, in order to compute the differences $\varDelta_1^5$,$\varDelta_2^5$,$\varDelta_3^5$,$\varDelta_4^5$,$\varDelta_{11}^5$. Therefore, we do not only know the differences $\varDelta_1^5$,$\varDelta_2^5$,$\ldots$,$\varDelta_{11}^5$, but also the corresponding values of the word pairs. This allows us to compute the differences $\varDelta_{12}^4$,$\varDelta_{13}^4$,$\varDelta_{14}^4$ from the state after round 4. Using the latter, we can finally compute the UNAF difference $\{\varDelta^U\}_{11}^3$. If it is equal to 0x01000024, then our guess of the key words was correct with some probability. This process is illustrated in Appendix D.

Since the probability of the differential (16) is $2^{-3.38} \geq 2^{-4}$, from $M = 2^6$ chosen plaintext pairs we expect that $2^{-4} \cdot 2^6 = 2^2 = 4$ pairs will follow the differential (i.e. will satisfy the output difference $\{\varDelta^U\}_{11}^3$).

We assume that a pair encrypted under a wrong key results in a uniformly random difference. The probability that this difference falls into the set $\{\varDelta^U\}_{11}^3$ is $P_{\text{rand}} = 2^3/2^{32} = 2^{-29}$. Therefore the probability that at least 4 plaintext pairs turn out to be all false positives (i.e. they satisfy the differential, but are encrypted under a wrong key) can be calculated using the binomial

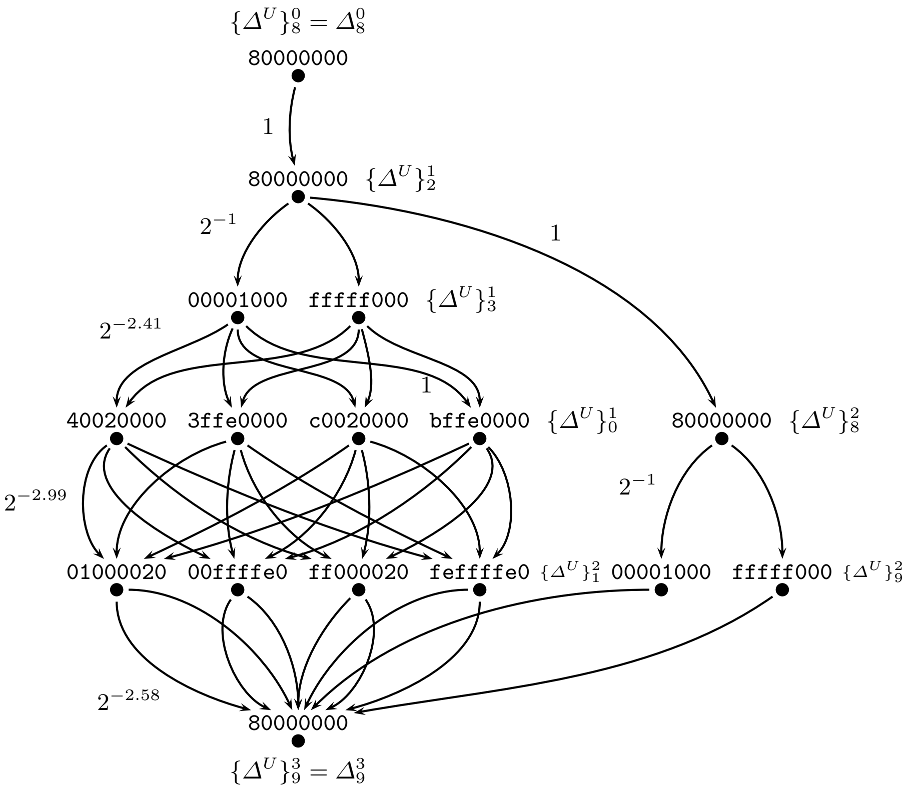

$\{\varDelta^U\}^0_8 = \varDelta^0_8$
80000000
1
80000000 $\{\varDelta^U\}^1_2$
2^{-1}
1
00001000
fffff000 $\{\varDelta^U\}^1_3$
$2^{-2.41}$
1
40020000
3ffe0000
c0020000
bffe0000 $\{\varDelta^U\}^1_0$
80000000 $\{\varDelta^U\}^2_8$
$2^{-2.99}$
2^{-1}
01000020
00ffffe0
ff000020
feffffe0 $\{\varDelta^U\}^2_1$
00001000
fffff000 $\{\varDelta^U\}^2_9$
$2^{-2.58}$
80000000
$\{\varDelta^U\}^3_9 = \varDelta^3_9$

Fig. 3. A single UNAF characteristic, satisfying the differential $\varDelta^0_8 \to \varDelta^3_9$. It is composed of multiple additive characteristics.

distribution:

$$\sum_{i=4}^{64} \binom{64}{i} (2^{-29})^i (1 - 2^{-29})^{64-i} \approx 2^{-96.72} \quad . \tag{17}$$

As explained, because we guess 160 bits (5 words) of the secret key, in the attack we have to make 2^{160} guesses. For each guess, we encrypt 2^6 chosen plaintext pairs and we partially decrypt the resulting ciphertext pairs for 2 rounds in order to compute the output difference. From 2^{160} guesses, the expected number of wrong keys that result in at least 4 pairs with the right difference is $2^{-96.72} \cdot 2^{160} \approx 2^{63}$. For each of those keys, we guess the remaining 96 bits (3 words) i.e. we make 2^{96} guesses per candidate key. For each guess we encrypt one plaintext pair (i.e. two encryptions are performed) under the full key and check if the encryption matches the corresponding ciphertext pair. This results in $2 \cdot 2^{63} \cdot 2^{96} = 2^{160}$ additional operations. Thus we estimate the total number of encryptions of our attack to be:

$$2 \cdot 2^6 \cdot 2^{160} + 2 \cdot 2^{63} \cdot 2^{96} = 2^{167} + 2^{160} \approx 2^{167} \quad . \tag{18}$$

Therefore the presented attack on Salsa20/5 has data complexity 2^7 chosen plaintexts and time complexity 2^{167} encryptions. As shown in Table 5, it is comparable to the attack proposed by Crowley [5].

Table 4. Estimating the probabilities of differentials for three rounds of Salsa20 using UNAF differences.

i	Δ_i^3	$\{\Delta^U\}_i^3$	$\hat{p}_{\text{add}}$	$\hat{p}_{\text{unaf}}/\#\Delta^U$	p_{exper}
9	`80000000`	`80000000`	$2^{-10.00}$	$2^{-4.00}$	$2^{-3.38}$
13	`ffe00100`	`00200100`	$2^{-15.75}$	$2^{-7.75}$	$2^{-4.93}$
14	`ff00001c`	`01000024`	$2^{-16.29}$	$2^{-8.31}$	$2^{-6.35}$
1	`00e00fe4`	`01201024`	$2^{-23.01}$	$2^{-13.04}$	$2^{-10.18}$
2	`00000800`	`00000800`	$2^{-35.59}$	$2^{-16.62}$	$2^{-11.08}$
3	`fff000a0`	`001000a0`	$2^{-41.48}$	$2^{-20.04}$	$2^{-14.68}$
6	`01038020`	`01048020`	$2^{-41.76}$	$2^{-21.91}$	$2^{-15.68}$
7	`ffefc000`	`00104000`	$2^{-44.65}$	$2^{-22.15}$	$2^{-17.42}$

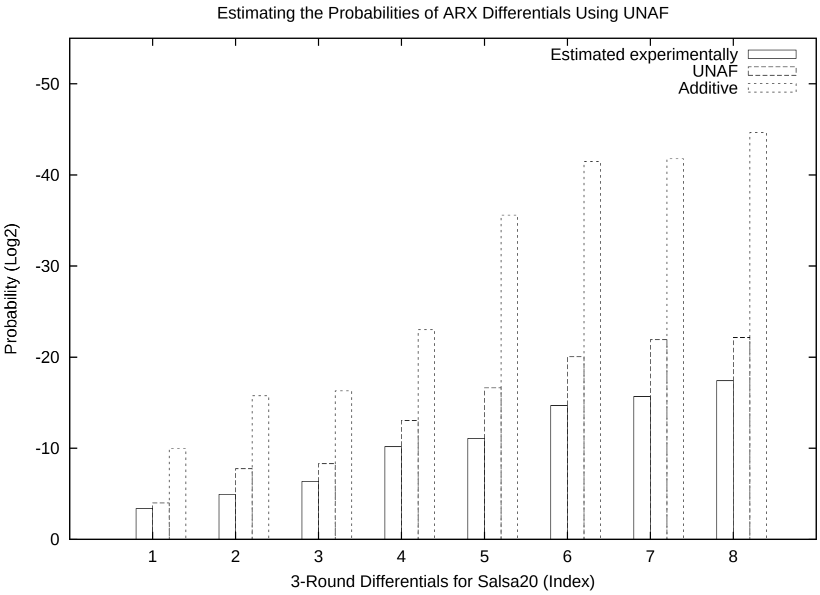


Fig. 4. Three estimates of the probabilities of eight differentials for three rounds of Salsa20, based on the data from Table 4: (1) estimation obtained experimentally, (2) based on UNAF differences and (3) based on single additive differences.

4 Conclusion

In this paper, we introduced UNAF differences. These are sets of specially chosen additive differences used to estimate the probabilities of differentials through sequences of ARX operations more accurately.

We presented the main UNAF theorem, which shows how a UNAF difference groups several possible additive differences together. Further, we investigated the propagation of UNAF differences through the ARX operation. We defined the UNAF differential probability of ARX and noted that it can be computed efficiently using the S-functions framework proposed by Mouha et al.

UNAF differences were applied to the cryptanalysis of the stream cipher Salsa20. We found that for three rounds of Salsa20, the probability of the best differential based on additive differences is estimated as 2^{-10}. Evaluating the same probability using UNAF differences leads to the value 2^{-4}. The latter is closer to the the probability of the differential $2^{-3.39}$ that was determined experimentally.

A general algorithm for the automatic search for differentials was briefly discussed. It was used to find high-probability UNAF differentials for three rounds of Salsa20. One of them was used to mount a key-recovery attack on Salsa20 reduced to five rounds. The attack has a time

Table 5. Overview of key-recovery attacks on Salsa20.

Rounds	Reference	Time	Data	Type of Differences
Salsa20/5	**Our result**	$\mathbf{2^{167}}$	$\mathbf{2^{7}}$	**Additive**
Salsa20/5	Crowley [5]	2^{165}	2^{6}	XOR
Salsa20/6	Fischer et al. [10]	2^{177}	2^{16}	XOR
Salsa20/7	Aumasson et al. [1]	2^{151}	2^{26}	XOR
Salsa20/8	Aumasson et al. [1]	2^{251}	2^{31}	XOR

complexity of 2^7 and a data complexity of 2^{167}. It therefore does not improve the best-known attack on the cipher. Nevertheless, to the best of our knowledge, this is the first cryptanalysis result on Salsa20 that is based on additive differences. Furthermore, we expect that the attack can be extended to more rounds. One possibility in this direction is to group two or more ARX operations and consider them as a single operation. Another is to improve the method for finding differential characteristics for multiple rounds.

The results in this paper were obtained for the Salsa20 stream cipher. We see the application of UNAF differences to other ARX-based ciphers as another interesting topic for future research.

Acknowledgments. The authors would like to thank Florian Mendel for his detailed comments and for his suggestions on the overall structure of the paper. His feedback was critical for improving the quality of the final text. We thank prof. Vincent Rijmen for reviewing the draft version of the paper. We also thank our colleagues at COSIC for the useful discussions, as well as the anonymous reviewers for their detailed comments.

References

1. J.-P. Aumasson, S. Fischer, S. Khazaei, W. Meier, and C. Rechberger. New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba. In K. Nyberg, editor, *FSE*, volume 5086 of *Lecture Notes in Computer Science*, pages 470–488. Springer, 2008.
2. J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan. SHA-3 proposal BLAKE. Submission to the NIST SHA-3 Competition (Round 2), 2008.
3. D. J. Bernstein. The Salsa20 Family of Stream Ciphers. In M. J. B. Robshaw and O. Billet, editors, *The eSTREAM Finalists*, volume 4986 of *LNCS*, pages 84–97. Springer, 2008.
4. E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *J. Cryptology*, 4(1):3–72, 1991.
5. P. Crowley. Truncated differential cryptanalysis of five rounds of Salsa20. SASC 2006 Workshop: Stream Ciphers Revisted. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/073, 2005. http://www.ecrypt.eu.org/stream.
6. J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.
7. N. M. Ebeid and M. A. Hasan. On binary signed digit representations of integers. *Des. Codes Cryptography*, 42(1):43–65, 2007.
8. eSTREAM. ECRYPT stream cipher project. http://www.ecrypt.eu.org/stream.
9. N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker. The Skein Hash Function Family. Submission to the NIST SHA-3 Competition (Round 2), 2009.
10. S. Fischer, W. Meier, C. Berbain, J.-F. Biasse, and M. J. B. Robshaw. Non-randomness in eSTREAM Candidates Salsa20 and TSC-4. In R. Barua and T. Lange, editors, *INDOCRYPT*, volume 4329 of *Lecture Notes in Computer Science*, pages 2–16. Springer, 2006.
11. P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions On Systems Science And Cybernetics*, 4(2):100–107, July 1968.
12. H. Lipmaa and S. Moriai. Efficient Algorithms for Computing Differential Properties of Addition. In M. Matsui, editor, *FSE*, volume 2355 of *LNCS*, pages 336–350. Springer, 2001.
13. H. Lipmaa, J. Wallén, and P. Dumas. On the Additive Differential Probability of Exclusive-Or. In B. K. Roy and W. Meier, editors, *FSE*, volume 3017 of *LNCS*, pages 317–331. Springer, 2004.
14. M. Matsui and A. Yamagishi. A New Method for Known Plaintext Attack of FEAL Cipher. In *EUROCRYPT*, pages 81–91, 1992.

15. N. Mouha, V. Velichkov, C. De Cannière, and B. Preneel. The Differential Analysis of S-Functions. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 36–56. Springer, 2010.
16. G. W. Reitwiesner. Binary arithmetic. *Advances in Computers*, 1:231–308, 1960.
17. A. Shimizu and S. Miyaguchi. Fast Data Encipherment Algorithm FEAL. In *EUROCRYPT*, pages 267–278, 1987.
18. V. Velichkov, N. Mouha, C. De Cannière, and B. Preneel. The Additive Differential Probability of ARX. In A. Joux, editor, *FSE*, volume 6733 of *LNCS*, pages 342–358. Springer, 2011.

A Proof of Theorem 1

The following Lemma provides the condition under which the probability $\mathrm{adp}^{\oplus}$ is non-zero.

Lemma 1 (Theorem 2 of [13]). *All differences $\Delta^+ a$, $\Delta^+ b$ and $\Delta^+ c$ for which* $\mathrm{adp}^{\oplus}(\Delta^+ a, \Delta^+ b \rightarrow \Delta^+ c) > 0$*, are $\Delta^+ a = \Delta^+ b = \Delta^+ c = 0$, and*

$$\Delta^+ a = \Delta^+ a[n-1 \dots q+1] \parallel \Delta^+ a[q] \parallel 0^* \ , \tag{19}$$

$$\Delta^+ b = \Delta^+ b[n-1 \dots q+1] \parallel \Delta^+ b[q] \parallel 0^* \ , \tag{20}$$

$$\Delta^+ c = \Delta^+ c[n-1 \dots q+1] \parallel \Delta^+ c[q] \parallel 0^* \ , \tag{21}$$

where $\neg(\Delta^+ a[q] = \Delta^+ b[q] = \Delta^+ c[q] = 0)$ and $\Delta^+ a[q] \oplus \Delta^+ b[q] = \Delta^+ c[q]$. Each of the sub-word differences $\Delta^+ a[n-1 \dots q+1]$, $\Delta^+ b[n-1 \dots q+1]$ and $\Delta^+ c[n-1 \dots q+1]$ can take any arbitrary value. The symbol $*$ represents the Kleene star.

We proceed next with the proof of Theorem 1.

Proof. From Reitwiesner's algorithm for the construction of the NAF [16], it follows that if the first non-zero bit (starting from the LSB) of $\Delta^+ a_i$ is at position q, then the first non-zero bit of its NAF representation $\Delta^N a_i$ is also at position q. Since all $\Delta^+ a_i$ in (5) belong to the same UNAF set $\Delta^U a$, the first non-zero bit for all of them is in the same position q. The same observation holds for $\Delta^+ b_j$ and $\Delta^+ c_k$. From $\mathrm{adp}^{\oplus}(\Delta^+ a, \Delta^+ b \rightarrow \Delta^+ c) > 0$ and Lemma 1, it follows that $\Delta^+ a[q] \oplus \Delta^+ b[q] = \Delta^+ c[q]$. Therefore $\Delta^+ a_i[q] \oplus \Delta^+ b_j[q] = \Delta^+ c_k[q], \forall i, j, k$. Again by Lemma 1, it follows that if $\Delta^+ a$ is replaced by any $\Delta^+ a_i$ belonging to the same UNAF set $\Delta^U a$, the resulting probability $\mathrm{adp}^{\oplus}$ is still non-zero. The same observation can be made for $\Delta^+ b$ and $\Delta^+ c$, which completes the proof. □

B Computation of $\mathrm{udp}^{\mathtt{ARX}}$

The probability $\mathrm{udp}^{\mathtt{ARX}}$ can be efficiently computed using the S-function framework [15, 18]. We briefly describe this computation below. It is also a part of a toolkit that will be made publicly available.

The propagation of input UNAF differences $\Delta^U a$, $\Delta^U b$ and $\Delta^U d$ to output UNAF difference $\Delta^U e$ is represented as an S-function. The latter is used to compute 16 adjacency matrices. Each of them corresponds to a given value of the i-th bit of each of the four UNAF differences and connects a set of possible input states to a set of possible output states.

The differential $(\Delta^U a[i], \Delta^U b[i], \Delta^U d[i+t] \xrightarrow{t} \Delta^U e[i+t])$ at bit position i is written as the bit string $w[i] \leftarrow (\Delta^U a[i] \parallel \Delta^U b[i] \parallel \Delta^U d[i+t] \parallel \Delta^U e[i+t])$. At each bit position $0 \leq i < n$, the index $w[i] \in \{0, \dots, 15\}$ selects one of the 16 adjacency matrices $A_{w[i]}$. The probability $\mathrm{udp}^{\mathtt{ARX}}$ is computed as follows:

$$\mathrm{udp}^{\mathtt{ARX}}(\Delta^U a, \Delta^U b, \Delta^U d \xrightarrow{t} \Delta^U e) = \sum_{j=0}^{14} L_j \left(\prod_{i=n-t}^{n-1} A_{w[i]} \right) R \left(\prod_{i=0}^{n-t-1} A_{w[i]} \right) C_j \ . \tag{22}$$

In (22), the summation is performed over each of the 14 possible initial states. The reason for having multiple initial states is the bit rotation by t positions, as explained in [18]. The multiplication by the projection matrix R at bit position t is necessary because of the rotation operation. The column vectors $C_j,\ 0 \leq j < 15$ represent the 15 possible initial states. The row vectors $L_j,\ 0 < j < 15$ represent their corresponding final states. For further details, we refer to [18].

Note that the matrices $A_{w[i]}$ are of dimension 540×540, but these can be minimized to 60×60 by combining equivalent states using the algorithm of [15, §3.5] .

C An Algorithm for Finding the Best Output Difference

Let $\Box$ be an operation that takes a finite number of n-bit input words $a_1, b_1, d_1, \ldots$ and computes an n-bit output word $c_1 = \Box(a_1, b_1, d_1, \ldots)$. Let $\bullet$ be a type of difference. Let α,β,ζ,... and γ be differences of type $\bullet$ such that $a_1 \bullet a_2 = \alpha$, $b_1 \bullet b_2 = \beta$, $d_1 \bullet d_2 = \zeta$, ... and $c_1 \bullet c_2 = \gamma$ for some a_2,b_2,d_2,... and some c_2. The differential probability with which input differences α, β, ζ, ... propagate to output difference γ with respect to the operation $\Box$ is denoted as $\bullet\mathrm{dp}^{\Box}(\alpha, \beta, \zeta, \ldots \rightarrow \gamma)$. Finally, let the difference $\bullet$ be such that it is possible to express its propagation through the operation $\Box$ as an S-function consisting of N states. Therefore, there exist adjacency matrices $A_{w[i]}$ such that the probability $\bullet\mathrm{dp}^{\Box}$ can be efficiently computed as $LA_{w[n-1]} \ldots A_{w[1]}A_{w[0]}C$, where $L = [1\,1 \cdots 1]$ is a $1 \times N$ matrix and $C = [1\,0 \cdots 0]^T$ is an $N \times 1$ matrix (as in [15]). The problem is to find an output difference γ such that its probability p_γ over all possible output differences is maximal:

$$p_\gamma = \bullet\mathrm{dp}^{\Box}(\alpha, \beta, \zeta, \ldots \rightarrow \gamma) = \max_j \ \bullet\,\mathrm{dp}^{\Box}(\alpha, \beta, \zeta, \ldots \rightarrow \gamma_j)\ . \tag{23}$$

We represent (23) as a problem of finding the shortest path in an *node-weighted binary tree*. We define the binary tree $T = (N, E)$, where N is the set of nodes and E is the set of edges. The height of T is $n + 1$ with a dummy start node positioned at level -1 and the leaves positioned at level $n - 1$. Each node at level $i : 0 \leq i < n$ contains a value of $\gamma[i]$, where $i = 0$ is the LSB and $i = n - 1$ is the MSB. Every node on level i has two children at level $i + 1$. Since the input differences $\alpha, \beta, \zeta, \ldots$ are fixed, at every bit position i we can choose between two matrices $A_{w[i]}$, corresponding to the two possibilities for the output difference $\gamma[i]$.

To find the output difference with the highest probability, we use the A* search algorithm [11]. In this algorithm, an evaluation function f can be computed for every node in the search tree. The f-function represents the weight of a node, and is based on the cost of the path from the start node, and a heuristic that estimates the distance to the goal node. The algorithm always expands the node with the highest f-value (corresponding to the highest probability). The A* search algorithm guarantees that the optimal solution will be found, provided that the evaluation function f never underestimates the probability of the best output difference. After introducing some definitions, we will define an evaluation function f and prove in Theorem 2 that this f satisfies the required condition.

Let vector $X_i = [x_{i,0}\, x_{i,1} \cdots x_{i,N-1}]$ be a transition probability vector, i.e. $x_{i,r} \geq 0$ for $0 \leq r < N$ and $\sum_{r=0}^{N-1} x_{i,r} \leq 1$. We define H_r as a column vector of length N, of which the r-th element (counting from 0) is 1 and all other elements are 0. The cost of a node at level i is then denoted by $\|X_i\|$ (the 1-norm of X_i) and is calculated as $\|A_{w[i]}A_{w[i-1]} \cdots A_{w[0]}C\|$. Let us define a sequence of row vectors $\hat{G}_{i,r}$, $0 \leq r < N$ and $0 \leq i < n$. Each $\hat{G}_{i,r}$ is a product of matrices $LA_{w[n-1]}A_{w[n-2]} \ldots A_{w[i+1]}$, where each of the A-matrices are chosen such that $\hat{G}_{i,r}H_r$ is maximized. The choice of the A-matrices may differ for different values of r. We define row vector G_i as the product of matrices $LA_{w[n-1]}A_{w[n-2]} \ldots A_{w[i+1]}$, where the A-matrices are chosen such that $G_i X_i$ is maximized. For a node at level i with cost $\|X_i\|$, the evaluation function f is defined as $\sum_{r=0}^{N-1} \hat{G}_{i,r} H_r x_{i,r}$.

Theorem 2. *The evaluation function $f = \sum_{r=0}^{N-1} \hat{G}_{i,r} H_r x_{i,r}$ never underestimates the probability of the best output difference.*

Proof. The following inequality holds: $\hat{G}_{i,r} H_r \geq G_i H_r$ for $0 \leq r < N$. The latter can be proven by contradiction: if $\hat{G}_{i,r} H_r < G_i H_r$ for some r, then $\hat{G}_{i,r}$ is not the product of A-matrices that maximizes $\hat{G}_{i,r} H_r$, which contradicts its definition. Because probabilities are non-negative, we can multiply both sides of the inequality by the state probability $x_{i,r}$, to obtain $\hat{G}_{i,r} H_r x_{i,r} \geq G_i H_r x_{i,r}$, $0 \leq r < N$. By summing the left and the right sides of the N inequalities, we obtain $\sum_{r=0}^{N-1} \hat{G}_{i,r} H_r x_{i,r} \geq \sum_{r=0}^{N-1} G_i H_r x_{i,r} = G_i X_i$. By definition, $G_i X_i$ is the best choice of A-matrices, starting from transition probability X_i. This proves that the left-hand side of the inequality never underestimates the probability, which proves the theorem. □

Before we can apply the A* algorithm to compute the best output difference, we must determine the values of $\hat{G}_{i,r} H_r$ for $0 \leq i < n$ and $0 \leq r < N$. This is done by again running the A* algorithm for the most significant bit, then for the two most significant bits, and so on until we process the entire word. For the MSB, we define $\hat{G}_{n-1,r} = L$ for $0 \leq r < N$. For the two MSBs, we run the A* algorithm for every $0 \leq r < N$, setting the transition probability vector X_{n-2} to H_r. This allows us to compute $\hat{G}_{n-2,r} H_r$. This process is continued until $\hat{G}_{0,r} H_r$ for $0 \leq r < N$ is calculated. Having calculated all values of $\hat{G}_{i,r} H_r$, we then use the A* algorithm to search for the best output difference by setting the state transition probability vector $X_{-1} = C$. Pseudo-code of the entire A* search algorithm is provided in Algorithm 1.

D Attack on Salsa20/5 using UNAF Differences

Fig. 5 illustrates the attack presented in Sect. 3.3. Gray boxes denote guessed words and white boxes denote words that are either known or can be computed.

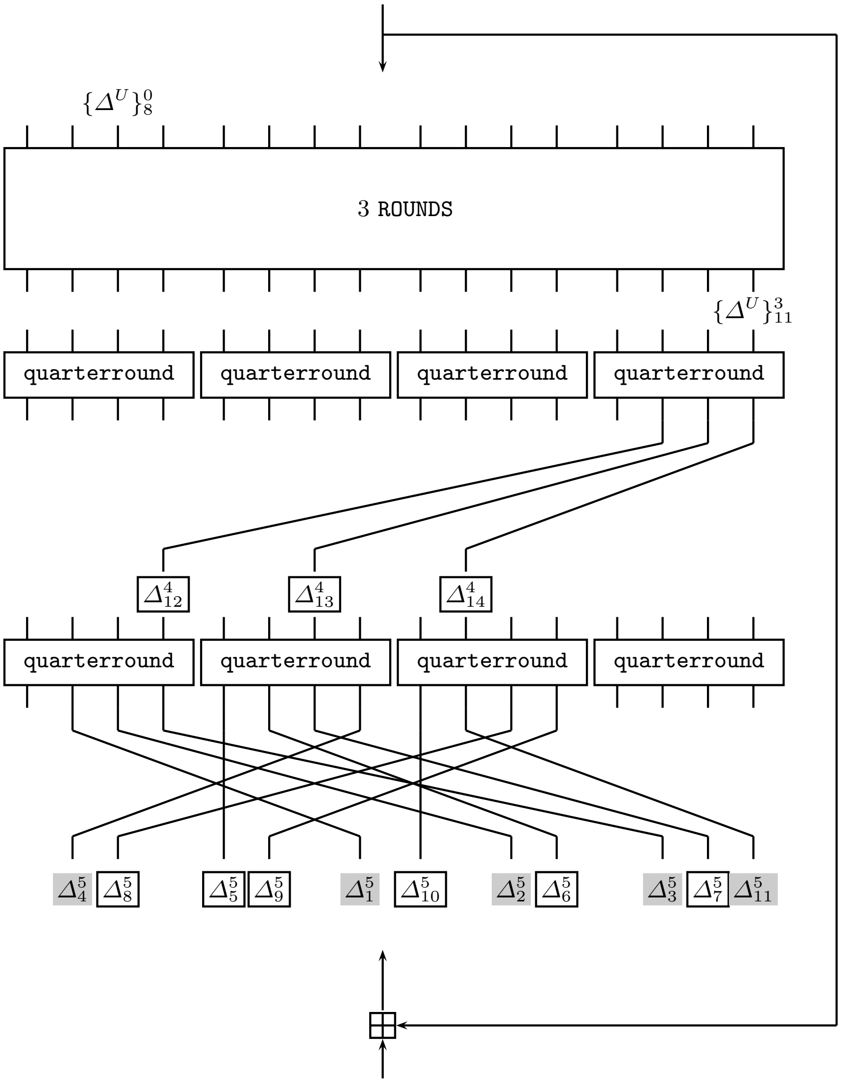


Fig. 5. Key-recovery attack on Salsa20/5 using the 3-round UNAF differential $\{\Delta^U\}_8^0 \rightarrow \{\Delta^U\}_{11}^3$. Gray boxes denote guessed words; white boxes denote words that are either known or can be computed.

Algorithm 1 Find the Best Output Diff. of Type $\bullet$ w.r.t. Operation $\Box$.

Input: Matrices $A_{w[i]}$ for $\bullet\mathrm{dp}^{\Box}$; input diffs. α, β, ζ, $\ldots$,; num. states N.
Output: Output difference γ and probability p_γ such that

$$p_\gamma = \bullet\mathrm{dp}^{\Box}(\alpha, \beta, \zeta, \ldots \to \gamma) = \max_j \ \bullet\mathrm{dp}^{\Box}(\alpha, \beta, \zeta, \ldots \to \gamma_j) \ .$$

```
Define struct **node** = $\{\mathrm{index},\ \gamma,\ f_{\mathrm{index}-1},\ \hat{H}_{\mathrm{index}-1}\}$
Init priority queue of nodes ordered by $f$: $Q = \varnothing$
Init output difference: $\gamma \leftarrow \varnothing$
for $i = n-1$ downto $0$ do
    if $i = n-1$ then
        $\hat{G}_i \leftarrow L = [1\ 1 \cdots 1]$
    else
        $\hat{G}_i \leftarrow [\hat{G}_{i,0}\ \hat{G}_{i,1} \ldots \hat{G}_{i,N-1}]$
    end if
    if $i = 0$ then
        $N = 1$
    end if
    for $r = 0$ to $N-1$ do
        Reset priority queue: $Q = \varnothing$
        Init the total probability of node $v_{i-1}$: $f_{i-1} \leftarrow 1$
        Init the transition probability vector $v_i$: $\hat{H}_{i-1} \leftarrow \hat{H}_{i-1,r}$
        Init node $v_i \leftarrow \{i,\ \gamma,\ f_{i-1},\ \hat{H}_{i-1}\}$
        Add new node to the queue: $Q$.push($v_i$)
        $v_{\mathrm{best}} \leftarrow Q$.top();  $\{j,\ \gamma,\ f_{j-1},\ \hat{H}_{j-1}\} \leftarrow v_{\mathrm{best}}$
        while $j \neq n$ do
            Remove $v_{\mathrm{best}}$ from the queue: $Q$.pop()
            for $q = 0$ to $1$ do
                Set the $j$-th bit of $\gamma$: $\gamma[j] \leftarrow q$
                Estimate the total probability: $f_j \leftarrow \hat{G}_j A^q_{w[j]} \hat{H}_{j-1}$
                Compute the transition probability vector: $\hat{H}_j \leftarrow A^q_{w[j]}\ \hat{H}_{j-1}$
                Init child of $v_{\mathrm{best}}$: node $v^q_{j+1} \leftarrow \{j+1,\ \gamma,\ f_j,\ \hat{H}_j\}$
                Add the child to the queue: $Q$.push($v^q_{j+1}$)
            end for
            Extract the node with the lowest total cost: $v_{\mathrm{best}} \leftarrow Q$.top()
            $\{j,\ \gamma,\ f_{j-1},\ \hat{H}_{j-1}\} \leftarrow v_{\mathrm{best}}$
        end while
        $v_{\mathrm{best}} \leftarrow Q$.top();  $f_{\mathrm{best}} \leftarrow$ get_cost($v_{\mathrm{best}}$)
        Set the $r$-th element of $\hat{G}_i$: $\hat{G}_{i,r} \leftarrow f_{\mathrm{best}}$
    end for
end for
Extract the node with highest total probability: $v_{\mathrm{best}} \leftarrow Q$.top()
Get the output difference associated to $v_{\mathrm{best}}$: $\gamma, p_\gamma \leftarrow$ get_gamma($v_{\mathrm{best}}$)
return $\gamma, p_\gamma$
```

ElimLin Algorithm Revisited

Nicolas T. Courtois[1], Pouyan Sepehrdad[2], Petr Sušil[2], and Serge Vaudenay[2]

[1] University College London, UK
[2] EPFL, Lausanne, Switzerland
n.courtois@ucl.ac.uk,
{pouyan.sepehrdad, petr.susil, serge.vaudenay}@epfl.ch

Abstract. ElimLin is a simple algorithm for solving polynomial systems of multivariate equations over small finite fields. It was initially proposed by Courtois to attack DES. It can reveal some hidden linear equations existing in the ideal generated by the system. We report a number of key theorems on ElimLin. Our main result is to characterize ElimLin in terms of a sequence of intersections of vector spaces. It implies that the linear space generated by ElimLin is invariant with respect to any variable ordering during elimination and substitution. This can be seen as surprising given the fact that it eliminates variables. On the contrary, monomial ordering is a crucial factor in Gröbner Bases algorithms such as F4. Moreover, we prove that the result of ElimLin is invariant with respect to any affine bijective variable change. Analyzing an overdefined dense system of equations, we argue that to obtain more linear equations in the succeeding iteration in ElimLin some restrictions should be satisfied. Finally, we compare the security of LBlock and MIBS block ciphers with respect to algebraic attacks and propose several attacks on Courtois Toy Cipher version 2 (CTC2) with distinct parameters using ElimLin.

Keywords: block ciphers, algebraic cryptanalysis, systems of sparse polynomial equations of low degree

[Breaking a good cipher should require]
"as much work as solving a system of simultaneous equations
in a large number of unknowns of a complex type."

Claude Elwood Shannon [46]

1 Introduction

Various techniques exist in cryptanalysis of symmetric ciphers. Some involve statistical analysis and some are purely deterministic. One of the latter methods is *algebraic attack* formulated as early as 1949 by Shannon [46].

Any algebraic attack consists of two distinct stages:

- Writing the cipher as a system of polynomial equations of low degree often over $\mathsf{GF}(2)$ or $\mathsf{GF}(2^k)$, which is feasible for any cipher [49, 21, 43].
- Recovering the secret key by solving such a large system of polynomial equations.

Algebraic attacks have been successful in breaking several stream ciphers (see [1, 19, 12, 25, 20, 15, 24, 11] for instance) and a few block ciphers such as Keeloq [38] and GOST [16], but they are not often as successful as statistical attacks. On the other hand, they often require low data complexity. This is not the case for statistical attacks.

General purpose algebraic attack techniques were developed in the last few years by Courtois, Bard, Meier, Faugère, Raddum, Semaev, Vielhaber, Dinur and Shamir to solve these systems [17, 22, 21, 19, 12, 31, 32, 45, 48, 24, 25]. The problem of solving such polynomial systems of multivariate equations is called MQ problem and is known to be NP hard for a random system. Currently, for a random system in which the number of equations is equal to the number of unknowns, there exists no technique faster than an exhaustive key search which can solve such systems. On the other hand, the equations derived from symmetric ciphers turn out to be overdefined and sparse for most ciphers. So, they might be easier to solve. This sparsity is coming from the fact that due to the limitations in hardware and the need for lightweight algorithms, simple operations arise in the definition of cryptosystems. They are also overdefined due to the non-linear operations.

The traditional method for solving overdefined polynomial systems of equations are known to be various Gröbner basis algorithms such as Buchberger algorithm [10], F4 and F5 [31, 32] and XL [22]. The most critical drawback of the Gröbner basis approach is the elimination step where the degree of the system increases. This leads to an explosion in memory space and in the worst case scenario they run in double exponential time and even the most current efficient implementations of Faugère algorithm [31, 32] under PolyBoRi framework [8] or Magma [41] are not capable of handling *large* systems of equations efficiently. On the other hand, they are faster than other methods for overdefined dense systems or when the equations are over $\mathsf{GF}(q)$ where $q > 2$. In fact, together with SAT solvers, they are currently the most successful methods for solving polynomial systems.

Nevertheless, due to the technical reasons mentioned above, the system of equations extracted from symmetric ciphers turns out to be sparse. Unfortunately, the Gröbner basis algorithms can not exploit this property. In such cases, algorithms such as XSL [21], SAT solving techniques [4, 28, 3], Raddum-Semaev algorithm [45] and ElimLin [17] are of interest.

In this paper, we study the elimination algorithm ElimLin that falls within the remit of Gröbner basis algorithms, though it is conceptually much simpler and is based on a mix of simple linear algebra and substitution. It maintains the degree of the equations and it does not require any fixed ordering on the set of all monomials. This is not the case for the Gröbner basis algorithms, where monomial ordering is a prominent factor. On the contrary, we need to work with ad-hoc monomial orderings to preserve the sparsity and make it run faster. This simple algorithm reveals some hidden linear equations existing in the ideal generated by the system. We show in Sec. 7 that ElimLin does not find all such linear equations.

As far as the authors are aware, no clue has been found yet which demonstrates that ElimLin at some stage stops working. This does not mean that ElimLin can break any system. As mentioned earlier, for a random system this problem is NP hard and Gröbner basis algorithms behave much better for such dense random systems. But, the equations derived from cryptosystems are often not random (see [33] for the huge difference between a random system and the algebraic representation of cryptographic protocols). What we mean here is that if for some small number of rounds ElimLin performs well but then it stops working for more rounds, we can increase the number of samples and it will become effective again (see the Appendix). The bottleneck is having an efficient data structure for implementing ElimLin together with a rigorous theory behind it to anticipate its behaviour. These two factors are currently missing in the literature.

Except two simple theorems by Bard (see Chapter 12, Section 5 of [4]), almost nothing has been done regarding the theory behind ElimLin. As ElimLin can also be used as a pre-processing step in any algebraic attack, building a proper theory is vital for improving the state of the art algebraic attacks. We are going to shed some lights on the way this ad-hoc algorithm works and the theory behind it.

In this paper, we show that the output of ElimLin is invariant with respect to any variable ordering. This is a surprising result, i.e., while the spaces generated are different depending on how substitution is performed, we prove that their intersection is exactly the same. Furthermore, we prove that no affine bijective variable change can modify the output of ElimLin. Then, we prove a theorem on how the number of linear equations evolves in each iteration of ElimLin.

An unannounced competition is currently running for designing lightweight cryptographic primitives. This includes several designs which have appeared in the last few years (see [7, 23, 40, 35, 30, 37, 47, 2, 36, 6]). These designs mainly compete over the gate equivalent (GE) and throughput. This might not be a fair comparison of efficiency, since they do not provide the same level of security with respect to distinct types of attacks. In this paper, we compare the two lightweight Feistel-based block ciphers MIBS [39] and LBlock [50] and show that with the same number of rounds, LBlock provides a much lower level of security compared to MIBS with respect to algebraic attacks. In fact, we attack both ciphers with ElimLin and F4 algorithm. Finally, we provide several algebraic attacks against Courtois Toy Cipher version 2 (CTC2) with distinct parameters using ElimLin.

In Sec. 2, we elaborate the ElimLin algorithm. Then, we remind some basic theorems on ElimLin in Sec. 3. As our main contribution (Theorem 7), we prove in Sec. 4 that ElimLin can be formulated as an intersection of vector spaces. We also discuss its consequences in Sec. 4.2 and prove a theorem regarding the evolution of linear equations in Sec. 4.3. We perform some attacks simulations on CTC2, LBlock and MIBS block ciphers in Sec. 5.2, 5.3, 5.4 respectively. In Sec. 6 we compare ElimLin and F4. We mention some open problems and a conjecture in Sec. 7 and we conclude. Finally, in the Appendix we give a toy example on ElimLin and discuss the effect of multiple samples.

2 ElimLin Algorithm

ElimLin stands for **Elim**inate **Lin**ear and it is a technique for solving polynomial systems of multivariate equations of low degree d mostly: 2, 3, or 4 over a finite field specifically $\mathsf{GF}(2)$. Originally, it was proposed in [17] to attack DES. It broke 5-round DES. Later, it was applied to break 5-round PRESENT block cipher [44] and to analyze the resistance of Snow 2.0 stream cipher against algebraic attacks [18]. It is a simple but a powerful algorithm which can be applied to any symmetric cipher and is capable of breaking their reduced versions. There is no specific requirement for the system except that there should exist at least one linear term, otherwise ElimLin trivially fails. The key question for such an algorithm is to predict its behavior. Currently, very similar to most other types of algebraic attacks such as [48, 24, 25], multiple parts of the algorithm are heuristic, so it is worthwhile to prove which factors can improve its results, make it run faster or does not have any influence on its ultimate result. This will yield a better understanding of how ElimLin works and can be extended.

ElimLin is composed of two sequential distinct stages, namely:

- *Gaussian Elimination:* All the linear equations in the linear span of initial equations are found. They are the intersection between two vector spaces: The vector space spanned by all monomials of degree 1 and the vector space spanned by all equations.
- *Substitution:* Variables are iteratively eliminated in the whole system based on linear equations until there is no linear equation left. Consequently, the remaining system has fewer variables.

This routine is iterated until no linear equation is obtained in the linear span of the system. See Fig. 1 for a more precise definition of the algorithm. We also give a toy system of equations in the Appendix and solve it with ElimLin.

Clearly, the algorithm shall depend on ordering strategies to apply in step 5, 11, and 12 of Fig. 1. We will see that it is not, i.e., the span of the resulting $\mathcal{S}_L$ is invariant.

We observe that new linear equations are derived in each iteration of the algorithm that did not exist in the former spans. This phenomenon is called *avalanche effect* in ElimLin and is the

```
Input : A system of polynomial equations $\mathcal{S}^0 = \{\mathsf{Eq}_1^0, \ldots, \mathsf{Eq}_{m_0}^0\}$ over $\mathsf{GF}(2)$.
Output : An updated system of equations $\mathcal{S}^T$ and a system of linear equations $\mathcal{S}_L$.
Set $\mathcal{S}_L \leftarrow \emptyset$ and $\mathcal{S}^T \leftarrow \mathcal{S}^0$ and $k \leftarrow 1$.
repeat
  Perform Gaussian elimination $\mathsf{Gauss}(.)$ on $\mathcal{S}^T$ with an arbitrary ordering of equations and monomials to
  eliminate non-linear monomials.
  Set $\mathcal{S}_{L'} \leftarrow$ Linear equations from $\mathsf{Gauss}(\mathcal{S}^T)$.
  Set $\mathcal{S}^T \leftarrow \mathsf{Gauss}(\mathcal{S}^T) \setminus \mathcal{S}_{L'}$.
  Set flag.
  for all $\ell \in \mathcal{S}_{L'}$ in an arbitrary order do
    if $\ell$ is a trivial equation then
      if $\ell$ is unsolvable then
        Terminate and output "No Solution".
      end if
    else
      Unset flag.
      Let $x_{t_k}$ be a monomial from $\ell$.
      Substitute $x_{t_k}$ in $\mathcal{S}^T$ and $\mathcal{S}_L'$ using $\ell$.
      Insert $\ell$ in $\mathcal{S}_L$.
      $k \leftarrow k + 1$
    end if
  end for
until flag is set.
Output $\mathcal{S}_T$ and $\mathcal{S}_L$.
```

Fig. 1. ElimLin algorithm.

consequence of Theorem 7. At the end, the system is solved linearly (when $\mathcal{S}_L$ is large enough) or ElimLin fails. If the latter occurs, we can increase the data complexity [3] and re-run the attack.

3 State of the Art Theorems

The only theoretical analysis of ElimLin was done by Bard in [4]. He proved the following theorem and corollary for **one** iteration of ElimLin:

Theorem 1 ([4]). *All linear equations in the linear span of a polynomial equation system $\mathcal{S}^0$ are found in the linear span of linear equations derived by performing the first iteration of ElimLin algorithm on the system.*

The following corollary (also from [4]) is the direct consequence of the above theorem.

Corollary 2. *The linear equations generated after performing the first Gaussian elimination in ElimLin algorithm form a basis for all possible linear equations in the linear span of the system.*

This shows that any method to perform Gaussian elimination does not affect the linear space obtained at an arbitrary iteration of ElimLin. All linear equations derived from one method exist in the linear span of the equations cumulated from another method. This is trivial to see.

4 Algebraic Representation of ElimLin

4.1 ElimLin as an Intersection of Vector Spaces

We also formalize ElimLin in an algebraic way. This representation is used in proving Theorem 7. First, we define some notations.

[3] For instance, the number of plaintext-ciphertext pairs.

We call an *iteration* a Gaussian elimination preceding a substitution; The system of equations for $\mathsf{ElimLin}$ can be stored as a matrix $\mathcal{M}_\alpha$ of dimension $m_\alpha \times T_\alpha$, where each m_α rows represents an equation and each T_α columns represents a monomial at iteration α. Also, r_α denotes the rank of $\mathcal{M}_\alpha$. Let n_α be the number of variables at iteration α. We use a reverse lexicographical ordering of columns during Gaussian elimination to accumulate linear equations in the last rows of the matrix. In fact, we use the same matrix representation as described in [4].

Let $K = \mathsf{GF}(2)$ and $x = (x_1, \ldots, x_n)$ be a set of free variables. We denote by $K[x]$ the ring of multivariate polynomials over K. For $\mathcal{S} \subset K[x]$, we denote $\mathsf{Span}\,(\mathcal{S})$ the K-vector subspace of $K[x]$ spanned by $\mathcal{S}$. Let $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_n)$ be a power vector in $\mathbf{N}^n$. The term x^γ is defined as the product $x^\gamma = x_1^{\gamma_1} \times x_2^{\gamma_2} \times \cdots \times x_n^{\gamma_n}$. The total degree of x^γ is defined as $deg(x^\gamma) \overset{\mathsf{def}}{=} \gamma_1 + \gamma_2 + \cdots + \gamma_n$. Let $\mathsf{Ideal}\,(\mathcal{S})$ be the ideal spanned by $\mathcal{S}$ and $\mathsf{Root}\,(\mathcal{S})$ be the set of all tuples $m \in K^n$ such that $f(m) = 0$ for all $f \in \mathcal{S}$. Let

$$R_d = \mathsf{Span}\,(\text{monomials of degree} \leq d)\,/\mathsf{Ideal}\left(x_1^2 - x_1, x_2^2 - x_2, \ldots, x_n^2 - x_n\right)$$

Let $\mathcal{S}^\alpha$ be $\mathcal{S}^T$ after the α-th iteration of $\mathsf{ElimLin}$ and $\mathcal{S}^0$ be the initial system. Moreover, n_L^α is the number of non-trivial linear equations in $\mathcal{S}_{L'}$ at the α-th iteration. We denote $\mathcal{S}_L^\alpha$ the $\mathcal{S}_L$ after the α-th iteration. Also,

$$C^\alpha \overset{\mathsf{def}}{=} \#\mathcal{S}_L^\alpha$$

Let assume that $\mathcal{S}^0$ has degree bounded by d. We denote by $\mathsf{Var}(f)$ the set of variables x_i expressed in f. Let $x_{t_1}, \ldots, x_{t_k}$ be the sequence of eliminated variables. We define $\mathsf{V}_k = \{x_1, \ldots, x_n\} \backslash \{x_{t_1}, \ldots, x_{t_k}\}$. Also, let $\ell_1, \ell_2, \ldots, \ell_k$ be the sequence of linear equations as they are used during elimination (step 11 of Fig. 1). Hence, we have $x_{t_k} \in \mathsf{Var}(\ell_k) \subseteq \mathsf{V}_{k-1}$.

We prove the following crucial lemma which we use later to prove Theorem 7.

Lemma 3. *After the α-th iteration of ElimLin, an arbitrary equation Eq_i^α in the system $(\mathcal{S}^\alpha \cup \mathcal{S}_L^\alpha)$ for an arbitrary i can be represented as*

$$\mathsf{Eq}_i^\alpha = \sum_{t=1}^{m_0} \beta_{ti}^\alpha \cdot \mathsf{Eq}_t^0 + \sum_{t=1}^{C^\alpha} \ell_t(x) \cdot g_{ti}^\alpha(x) \tag{1}$$

where $\beta_{ti}^\alpha \in K$ and $g_{ti}^\alpha(x)$ is a polynomial in R_{d-1} and $\mathsf{Var}(g_{ti}^\alpha) \subseteq \mathsf{V}_t$.

Proof. Let x_{t_1} be one of the monomials existing in the first linear equation $\ell_1(x)$ and this specific variable is going to be eliminated. Substituting x_{t_1} in an equation $x_{t_1} \cdot h(x) + z(x)$, where $h(x)$ has degree at most $d - 1$, $x_{t_1} \notin \mathsf{Var}(h)$ and $x_{t_1} \notin \mathsf{Var}(z)$ is identical to subtracting $h(x) \cdot \ell_1(x)$. Consequently, the proof follows by induction on α.

□

Now, we prove the inverse of the above lemma.

Lemma 4. *For each i and each α, there exists $\beta_{ti}'^\alpha \in K$ and $g_{ti}'^\alpha(x)$ such that*

$$\mathsf{Eq}_i^0 = \sum_{t=1}^{m_\alpha} \beta_{ti}'^\alpha \cdot \mathsf{Eq}_t^\alpha + \sum_{t=1}^{C^\alpha} \ell_t(x) \cdot g_{ti}'^\alpha(x) \tag{2}$$

where $g_{ti}'^\alpha(x)$ is a polynomial in R_{d-1} and $\mathsf{Var}(g_{ti}'^\alpha) \subseteq \mathsf{V}_t$.

Proof. Gaussian elimination and substitution are invertible operations. We can use a similar induction as the previous lemma to prove the above equation.

□

In the next lemma, we prove that $\mathcal{S}_L^\alpha$ contains all linear equations which can be written in the form of Eq. (1).

Lemma 5. *If there exists $\ell \in R_1$ and some β_t and $g_t''(x)$ such that*

$$\ell(x) = \sum_{t=1}^{m_0} \beta_t \cdot \mathsf{Eq}_t^0 + \sum_{t=1}^{C^\alpha} \ell_t(x) \cdot g_t''(x) \tag{3}$$

at iteration α, where $g_t''(x)$ is a polynomial in R_{d-1}, then there exists $u_t \in K$ and $v_t \in K$ such that

$$\ell(x) + \sum_{t=1}^{C^\alpha} u_t \cdot \ell_t(x) = \sum_{t=1}^{m_\alpha} v_t \cdot \mathsf{Eq}_t^\alpha$$

So, $\ell(x) \in \mathsf{Span}\,(\mathcal{S}_L^\alpha)$.

Proof. We define u_k iteratively: u_k is the coefficient of x_{t_k} in

$$\ell(x) + \sum_{t=1}^{k-1} u_t \cdot \ell_t(x)$$

for $k = 1, \ldots, C^\alpha$. So, $\mathsf{Var}(\ell(x) + \sum_{t=1}^{k} u_t \cdot \ell_t(x)) \subseteq \mathsf{V}_k$. By substituting Eq_i^0 from Eq. (2) in Eq. (3) and integrating u_t and g_t'' in $g_{ti}'^{\alpha}$, we obtain

$$\underbrace{\ell(x) + \sum_{t=1}^{C^\alpha} u_t \cdot \ell_t(x)}_{\subseteq \mathsf{V}_1} = \underbrace{\sum_{t=1}^{m_\alpha} v_t \cdot \mathsf{Eq}_t^\alpha}_{\subseteq \mathsf{V}_1} + \underbrace{\sum_{t=1}^{C^\alpha} \ell_t(x) \cdot g_t'(x)}_{\Longrightarrow \subseteq \mathsf{V}_1} \tag{4}$$

with $g_t'(x) \in R_{d-1}$. All $g_t'(x)$ where $t > 1$ can be written as $\bar{g}_t(x) + x_{t_1} \cdot \bar{\bar{g}}_t(x)$ with $\mathsf{Var}(\bar{g}_t) \subseteq \mathsf{V}_1$, $\mathsf{Var}(\bar{\bar{g}}_t) \subseteq \mathsf{V}_1$ and $\bar{\bar{g}}_t(x) \in R_{d-2}$. Since,

$$\ell_1(x) \cdot g_1'(x) + \ell_t(x) \cdot g_t'(x) = \ell_1(x) \cdot \underbrace{\left(g_1'(x) + \ell_t(x) \cdot \bar{\bar{g}}_t(x)\right)}_{\text{new } g_1'(x)} + \underbrace{\ell_t(x)}_{\subseteq \mathsf{V}_1} \cdot \underbrace{\left(\bar{g}_t(x) + \bar{\bar{g}}_t(x) \cdot (x_{t_1} - \ell_1(x))\right)}_{(\text{new } g_t'(x))\ \subseteq \mathsf{V}_1}$$

we can re-arrange the sum in Eq. (4) using the above representation and obtain $\mathsf{Var}(g_t') \subseteq \mathsf{V}_1$ for all $t > 1$. Also, x_{t_1} only appears in $\ell_1(x)$ and $g_1'(x)$. So, the coefficient of x_{t_1} in the expansion of $\ell_1(x) \cdot g_1'(x)$ must be zero. In fact, we have

$$\begin{aligned} \ell_1(x) \cdot g_1'(x) &= (x_{t_1} + (\ell_1(x) - x_{t_1})) \cdot (\bar{g}_1(x) + x_{t_1} \cdot \bar{\bar{g}}_t(x)) \\ &= x_{t_1} \cdot (\bar{\bar{g}}_1(x) \cdot (1 + \ell_1(x) - x_{t_1}) + \bar{g}_1(x)) + \bar{g}_1(x) \cdot (\ell_1(x) - x_{t_1}) \end{aligned}$$

So, $\bar{g}_1(x) = \bar{\bar{g}}_1(x) \cdot (x_{t_1} - \ell_1(x) - 1)$ and we deduce,

$$g_1'(x) = \bar{\bar{g}}_1(x) \cdot (\ell_1(x) + 1)$$

over $\mathsf{GF}(2)$. But, then

$$\ell_1(x) \cdot g_1'(x) = 0$$

over R, since $\ell_1(x) \cdot (\ell_1(x) + 1) = 0$. Finally, we iterate and obtain

$$\ell(x) + \sum_{t=1}^{C^\alpha} u_t \cdot \ell_t(x) = \sum_{t=1}^{m_\alpha} v_t \cdot \mathsf{Eq}_t^\alpha$$

□

```
Input : A set $\mathcal{S}^0$ of polynomial equations in $R_d$.
Output : A system of linear equations $\mathcal{S}_L$.
Set $\bar{\mathcal{S}_L} := \emptyset$.
repeat
    $\bar{\mathcal{S}_L} \leftarrow \mathsf{Span}\left(\mathcal{S}^0 \cup (R_{d-1} \times \bar{\mathcal{S}_L})\right) \cap R_1$
until $\bar{\mathcal{S}_L}$ unchanged
Output $\mathcal{S}_L$: a basis of $\bar{\mathcal{S}_L}$.
```

Fig. 2. ElimLin algorithm from another perspective.

From another perspective, ElimLin algorithm can be represented as in Fig. 2. In fact, as a consequence of Lemma 3 and Lemma 5, Fig. 2 presents a unique characterization of $\mathsf{Span}(\mathcal{S}_L)$ in terms of a fixed point:

Lemma 6. *At the end of ElimLin, $\mathsf{Span}(\mathcal{S}_L)$ is the smallest subset $\bar{\mathcal{S}_L}$ of R_1, such that*

$$\bar{\mathcal{S}_L} = \mathsf{Span}\left(\mathcal{S}^0 \cup (R_{d-1} \times \bar{\mathcal{S}_L})\right) \cap R_1$$

Proof. By induction, at step α we have $\bar{\mathcal{S}_L} \subseteq \mathsf{Span}(\mathcal{S}_L^\alpha)$, using Lemma 5. Also, $\mathcal{S}_L^\alpha \subseteq \bar{\mathcal{S}_L}$ using Lemma 3. So, $\bar{\mathcal{S}_L} = \mathsf{Span}(\mathcal{S}_L^\alpha)$ at step α. Since $\bar{\mathcal{S}_L} \mapsto \mathsf{Span}\left(\mathcal{S}^0 \cup (R_{d-1} \times \bar{\mathcal{S}_L})\right) \cap R_1$ is increasing, we obtain the above equation.

□

ElimLin eliminates variables, thus it looks very unexpected that the number of linear equations in each step of the algorithm is invariant with respect to any variable ordering in the substitution step and the Gaussian elimination. We finally prove this important invariant property. Concretely, we formalize ElimLin as a sequence of intersection of vector spaces. Such intersection in each iteration is between the vector space spanned by the equations and the vector space generated by all monomials of degree 1 in the system. This implies that if ElimLin runs for α iterations (finally succeeds or fails), it can be formalized as a sequence of intersections of α pairs of vector spaces. These intersections of vector spaces only depend on the vector space of the initial system.

Theorem 7. *The following relations exist after running ElimLin on a polynomial system of equations Q:*

1. $\mathsf{Root}\left(\mathcal{S}^0\right) = \mathsf{Root}(\mathcal{S}^T \cup \mathcal{S}_L)$
2. *There is no linear equation in $\mathsf{Span}\left(\mathcal{S}^T\right)$.*
3. *$\mathsf{Span}(\mathcal{S}_L)$ is uniquely defined by $\mathcal{S}^0$.*
4. *$\mathcal{S}_L$ consists of linearly independent linear equations.*
5. *The complexity is $O\left(n_0^{d+1} m_0^2\right)$, where d is the degree of the system and n_0 and m_0 are the initial number of variables and equations, respectively.*

Proof (1). Due to Lemma 3 and Lemma 4, $\mathcal{S}^0$ and $(\mathcal{S}^T \cup \mathcal{S}_L)$ are equivalent. So, a solution of $\mathcal{S}^0$ is also a solution of $(\mathcal{S}^T \cup \mathcal{S}_L)$ and visa versa.

Proof (2). Since ElimLin stops on $\mathcal{S}^T$, the Gaussian reduction did not find any linear polynomial.

Proof (3). Due to Lemma 6.

Proof (4). $\mathcal{S}_L$ includes a basis for $\bar{\mathcal{S}_L}$. So, it consists of linearly independent equations.

Proof (5). n_0 is an upper bound on $\#\mathcal{S}_L$ due to the fact that $\mathcal{S}_L$ consists of linearly independent linear equations. So, the number of iterations is bounded by n_0. The total number of monomials is bounded by

$$T_0 \leq \sum_{i=0}^{d} \binom{n_0}{i} = O\left(n_0^d\right)$$

The complexity of Gaussian elimination is $O(m_0^2 T_0)$, since we have T_0 columns and m_0 equations. Therefore overall, the complexity of $\mathsf{ElimLin}$ is $O\left(n_0^{d+1} m_0^2\right)$.

□

4.2 Affine Bijective Variable Change

In the next theorem, we prove that the result of $\mathsf{ElimLin}$ algorithm does not change for any affine bijective variable change. It is an open problem to find an appropriate non-linear variable change which improves the result of $\mathsf{ElimLin}$ algorithm.

Theorem 8. *Any affine bijective variable change $A : \mathsf{GF}(2)^{n_0} \to \mathsf{GF}(2)^{n_0}$ on a n_0-variable system of equations $\mathcal{S}^0$ does not affect the result of $\mathsf{ElimLin}$ algorithm, implying that the number of linear equations generated at each iteration is invariant with respect to an affine bijective variable change.*

Proof. In Lemma 6, we showed that $\mathsf{Span}\,(\mathcal{S}_L)$ is the output of the algorithm in Fig. 2, iterating

$$\bar{\mathcal{S}_L} \leftarrow \mathsf{Span}\left(\mathcal{S}^0 \cup (R_{d-1} \times \bar{\mathcal{S}_L})\right) \cap R_1$$

We represent the composition of a polynomial f_1 with respect to A by $\mathsf{Com}(f_1)$. We then show that there is a commutative diagram

$$\begin{tikzcd}
\mathcal{S}^0 \arrow[r, "\mathsf{Com}"] \arrow[d, "\mathsf{ElimLin}"'] & \mathsf{Com}(\mathcal{S}^0) \arrow[d, "\mathsf{ElimLin}"] \\
\bar{\mathcal{S}_L} \arrow[r, "\mathsf{Com}"'] & \mathsf{Com}(\bar{\mathcal{S}_L})
\end{tikzcd}$$

We consider two parallel executions of the algorithm in Fig. 2, one with $\mathcal{S}^0$ and the other with $\mathsf{Com}(\mathcal{S}^0)$.

If we compose the polynomials in $\mathcal{S}^0$ with respect to A, in the above relation R_{d-1} remains the same. Since the transformation A is affine,

$$\mathsf{Com}(\mathsf{Span}\left(\mathcal{S}^0 \cup (R_{d-1} \times \bar{\mathcal{S}_L})\right) \cap R_1) = \mathsf{Span}\left(\mathsf{Com}(\mathcal{S}^0) \cup (R_{d-1} \times \mathsf{Com}(\bar{\mathcal{S}_L}))\right) \cap R_1$$

So, at each iteration, the second execution has the result of applying Com to the result of the first one.

□

4.3 Linear Equations Evolution

An open problem regarding ElimLin is to predict how the number of linear equations evolves in the preceding iterations. In the following theorem, we give a necessary (but not sufficient) condition for a dense overdefined system of equations to have an additional linear equation in the next iteration of ElimLin. Proving the similar result for a sparse system is not straightforward.

Theorem 9. *If we apply ElimLin to an overdefined dense system of quadratic equations over* $\mathsf{GF}(2)$*, For* $n_L^{\alpha+1} > n_L^{\alpha}$ *to hold, it is necessary to have*

$$\frac{b_\alpha}{2} - a_\alpha < n_L^\alpha < \frac{b_\alpha}{2} + a_\alpha$$

where $b_\alpha = 2n_\alpha - 1$ *and* $a_\alpha = \frac{\sqrt{b_\alpha^2 - 8n_L^\alpha}}{2}$.

Proof. For the system to generate linear equations, it is necessary that the *sufficient rank condition* [4] is satisfied. More clearly, we must have $r_\alpha > T_\alpha - 1 - n_\alpha$, otherwise no linear equations will be generated. This is true if the system of equations is overdefined. Hence, we obtain,

$$n_L^\alpha = r_\alpha + n_\alpha + 1 - T_\alpha \tag{5}$$

If some columns of the matrix $\mathcal{M}_\alpha$ are pivotless, it will shift the diagonal strand of ones to the right. Therefore, n_L^α will be more than what the above equation expresses. Assuming the system of equations is dense, this phenomenon happens with a very low probability. Suppose the above equation is true with high probability, then we get

$$n_L^{\alpha+1} = r_{\alpha+1} + n_{\alpha+1} + 1 - T_{\alpha+1} \tag{6}$$

In the $(\alpha+1)$-th iteration, the number of variables is reduced by n_L^α. Thus, $n_{\alpha+1} = n_\alpha - n_L^\alpha$. If the system of equations is dense, in a quadratic system,

$$T_\alpha = \binom{n_\alpha}{2} + n_\alpha + 1$$

and so,

$$T_{\alpha+1} = \binom{n_\alpha - n_L^\alpha}{2} + n_\alpha - n_L^\alpha + 1$$

Consequently, we have

$$T_\alpha - T_{\alpha+1} = n_L^\alpha \left(n_\alpha - \frac{1}{2}(n_L^\alpha - 1) \right) \tag{7}$$

Therefore, using Eq. (5), Eq. (6) and Eq. (7), we obtain,

$$\begin{aligned} n_L^{\alpha+1} &= (r_{\alpha+1} - r_\alpha) + (r_\alpha + n_\alpha - T_\alpha + 1) + n_L^\alpha(-\tfrac{1}{2}n_L^\alpha + n_\alpha - \tfrac{1}{2}) \\ &= n_L^\alpha(-\tfrac{1}{2}n_L^\alpha + n_\alpha + \tfrac{1}{2}) - (r_\alpha - r_{\alpha+1}) \end{aligned}$$

If $n_L^{\alpha+1} > n_L^\alpha$, then $n_L^\alpha(-\frac{1}{2}n_L^\alpha + n_\alpha + \frac{1}{2}) - (r_\alpha - r_{\alpha+1}) > n_L^\alpha$ and this leads to

$${n_L^\alpha}^2 + (1 - 2n_\alpha)n_L^\alpha + 2(r_\alpha - r_{\alpha+1}) < 0$$

$\Delta = (1 - 2n_\alpha)^2 - 8(r_\alpha - r_{\alpha+1})$, and if the above inequality holds, Δ should be positive and assuming $b_\alpha = 2n_\alpha - 1$, then, $b_\alpha - \sqrt{\Delta} < 2n_L^\alpha < b_\alpha + \sqrt{\Delta}$.

Considering Δ is positive, $n_\alpha > \sqrt{2(r_\alpha - r_{\alpha+1})} + \frac{1}{2}$. We also know that $r_{\alpha+1} \leq r_\alpha - n_L^\alpha$, which together lead to $n_\alpha > \frac{1}{2} + \sqrt{2n_L^\alpha}$. Therefore, for $n_L^{\alpha+1} > n_L^\alpha$, it is necessary to have $n_\alpha > \frac{1}{2} + \sqrt{2n_L^\alpha}$, but not visa versa. Simplifying $b_\alpha - \sqrt{\Delta} < 2n_L^\alpha < b_\alpha + \sqrt{\Delta}$ and deploying $r_\alpha - r_{\alpha+1} \geq n_L^\alpha$ results in

$$b_\alpha - 2a_\alpha < 2n_L^\alpha < b_\alpha + 2a_\alpha$$

where $b_\alpha = 2n_\alpha - 1$ and $2a_\alpha = \sqrt{b_\alpha^2 - 8n_L^\alpha}$.

Notice that $n_\alpha > \frac{1}{2} + \sqrt{2n_L^\alpha}$, which was obtained in the first stage of the proof, has been originated from the fact that $b_\alpha^2 - 8n_L^\alpha$ should be non-negative.

□

5 Attacks Simulations

In this section, we present our experimental results against CTC2, LBlock and MIBS block ciphers. The simulations for CTC2 were run on an ordinary PC with a 1.8 Ghz CPU and 2 GB RAM. All the other simulations were run on an ordinary PC with a 2.8 Ghz CPU and 4 GB RAM. The amount of RAM required by our implementation is negligible.

In our attacks, we build a system of quadratic equations with variables representing plaintext, ciphertext, key and state bits, which allows to express the system of equations of high degree as quadratic equations. Afterwards, for each sample we set the plaintext and ciphertext according to the result of the input/output of the cipher. In order to test the efficiency of the algebraic attack, we guess some bits of the key and set the key variables corresponding to the guess. Then, we run the solver (ElimLin, F4 or SAT solver) to recover the remaining key bits and test whether the guess was correct. Therefore, the complexity of our algebraic attack can be bounded by $2^g \cdot \mathcal{C}(solver)$, where $\mathcal{C}(solver)$ represents the running time of the solver and g is the number of bits we guess. $\mathcal{C}(solver)$ is represented as the the *"Running Time"* in all the following tables.

For a comparison with a brute force attack, we consider a fair implementation of the cipher, which requires 10 CPU cycles per round. This implies that the algebraic attack against t rounds of the cipher is faster than an exhaustive search for the 1.8 Ghz and 2.8 Ghz CPU iff recovering c bits of the key is faster than $5.55 \cdot t \cdot 2^{c-31}$ and $3.57 \cdot t \cdot 2^{c-31}$ seconds respectively. This is already twice faster than the complexity of exhaustive search. All the attacks reported in the following tables are faster than exhaustive search with the former argument. In fact, we consider the cipher to be broken for some number of rounds if the algebraic attack that recovers $(\#key - g)$ key bits is faster than an exhaustive key search over $(\#key - g)$ bits of the key.

5.1 Simulations Using F4 Algorithm under PolyBoRi Framework

The most efficient implementation of the F4 algorithm is available under PolyBoRi framework [9] running alone or under SAGE algebra system. PolyBoRi is a C++ library designed to compute Gröbner basis of an ideal applied to Boolean polynomials. A Python interface is used, surrounding the C++ core. It uses zero-suppressed binary decision diagrams (ZDDs) [34] as a high level data structure for storing Boolean polynomials. This representation stores the monomials more efficiently in memory and it makes the Gröbner basis computation faster compared to other algebra systems.

We use polybori-0.8.0 for our attacks. Together with ElimLin, we also attack LBlock and MIBS with F4 algorithm and then compare its efficiency with ElimLin.

5.2 Simulations on CTC2

Courtois Toy Cipher (CTC) is an SPN-based block cipher devised by Courtois [14] as a toy cipher to evaluate algebraic attacks on smaller variants of cryptosystems. It was designed to show that it is possible to break a cipher using an ordinary PC deploying a small number of known or chosen plaintext-ciphertext pairs.

Since the system of equations of well-known ciphers such as AES is often large, it is not feasible by the current algorithms and computer capacities to solve them in a reasonable time, therefore smaller but similar versions such as CTC can be exploited to evaluate the resistance of ciphers against algebraic cryptanalysis. This turns out to yield a benchmark on understanding the algebraic structure of ciphers. Ultimately, this might lead to break of a larger system later.

CTC was not designed to be resistant against all known types of attacks like linear and differential cryptanalysis. Nevertheless, in [26], it was attacked by linear cryptanalysis. Subsequently, CTC Version 2 or CTC2 was proposed [13] to resolve the flaw exists in CTC structure. CTC2 is very similar to CTC with a few changes. It is an SPN-based network with scalable number of rounds, block and key size. For the full specification, refer to [13]. In CT-RSA 2009, differential and differential-linear attacks could reach up to 8 rounds of CTC2 [27], but as stated before, the objective of the CTC designer was not applying statistical attacks to his design. Finally, there is a cube attack on 4 rounds of one variant of this cipher in [42].

Since block size and key size are flexible in CTC2 cipher, we break various versions with distinct parameters (see Table 1) using ElimLin. The block size is specified by a parameter B, which specifies the number of parallel S-boxes per round. CTC2 S-box is 3×3, hence the block size is computed as $3B$. We guess some LSB bits of the key and we show that recovering the remaining is faster than exhaustive search.

It might be possible that during the intermediate steps of ElimLin, a quadratic equation in only key bits (possibly linear) appears. In such cases, approximately $\mathcal{O}(\#key^2)$ samples are enough to break the system. This is due to the fact that we can simply change the plaintext-ciphertext pair and generate a new linearly independent equation in the key. Finally, when we have enough such equations, we solve a system of quadratic equations in only key bits using the linearization technique. When such phenomenon occurs, intuitively the cipher is close to be broken but not yet. We can increase the number of samples and most often it makes the cipher thoroughly collapse.

5.3 Simulations on LBlock

LBlock is a new lightweight Feistel-based block cipher, aimed at constrained environments, such as RFID tags and sensor networks [50] proposed at ACNS 2011. It operates on 64-bit blocks, uses a key of 80 bits and iterates 32 rounds. For a detailed specification of the cipher, refer to [50]. As far as the authors are aware, there is currently no cryptanalysis results published on this cipher.

We break 8 rounds of LBlock using 6 samples deploying an ordinary PC by ElimLin. Our results are summarized in Table 2. In the same scenario, PolyBoRi crashes due to running out of memory.

5.4 Simulations on MIBS

Similar to the LBlock block cipher, MIBS is also a lightweight Feistel-based block cipher, aimed at constrained environments, such as RFID tags and sensor networks [39]. It operates on 64-bit blocks, uses keys of 64 or 80 bits and iterates 32 rounds. For a detailed specification of the cipher, see [39].

Table 1. CTC2 simulations using ElimLin up to 6 rounds with distinct parameters.

B	N_r	$\#key$	g	Running Time^1 (in hours)	Running Time^2 (in hours)	Data	Attack notes
16	3	48	0	0.03		5 KP	ElimLin
16	3	48	0		0.12	14 KP	ElimLin
64	3	192	155		0.03	1 KP	ElimLin
85	3	255	210		0.04	1 KP	ElimLin
16	4	48	0	0.01		2 CP	ElimLin
16	4	48	0		0.05	4 CP	ElimLin
40	4	120	85	0.00		1 KP	ElimLin
40	4	120	85		0.84	16 KP	ElimLin
48	4	144	100		0.12	4 KP	ElimLin
64	4	192	148	0.05		1 KP	ElimLin
64	4	192	155		2.21	5 KP	ElimLin
85	4	255	220	0.29		1 KP	ElimLin
85	4	255	215	0.64		1 KP	ElimLin
85	4	255	220		0.26	2 KP	ElimLin
85	4	255	215		0.90	3 KP	ElimLin
85	4	255	210		1.33	4 KP	ElimLin
16	5	48	0	3		8 CP	ElimLin
40	5	120	85		0.03	2 CP	ElimLin
32	6	96	60	2.5		16 CP	ElimLin
40	6	120	80	1		8 CP	ElimLin
64	6	192	155	2.4		4 CP	ElimLin
85	6	255	210	3		2 CP	ElimLin
85	6	255	220		3	16 CP	ElimLin
85	6	255	210		180.5	64 CP	ElimLin
128	6	384	344		4.5	2 CP	ElimLin

B : Number of S-boxes per round. To obtain the block size, B should be multiplied by 3.
N_r : Number of rounds
g: Number of guessed LSB of the key
Running Time^1: Running time until we achieve equations only in key variables (no other internal variables). When this is achieved, the cipher is close to be broken, but not yet (see Sec. 5.2).
Running Time^2: Attack running time for recovering ($\#key - g$) bits of the key.
KP: Known plaintext
CP: Chosen plaintext

Currently, the best cryptanalysis results is a linear attack reaching 18-round MIBS with data complexity 2^{61} and time complexity of 2^{76} [5]. In fact, statistical attacks often require very large number of samples. This is not always achievable in practice.

We break 4 and 3 rounds of MIBS80 and MIBS64 using 32 and 2 samples deploying an ordinary PC by ElimLin. Our results are summarized in Table 3. In 2 out of 3 experiments, PolyBoRi crashes due to running out of memory. This is the first algebraic analysis of the cipher.

The designers in [39] have evaluated the security of their cipher with respect to algebraic attacks. They used the complexity of XSL algorithm for this evaluation, which is not a precise measurement for evaluating resistance of a cipher against algebraic attacks, since effectiveness of XSL is still controversial and under speculation. There are better methods such as SAT solvers [3] which solve MQ problem faster than expected due to the system being overdefined and sparse.

Let assume XSL can be precise enough to evaluate the security of a cipher with respect to algebraic attacks. According to [21, 39], the complexity of XSL can be evaluated with the work factor. For MIBS, work factor is computed as:

$$\mathsf{WF} = \Gamma^{\omega} \left((\text{Block Size}) \cdot N_r^2 \right)^{\omega \lceil \frac{T}{r} \rceil}$$

Table 2. Algebraic attack complexities on reduced-round LBlock using ElimLin and PolyBoRi.

N_r	$\#key$	g	Running Time in hours	Data	Attack notes
8	80	32	0.252	6 KP	ElimLin
8	80	32	crashed	6 KP	PolyBoRi

N_r : Number of rounds
g: Number of guessed LSB of the key
KP: Known plaintext
CP: Chosen plaintext

where Γ is a parameter which depends only on the S-box. For MIBS, $\Gamma = 85.56$. The value $r = 21$ is the number of equations the S-box can be represented with. $T = 37$ is the number of monomials in that representation. $\omega = 2.37$ is the exponent of the Gaussian elimination complexity. The work factor for attacking 5-round MIBS is $\mathsf{WF} = 2^{65.65}$ which is worse than an exhaustive key search for MIBS64. Deploying SAT solving techniques using MiniSAT 2.0 [29], we can break 5 rounds of MIBS64 (see Table 3). Our strategy is exactly the same as [3]. Table 3 already shows that we can do better than $2^{65.65}$ for MIBS64. We can perform a very similar attack on MIBS80. This already shows that considering the complexity of XSL is not a precise measure to evaluate the security of a cipher against algebraic cryptanalysis. Complexity of attacking such system with XL is extremely high.

We believe that due to the similarity between the structure of MIBS and LBlock, we can compare them with respect to algebraic attacks. As can be seen from the table of attacks, LBlock is much weaker. This is not surprising though, since the linear layer of LBlock is much weaker than MIBS, since it is nibble-wise instead of bit-wise. So, we could attack twice more rounds of LBlock. Thus, although LBlock is lighter with respect to the number of gates, but it provides a lower level of security with respect to algebraic attacks.

Table 3. Algebraic attack complexities on reduced-round MIBS using ElimLin, PolyBoRi and MiniSAT 2.0.

N_r	$\#key$	g	Running Time (in hours)	Data	Attack notes
4	80	20	0.137	32 KP	ElimLin
4	80	20	crashed	32 KP	PolyBoRi
5	64	16	0.395	6 KP	MiniSAT 2.0
5	64	16	crashed	6 KP	PolyBoRi
3	64	0	0.006	2 KP	ElimLin
3	64	0	0.002	2 KP	PolyBoRi

N_r : Number of rounds
g: Number of guessed LSB of the key
KP: Known plaintext
CP: Chosen plaintext

6 A Comparison Between ElimLin and F4

Gröbner basis and SAT solving techniques are currently the most successful methods for solving polynomial systems of equations. However, both these approaches have significant restrictions. The main bottleneck of the Gröbner basis techniques is the memory requirement and therefore most of the Gröbner basis attacks use relatively small number of samples. ElimLin algorithm on the other hand requires a large number of samples to work.

Table 2 and Table 3 show that F4 requires too much memory and crashes for a large number of samples. At the same time, ElimLin algorithm is slightly slower than PolyBoRi implementation attacking 2 samples of 3-round MIBS64 as in Table 3. This demonstrates that ElimLin can be more effective than PolyBoRi and vice versa, depending on memory requirements of PolyBoRi. However, whenever the system is solvable by ElimLin, our experiments revealed that PolyBoRi does not give a significant advantage over ElimLin because the memory requirements are too high.

The advantage of ElimLin compared to F4 is the fact that it always performs substitution using only linear equations, which means that the number of monomials is bounded by $\mathcal{O}\left(n_0^2\right)$ and the degree of the system is maintained. On the other hand, ElimLin may require many more samples to succeed compared to F4 or F5. While the Gröbner basis algorithms may yield a solution for a few samples, the success of ElimLin is determined by the number of samples provided to the algorithm. The evaluation of the number of sufficient samples in ElimLin is still an open problem.

We demonstrated that even a simple linear algebra technique can outperform the more sophisticated Gröbner basis algorithms, mainly due to the structural properties of the system of equations of a cryptographic primitive (such as sparsity). ElimLin takes advantage of such structural properties and uncovers the hidden linear equations using multiple samples. According to our experiments, F4 or F5 algorithms do not seem to be able to take advantage of these structural properties as would be expected, which results in higher memory requirements than would be necessary and ultimately their failure for large systems.

7 Further Work and Some Conjectures

An interesting area of research is to estimate the number of linear equations in ElimLin or anticipate how this number evolves in the succeeding iterations or evaluate after how many iterations ElimLin finishes. Also, to anticipate how many samples is enough to make the system collapse by ElimLin. Last but not least, it is prominent to find a very efficient method for implementing ElimLin and to find the most appropriate data structure to choose.

There are some evidence which illustrate that ElimLin does not reveal all hidden linear equations in the structure of the cipher up to a specific degree. We give an example, demonstrating such an evidence:

Assume there exists an equation in the system which can be represented as $\ell(x)g(x)+1=0$ over $\mathsf{GF}(2)$, where $\ell(x)$ is a polynomial of degree one and $g(x)$ is a polynomial of degree at most $d-1$. Running ElimLin on this single equation trivially fails. But, if we multiply both sides of the equation by $\ell(x)$, we obtain $\ell(x)g(x)+\ell(x)=0$. Summing these two equations, we derive $\ell(x)=1$. This hidden linear equation can be simply captured by the XL algorithm, but can not be captured by ElimLin. There exist multiple other examples which demonstrate that ElimLin does not generate all the hidden linear equations. A further work to this paper can be to extend ElimLin, leading to the capturing of all such linear equations.

For big ciphers, for example the full AES, it is also plausible that:

Conjecture 1 *For each number of rounds X, there exists Y such that AES is broken by ElimLin given Y Chosen or Known Plaintext-Ciphertext pairs.*

Disproving the above conjecture leads to the statement that “AES can not be broken by algebraic attack at degree 2”. But maybe this conjecture is true, then the capacities of the

ElimLin attack are considerable and it works for any number of rounds X. As a consequence, if for $X = 14$ this Y is not too large, say less than 2^{64}, the AES-256 will be broken faster than brute force by ElimLin at degree 2, which is much simpler than Gröbner basis objective of breaking it at degree 3 or 4 with 1 KP.

ElimLin is a polynomial time algorithm. If it can be shown that a polynomial number of samples is enough to gain a high success rate for ElimLin, this can already be considered a breakthrough in cryptography. Unfortunately, the correctness of this statement is not clear.

Conclusion

In this paper, we proved that ElimLin can be formulated in terms of a sequence of intersections of vector spaces. We showed that different monomial orderings and any affine bijective variable change do not influence the result of the algorithm. We did some predictions on the evolution of linear equations in the succeeding iterations in ElimLin. We presented multiple attacks deploying ElimLin against CTC2, LBlock and MIBS block ciphers.

References

1. F. Armknecht and G. Ars. Algebraic Attacks on Stream Ciphers with Gröbner Bases. *Gröbner Bases, Coding, and Cryptography*, pages 329–348, 2009.
2. J. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia. Quark: A Lightweight Hash. In *CHES*, volume 6225, pages 1–15, 2010.
3. G. Bard, N. Courtois, and C. Jefferson. Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers. In *presented at ECRYPT workshop Tools for Cryptanalysis*, 2007. http://eprint.iacr.org/2007/024.pdf.
4. G.V. Bard. *Algebraic Cryptanalysis.* Springer, 2009.
5. A. Bay, J. Nakahara, and S. Vaudenay. Cryptanalysis of Reduced-Round MIBS Block Cipher. In *CANS*, volume 6467, pages 1–19. Springer, 2010.
6. A. Bogdanov, M. Kneževicć, G. Leander, D. Toz, K. Varici, and I. Verbauwhede. SPONGENT: A Lightweight Hash Function. In *CHES*, volume 6917, pages 312–325, 2011.
7. A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *CHES*, volume 4727, pages 450–466, 2007.
8. M. Brickenstein and A. Dreyer. PolyBoRi: A framework for Gröbner basis computations with Boolean polynomials. In *Electronic Proceedings of MEGA 2007*, 2007. http://www.ricam.oeaw.ac.at/mega2007/electronic/26.pdf.
9. M. Brickenstein and A. Dreyer. PolyBoRi: A framework for Gröbner basis computations with Boolean polynomials. In *Electronic Proceedings of MEGA*, 2007.
10. B. Buchberger. Bruno Buchberger's PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. *Journal of Symbolic Computation*, 41(3-4):475–511, 2006.
11. N. Courtois. Higher Order Correlation Attacks, XL Algorithm and Cryptanalysis of Toyocrypt. In *ICISC*, volume 2587, pages 182–199, 2002.
12. N. Courtois. Fast Algebraic Attacks on Stream Ciphers with Linear Feedback. In *Advances in Cryptology - CRYPTO*, volume 2729, pages 176–194. Springer, 2003.
13. N. Courtois. CTC2 and Fast Algebraic Attacks on Block Ciphers Revisited. In *Cryptology ePrint Archive*, 2007. http://eprint.iacr.org/2007/152.pdf.
14. N. Courtois. How Fast can be Algebraic Attacks on Block Ciphers? In *Symmetric Cryptography*, volume 07021 of *Dagstuhl Seminar Proceedings*, 2007.
15. N. Courtois. The Dark Side of Security by Obscurity - and Cloning MiFare Classic Rail and Building Passes, Anywhere, Anytime. In *SECRYPT*, pages 331–338, 2009.
16. N. Courtois. Algebraic Complexity Reduction and Cryptanalysis of GOST. In *Cryptology ePrint Archive*, 2011. http://eprint.iacr.org/2011/626.
17. N. Courtois and G.V. Bard. Algebraic Cryptanalysis of the Data Encryption Standard. In *IMA Int. Conf.*, volume 4887, pages 152–169. Springer, 2007.
18. N. Courtois and B. Debraize. Algebraic Description and Simultaneous Linear Approximations of Addition in Snow 2.0. In *ICICS*, volume 5308, pages 328–344. Springer, 2008.

19. N. Courtois and W. Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In *Advances in Cryptology - EUROCRYPT*, volume 2656, pages 345–359. Springer, 2003.
20. N. Courtois, S. O'Neil, and J. Quisquater. Practical Algebraic Attacks on the Hitag2 Stream Cipher. In *ISC*, volume 5735, pages 167–176, 2009.
21. N. Courtois and J. Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In *Asiacrypt*, volume 2501, pages 267–287. Springer, 2002.
22. N. Courtois, A. Shamir, J. Patarin, and A. Klimov. Efficient Algorithms for Solving Overfined Systems of Multivariate Polynomial Equations. In *Advances in Cryptology, Eurocrypt*, volume 1807, pages 392–407. Springer, 2000.
23. C. De Canniére, O. Dunkelman, and M. Knezević. KATAN and KTANTAN - a family of small and efficient hardware-oriented block ciphers. In *CHES*, volume 5747, pages 272–288, 2009.
24. I. Dinur and A. Shamir. Cube Attacks on Tweakable Black Box Polynomials. In *Advances in Cryptology - EUROCRYPT*, volume 5479, pages 278–299. Springer, 2009.
25. I. Dinur and A. Shamir. Breaking Grain-128 with Dynamic Cube Attacks. In *FSE*, volume 6733, pages 167–187. Springer, 2011.
26. O. Dunkelman and N. Keller. Linear Cryptanalysis of CTC. In *Cryptology ePrint Archive*, 2006. http://eprint.iacr.org/2006/250.pdf.
27. O. Dunkelman and N. Keller. Cryptanalysis of CTC2. In *CT-RSA*, volume 5473, pages 226–239. Springer, 2009.
28. N. Eén and N. Sörensson. MiniSat 2.0. An open-source SAT solver package. http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/.
29. N. Een and N. Sorensson. Minisat - A SAT Solver with Conflict-Clause Minimization. In *Theory and Applications of Satisfiability Testing*, 2005.
30. D. Engels, M.O. Saarinen, P. Schweitzer, and E.M. Smith. The Hummingbird-2 Lightweight Authenticated Encryption Algorithm. In *RFIDsec*, volume 7055, pages 19–31, 2011.
31. J. Faugère. A new effcient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, 1999.
32. J. Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *Symbolic and Algebraic Computation - ISSAC*, page 7583, 2002.
33. G. Fusco and E. Bach. Phase transition of multivariate polynomial systems. *Journal of Mathematical Structures in Computer Science*, 19(1), 2009.
34. M. Ghasemzadeh. *A New Algorithm for the Quantified Satisfiability Problem, Based on Zero-suppressed Binary Decision Diagrams and Memoization.* PhD thesis, University of Potsdam, Germany, 2005.
35. Z. Gong, S. Nikova, and Y.W. Law. KLEIN: A New Family of Lightweight Block Ciphers. In *RFIDsec*, volume 7055, pages 1–18, 2011.
36. J. Guo, T. Peyrin, and A. Poschmann. The PHOTON Family of Lightweight Hash Functions. In *CRYPTO*, volume 6841, pages 222–239, 2011.
37. J. Guo, T. Peyrin, A. Poschmann, and M.J.B Robshaw. The LED Block Cipher. In *CHES*, volume 6917, pages 326–341, 2011.
38. S. Indesteege, N. Keller, O. Dunkelman, E. Biham, and B. Preneel. A Practical Attack on Keeloq. In *EUROCRYPT*, volume 4965, pages 1–18, 2008.
39. M. Izadi, B. Sadeghiyan, S. Sadeghian, and H. Arabnezhad. MIBS: A New Lightweight Block Cipher. In *CANS*, volume 5888, pages 334–348. Springer, 2009.
40. L.R. Knudsen, G. Leander, A. Poschmann, and M.J.B. Robshaw. PRINTcipher: A Block Cipher for IC-Printing. In *CHES*, volume 6225, pages 16–32, 2010.
41. Magma, software package. http://magma.maths.usyd.edu.au/magma/.
42. P. Mroczkowski and J. Szmidt. The Cube Attack on Courtois Toy Cipher. In *Cryptology ePrint Archive*, 2009. eprint.iacr.org/2009/497.pdf.
43. S. Murphy and M. Robshaw. Essential Algebraic Structure within AES. In *Advances in Cryptology - CRYPTO*, volume 2442, pages 1–16. Springer-Verlag, 2002.
44. J. Nakahara, P. Sepehrdad, B. Zhang, and M. Wang. Linear (Hull) and Algebraic Cryptanalysis of the Block Cipher PRESENT. In *CANS*, volume 5888, pages 58–75. Springer, 2009.
45. H. Raddum and I. Semaev. Solving Multiple Right Hand Sides linear equations. *Journal of Designs, Codes and Cryptography*, 49(1-3):147–160, 2008.
46. C.E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28, 1949.
47. K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai. Piccolo: An Ultra-Lightweight Blockcipher. In *CHES*, volume 6917, pages 342–357, 2011.
48. M. Vielhaber. Breaking ONE.FIVIUM by AIDA an Algebraic IV Differential Attack. In *Cryptology ePrint Archive*, 2007. http://eprint.iacr.org/2007/413.
49. R. Weinmann. Evaluating Algebraic Attacks on the AES. Master's thesis, Technische Universität Darmstadt, 2003.
50. W. Wu and L. Zhang. LBlock: A Lightweight Block Cipher. In *ACNS*, volume 6715, pages 327–344. Springer, 2011.

A A Toy Example of ElimLin

Let assume that we have the following overdefined system of multivariate equations over $\mathsf{GF}(2)$ with 5 variables $x_1, \ldots, x_5$ and 6 equations,

$$\begin{cases} x_1x_2 + x_1x_3 + x_2x_5 + x_3x_5 + x_2 + x_4 + x_5 + 1 = 0 \\ x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 = 0 \\ x_1x_4 + x_2x_3 + x_3 + 1 = 0 \\ x_1x_4 + x_1x_5 + x_2x_5 + x_1 = 0 \\ x_1x_5 + x_2x_3 + x_3x_5 + x_1 + x_3 + x_4 = 0 \\ x_1x_5 + x_2x_3 + x_3x_5 + x_5 + x_4 + x_2 + 1 = 0 \end{cases}$$

We perform Gaussian elimination on the system, and obtain,

$$\begin{cases} x_1x_2 + x_2x_4 + x_2x_5 + x_3x_5 + x_2 + x_3 + x_4 + x_5 = 0 \\ x_1x_3 + x_2x_4 + x_3 + 1 = 0 \\ x_1x_4 + x_2x_3 + x_3 + 1 = 0 \\ x_1x_5 + x_2x_3 + x_3x_5 + x_1 + x_3 + x_4 = 0 \\ x_2x_5 + x_3x_5 + x_4 + 1 = 0 \\ x_1 + x_2 + x_3 + x_5 + 1 = 0 \end{cases}$$

The linear equation we obtain is used for the substitution of variable $x_5 = x_1 + x_2 + x_3 + 1$. Then, we perform Gaussian elimination on the system again. We derive

$$\begin{cases} x_2x_4 + x_1 = 0 \\ x_1x_4 + x_2x_3 + x_3 + 1 = 0 \\ x_1x_2 + x_1 + x_3 + x_4 = 0 \\ x_1x_3 + x_1 + x_3 + 1 = 0 \\ x_4 + x_3 + x_1 = 0 \end{cases}$$

The new linear equation is used for the substitution of the variable $x_4 = x_3 + x_1$. After the substitution, we perform the Gaussian elimination again and obtain,

$$\begin{cases} x_2x_3 + x_1 = 0 \\ x_1x_3 + x_3 + 1 = 0 \\ x_1x_2 = 0 \\ x_1 = 0 \end{cases}$$

We derive a new linear equation $x_1 = 0$. Consequently, we perform substitution and Gaussian elimination, which yields,

$$\begin{cases} x_2x_3 = 0 \\ x_3 + 1 = 0 \end{cases}$$

The new linear equation we obtain is $x_3 = 1$. After the substitution of this variable, we obtain $x_2 = 0$. Hence, we have gathered 5 linear equations in 5 variables as follows, which can be simply solved by

$$\begin{cases} x_1 + x_2 + x_3 + x_5 + 1 = 0 \\ x_1 + x_3 + x_4 = 0 \\ x_1 = 0 \\ x_3 + 1 = 0 \\ x_2 = 0 \end{cases}$$

leading to $x_1 = x_2 = x_5 = 0$ and $x_3 = x_4 = 1$.

B Multiple Samples Effect on ElimLin

A prominent question regarding ElimLin is that how we can extract more linear equations from the structure of the cipher. One approach is to use more samples. On one hand, having multiple instances increases the number of variables, since the state bits are totally distinct, on the other hand all the instances share the same key bits. The speed in which the number of equations increases is higher than which of the number of variables. Consequently, we expect that at one moment the system is solved. We have performed many experiments using ElimLin. In some cases, we would expect it to fail, since the number of linear equations at some stage dropped significantly, but those few equations could cause the system to collapse at the consequent iterations and the system is finally solved. We give an example to be more clear:

We attacked 8-round LBlock [50] block cipher with 32 LSB key bits fixed starting from 1 pair to 8 pairs (see Sec. 5.3 for details). As can be observed from Tables 4 to Table 11, the cipher is unbroken for 5 plaintext-ciphertext pairs, but then 6 pairs is enough to break the system. We use the following legend in those tables.

Legend
I : iteration number in ElimLin.
n : number of variables.
m_0 : number of initial equations.
AvS : the average number of monomials per equation (It represents the sparsity).
T : number of monomials.
n_L : number of linear equations.
n_c : cumulative number of linear equations.

For instance, in Table 9 we start with $n_0 = 8\,784$ variables and $m_0 = 27\,758$ equations. We then eliminate $n_L^1 = 7\,528$ variables at iteration 1. The variable elimination is repeated until the iteration 15, when we finish with 2 variables and $n_L^{15} = 2$ linear equations. As can be observed, at the last iteration the number of cumulative linear equations n_c is the same as the initial number of variables n_0. This implies that we only need to solve a linear system of equations in 8 784 variables and the system is solved. This is not the case for smaller number of samples. For instance, in Table 8, at the last iteration we finish with 499 variables and no linear equations. This implies that all 499 variables should be guessed.

Table 4. Attacking 8-round LBlock with 1 pair and 32 LSB key bits guessed.

I	n	m_0	AvS	T	n_L	n_c
1	2064	5174	3	4249	1768	1768
2	296	5174	7	5678	42	1810
3	254	5174	6	5035	16	1826
4	238	5174	7	4868	3	1829
5	235	5174	7	5178	0	1829

Table 5. Attacking 8-round LBlock with 2 pairs and 32 LSB key bits guessed.

I	n	m_0	AvS	T	n_L	n_c
1	3408	8822	3	7385	2920	2920
2	488	8822	8	10855	85	3005
3	403	8822	9	11545	48	3053
4	355	8822	11	11955	22	3027
5	333	8822	15	13779	8	3035
6	325	8822	16	13729	0	3035

Table 6. Attacking 8-round LBlock with 3 pairs and 32 LSB key bits guessed.

I	n	m_0	AvS	T	n_L	n_c
1	4752	13110	3	10521	4072	4072
2	680	13110	8	16032	128	4200
3	552	13110	9	17495	83	4283
4	469	13110	13	18190	40	4323
5	429	13110	17	19913	21	4344
6	408	13110	20	20547	5	4349
7	403	13110	22	20843	1	4350
8	402	13110	21	20725	1	4351
9	401	13110	21	20561	0	4351

Table 7. Attacking 8-round LBlock with 4 pairs and 32 LSB key bits guessed.

I	n	m_0	AvS	T	n_L	n_c
1	6096	17839	3	13657	5224	5224
2	872	17839	8	21209	171	5395
3	701	17839	10	24035	118	5511
4	583	17839	14	25396	66	5577
5	517	17839	20	27955	40	5617
6	477	17839	26	31106	21	5638
7	456	17839	31	31611	16	5654
8	440	17839	28	28934	1	5655
9	439	17839	28	28717	0	5655

Table 8. Attacking 8-round LBlock with 5 pairs and 32 LSB key bits guessed.

I	n	m_0	AvS	T	n_L	n_c
1	7440	22730	3	16793	6376	6376
2	1064	22730	8	26386	214	6590
3	850	22730	10	30368	151	6741
4	699	22730	15	33097	91	6832
5	608	22730	23	37005	55	6887
6	553	22730	32	39058	35	6922
7	518	22730	35	37629	16	6938
8	502	22730	34	35748	1	6939
9	501	22730	35	35709	1	6940
10	500	22730	34	35509	1	6941
11	499	22730	34	34649	0	6941

Table 9. Attacking 8-round LBlock with 6 pairs and 32 LSB key bits guessed.

I	n	m_0	AvS	T	n_L	n_c
1	8784	27758	3	19929	7528	7528
2	1256	27758	7	31563	257	7785
3	999	27758	10	36607	189	7974
4	810	27758	17	41351	123	8097
5	687	27758	26	48066	83	8180
6	604	27758	34	46540	41	8221
7	563	27758	36	42910	15	8236
8	548	27758	37	41469	8	8244
9	540	27758	32	39312	24	8268
10	516	27758	16	29409	126	8394
11	390	27758	19	23370	108	8502
12	282	27758	20	14889	87	8589
13	195	27758	15	9157	122	8711
14	73	27758	4	1454	71	8782
15	2	27758	0	3	2	8784

Table 10. Attacking 8-round LBlock with 7 pairs and 32 LSB key bits guessed.

I	n	m_0	AvS	T	n_L	n_c
1	10128	32815	3	23065	8680	8680
2	1448	32815	7	36740	300	8980
3	1148	32815	10	42889	228	9208
4	920	32815	17	48974	157	9365
5	763	32815	30	58471	111	9476
6	652	32815	40	55476	47	9523
7	605	32815	42	51967	20	9543
8	585	32815	37	47625	25	9568
9	560	32815	19	36163	141	9709
10	419	32815	21	27254	126	9835
11	293	32815	20	16116	145	9980
12	148	32815	8	4960	142	10122
13	6	32815	0	8	6	10128

Table 11. Attacking 8-round LBlock with 8 pairs and 32 LSB key bits guessed.

I	n	m_0	AvS	T	n_L	n_c
1	11472	37945	3	26201	9832	9832
2	1640	37945	7	41917	343	10175
3	1297	37945	9	47974	268	10443
4	1029	37945	17	55084	186	10629
5	843	37945	30	65625	129	10758
6	714	37945	39	63385	57	10815
7	657	37945	41	57671	22	10837
8	635	37945	34	50898	21	10858
9	614	37945	20	40883	161	11019
10	453	37945	23	30905	144	11163
11	309	37945	22	19850	160	11323
12	149	37945	8	5108	145	11468
13	4	37945	0	6	4	11472

Short-output universal hash functions and their use in fast and secure message authentication

Long Hoang Nguyen and Andrew William Roscoe[⋆]

Oxford University Department of Computer Science
Email: {Long.Nguyen, Bill.Roscoe}@cs.ox.ac.uk

Abstract. Message authentication codes usually require the underlining universal hash functions to have a long output so that the probability of successfully forging messages is low enough for cryptographic purposes. To take advantage of fast operation on word-size parameters in modern processors, long-output universal hashing schemes can be securely constructed by concatenating several instances of short-output primitives. In this paper, we describe a new method for short-output universal hash function termed *digest*() suitable for very fast software implementation and applicable to secure message authentication. The method possesses a higher level of security relative to other well-studied short-output universal hashing schemes. Suppose that the universal hash output is fixed at one word of b bits, then the collision probability of ours is 2^{1-b} compared to 6×2^{-b} of MMH, whereas $2^{-b/2}$ of NH within UMAC is far away from optimality. In addition to message authentication codes, we show how short-output universal hashing is applicable to manual authentication protocols where universal hash keys are used in a very different and interesting way.

1 Introduction

Universal hash functions (or UHFs) first introduced by Carter and Wegman [6, 31] have many applications in computer science, including randomised algorithms, database, cryptography and many others. A UHF takes two inputs which are a key k and a message m: $h(k, m)$, and produces a fixed-length output. Normally what we require of a UHF is that for any pair of distinct messages m and m' the collision probability $h(k, m) = h(k, m')$ is small when key k is randomly chosen from its domain. In the majority of cryptographic uses, UHFs usually have long outputs so that combinatorial search is made infeasible. For example, UHFs can be used to build secure message authentication codes or MAC schemes where the intruder's ability to forge messages is bounded by the collision probability of the UHF. In a MAC, parties share a secret universal hash key and an encryption key, a message is authenticated by hashing it with the shared universal hash key and then encrypting the resulting hash. The encrypted hash value together with the message is transmitted as an authentication tag that can be validated by the verifier. We note however that our new construction presented here applies to other cryptographic uses of universal hashing, e.g., manual authentication protocols as seen later as well as non-cryptographic applications.

Since operating on short-length values of 16, 32 or 64 bits is fast and convenient in ordinary computers, long-output UHFs can be securely constructed by concatenating the results of multiple instances of short-output UHFs to increase computational efficiency. To our knowledge, a number of short-output UHF schemes have been proposed, notably MMH (Multilinear-Modular-Hashing) of Halevi and Krawczyk [9] and NH within UMAC of Black et al. [4]. We note that widely studied polynomial universal hashing schemes PolyP, PolyQ [14] and GHASH [24] can also be designed to produce a short output. While polynomial based UHFs only require short and fixed length keys, they suffer from two unpleasant properties relating to security and computational efficiency as will be discussed later in the paper.

Our main contribution presented in Section 3 is the introduction of a new short-output UHF algorithm termed $digest(k, m)$ that can be efficiently computed on any modern microprocessors.

[⋆] Long Nguyen's work on this paper was supported by a research grant from the US Office of Naval Research. Andrew William Roscoe's was partially supported by funding from the US Office of Naval Research.

The main advantage of ours is that it provides a higher level of security regarding both collision and distribution probabilities relative to MMH and NH described in Section 4. Our *digest*() algorithm operates on word-size parameters via word multiplication and word addition instructions, i.e. finite fields or non-trivial reductions are excluded, because the emphasis is on high speed implementation using software.

Let us suppose that the universal hash output is fixed at one word of b bits then the collision probability of ours is 2^{1-b} compared to 6×2^{-b} of MMH, whereas $2^{-b/2}$ of NH is much weaker in security. For clarity, the security bounds of our constructions as well as MMH and NH are independent of the length of message being hashed, which is the opposite of polynomial universal hashing schemes mentioned earlier. For multiple-word output universal hashing constructions as required in MACs, the advantage in security of ours becomes more apparent. When the universal hash output is extended to n words or $n \times b$ bits for any $n \in \mathbb{N}^*$, then the collision probability of ours is 2^{n-nb} as opposed to $6^n \times 2^{-nb}$ of MMH and $2^{-nb/2}$ of NH. There is however a trade-off between security and computational cost as illustrated by our estimated operation counts and software implementations of these constructions. On a 1GHz AMD Athlon processor, one version of *digest*() (where the collision probability ϵ_c is 2^{-31}) achieves peak performance of 0.53 cycles/byte (or cpb) relative to 0.31 cpb of MMH (for $\epsilon_c = 2^{-29.5}$) and 0.23 cpb of NH (for $\epsilon_c = 2^{-32}$). Another version of $digest(k, m)$ for $\epsilon_c = 2^{-93}$ achieves peak performance of 1.54 cpb. For comparison purpose, 12.35 cpb is the speed of SHA-256 recorded on our computer. A number of files that provide the software implementations in C programming language of NH, MMH and our proposed constructions can be downloaded from [1] so that the reader can run them and adapt them for other uses of the short-output universal hash schemes.

We will briefly discuss the motivation of designing (and the elegant graphical structure of) our *digest*() scheme which, we have recently discovered, relates to the well-studied multiplicative universal hashing schemes of Dietzfelbinger et al. [7], Krawczyk [12, 13] and Mansour et al. [18]. The latter algorithms are however not efficient when the input message is of a significant size.

Although researchers from cryptographic community have mainly studied UHFs to construct message authentication codes, we would like to point out that short-output UHF on its own has found applications in manual authentication protocols [2, 8, 15, 17, 19, 10, 20–23, 25, 30]. In the new family of authentication protocols, data authentication can be achieved without the need of passwords, shared private keys as required in MACs, or any pre-existing security infrastructures such as a PKI. Instead human owners of electronic devices who seek to exchange their data authentically would need to manually compare a short string of bits that is often outputted from a UHF. Since humans can only compare short strings, the UHF ideally needs to have a short output of say 16 or 32 bits. There is however a fundamental difference in the use of universal hash keys between manual authentication protocols and message authentication codes, it will be clear in Section 5 that none of the short-output UHF schemes including ours should be used directly in the former. Thus we will propose a general framework where any short-output UHFs can be used efficiently and securely to digest a large amount of data in manual authentication protocols.

While existing universal hashing methods are already as fast as the rate information is generated, authenticated and transmitted in high-speed network traffic, one may ask whether we need another universal hashing algorithm. Besides keeping up with network traffic, as excellently explained by Black et al. [4] — *the goal is to use the smallest possible fraction of the CPU's cycles (so most of the machine's cycles are available for other work), by the simplest possible hash mechanism, and having the best proven bounds.* This is relevant to MACs as well as manual authentication protocols where large data are hashed into a short string, and hence efficient short-output UHF constructions possessing a higher (or optimal) level of security are needed.

2 Notation and definitions

We define M, K and b the bit length of the message, the key and the output of a universal hash function. We denote $R = \{0,1\}^K$, $X = \{0,1\}^M$ and $Y = \{0,1\}^b$.

Definition 1. [12, 13] A ϵ-balanced universal hash function, $h : R \times X \to Y$, must satisfy that for every $m \in X \setminus \{0\}$ and $y \in Y$: $\Pr_{\{k \in R\}}[h(k,m) = y] \leq \epsilon$

Many existing UHF constructions [4, 9, 12, 13] as well as our newly proposed scheme rely on (integer or matrix) multiplications of message and key, and hence non-zero input message is required; for otherwise $h(k, 0) = 0$ for any key $k \in R$.

Definition 2. [13, 27] A ϵ-almost universal hash function, $h : R \times X \to Y$, must satisfy that for every $m, m' \in X$ $(m \neq m')$: $\Pr_{\{k \in R\}}[h(k,m) = h(k,m')] \leq \epsilon$

Since it is useful particularly in manual authentication protocols discussed later to have both the collision and distribution probabilities bounded, we combine Definitions 1 and 2 as follows

Definition 3. An ϵ_d-balanced and ϵ_c-almost universal hash function, $h : R \times X \to Y$, satisfies

- for every $m \in X \setminus \{0\}$ and $y \in Y$: $\Pr_{\{k \in R\}}[h(k,m) = y] \leq \epsilon_d$
- for every $m, m' \in X$ $(m \neq m')$: $\Pr_{\{k \in R\}}[h(k,m) = h(k,m')] \leq \epsilon_c$

3 Integer multiplication construction

We first discuss the multiplicative universal hashing algorithm of Dietzfelbinger et al. [7] which obtains a very high level of security. Although this scheme is not efficient with long input data, it strongly relates to our *digest*() method that make use of word multiplication instructions.

We note that there are two other universal hashing schemes which use arithmetic that computer likes to do to increase computational efficiency, namely MMH of Halevi and Krawczyk [9] and NH of Black et al. [4]. Both of which will be compared against our construction in Section 4.

3.1 Multiplicative universal hashing

Suppose that we want to compute a b-bit universal hash of a M-bit message, then the universal hash key k is drawn randomly from $R = \{1, 3, \ldots, 2^M - 1\}$, i.e. k must be odd. Dietzfelbinger et al. [7] define:

$$h(k, m) = (k * m \bmod 2^M) \text{ div } 2^{M-b}$$

It was proved that the collision probability of this construction is $\epsilon_c = 2^{1-b}$ on equal length inputs [7]. While this has a simple description, for long input messages of several kilobytes or megabytes, such as documents and images, it will become very time consuming to compute the integer multiplication involved in this algorithm.

3.2 Word multiplicative construction

In this section, we will define and prove the security of a new short-output universal hashing scheme termed *digest*(k, m) that can be calculated using word multiplications instead of an arbitrarily long integer multiplication as seen in Equation 1 or an example from Figure 1.

Let us divide message m into b-bit blocks $\langle m_1, \ldots, m_{t=M/b} \rangle$. An $(M + b)$-bit key $k = \langle k_1, \ldots, k_{t+1} \rangle$ is selected randomly from $R = \{0,1\}^{M+b}$. A b-bit *digest*(k, m) is defined as

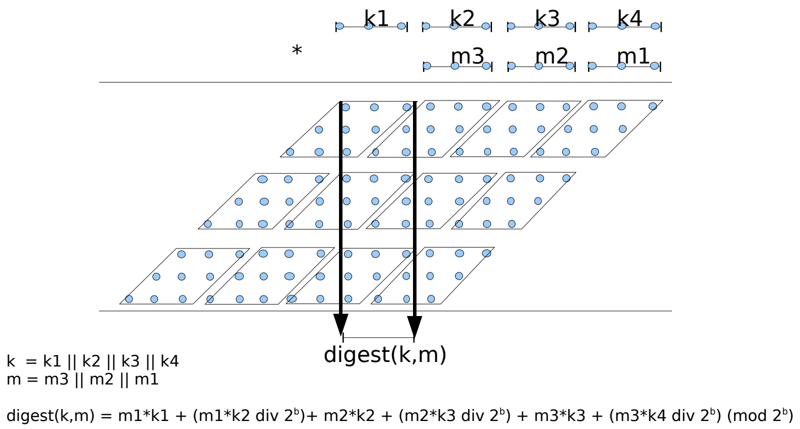

k1
k2
k3
k4
*
m3
m2
m1
digest(k,m)
k = k1 || k2 || k3 || k4
m = m3 || m2 || m1
digest(k,m) = m1*k1 + (m1*k2 div 2^b)+ m2*k2 + (m2*k3 div 2^b) + m3*k3 + (m3*k4 div 2^b) (mod 2^b)

Fig. 1. A b-bit output $digest(k, m)$: each parallelogram represents the expansion of a word multiplication between a b-bit key block and a b-bit message block.

$$digest(k, m) = \sum_{i=1}^{t} [m_i * k_i + (m_i * k_{i+1} \text{ div } 2^b)] \text{ mod } 2^b \tag{1}$$

Here, * refers to a word multiplication of two b-bit blocks which produces a $2b$-bit output, whereas both '+' and $\sum$ are additions modulo 2^b. It should be noted that (div 2^b) is equivalent to a right shift ($>> b$).

To see why this scheme is related to the multiplicative method of Dietzfelbinger et al. [7], one can study Figure 1 where all word multiplications involved in Equation 1 are elegantly arranged into the same shape as the overlap of the expanded multiplication between m and k.[1]

Operation count. To give an estimated operation count for an implementation of $digest()$, which will be subsequently compared against universal hashing schemes MMH and NH, we consider a machine with the same properties as one used by Halevi and Krawczyk [9]:[2]

- ($b = 32$)-bit machine integers, and arithmetic operations are done in registers.
- A multiplication of two 32-bit integers yields a 64-bit result that is stored in 2 registers.

A pseudo-code for $digest()$ on such machine may be as follows. For a 'C' implementation, please see [1].

```
$digest(key, msg)$
    $Sum = 0$
    load $key[1]$
    for $i = 1$ to $t$
        load $msg[i]$
```

[1] If we further ignore the effect of the carry in (word) multiplications of both $digest()$ and the scheme of Dietzfelbinger et al. then they become very similar to the Toeplitz matrix based construction of Krawczyk [12, 13] and Mansour et al. [18] discussed in Annex A. Such a carry-less multiplication instruction is available in a new Intel processor [3].

[2] Although this is a 32-bit machine, the same operation count is applicable to a ($2b = 64$)-bit machine. In the latter, a multiplication of two 32-bit unsigned integer is stored in a single 64-bit register, and $High$ and Low are the upper and lower 32-bit halves of the register.

```
        load $key[i+1]$
        $\langle High1, Low1\rangle = msg[i] * key[i]$
        $\langle High2, Low2\rangle = msg[i] * key[i+1]$
        $Sum = Sum + Low1 + High2$
    return $Sum$
```

This consists of $2t = 2M/b$ word multiplications (MULT) and $2t = 2M/b$ addition modulo 2^b (ADD). That is each message-word requires 1 MULT and 2 ADD operations. As in [9], a MULT/ADD operation should include not only the actual arithmetic instruction but also loading the message- and key-words to registers and/or loop handling.

The following theorem shows that the switch from a single (arbitrarily long) multiplication of Dietfelbinger et al. into word multiplications of *digest*() does not weaken the security of the construction. Namely the same collision probability of 2^{1-b} is retained while optimality in distribution is achieved. Moreover this change not only greatly increases computational efficiency but also removes the restriction of odd universal hash key as required in Dietfelbinger et al.

Theorem 1. For any $t, b \geq 1$, *digest*() of Equation 1 satisfies Definition 3 with the distribution probability $\epsilon_d = 2^{-b}$ and the collision probability $\epsilon_c = 2^{1-b}$ on equal length inputs.

Proof. We first consider the collision property. For any pair of distinct messages of equal length: $m = m_1 \cdots m_t$ and $m' = m'_1 \cdots m'_t$, without loss of generality we assume that $m_1 > m'_1$.[3] A digest collision is equivalent to:

$$\sum_{i=1}^{t}[m_i * k_i + (m_i * k_{i+1} \text{ div } 2^b)] = \sum_{i=1}^{t}[m'_i * k_i + (m'_i * k_{i+1} \text{ div } 2^b)] \pmod{2^b}$$

There are two possibilities as follows.

WHEN $m_1 - m'_1$ is odd. The above equality can be rewritten as

$$(m_1 - m'_1)k_1 = y \pmod{2^b} \tag{2}$$

where

$$y = (m'_1 k_2 \text{ div } 2^b) - (m_1 k_2 \text{ div } 2^b) + \sum_{i=2}^{t} \Big[(m'_i - m_i) * k_i + (m'_i * k_{i+1} \text{ div } 2^b) - (m_i * k_{i+1} \text{ div } 2^b)\Big]$$

We note that y depends only on keys $k_2, \ldots, k_{t+1}$, and hence we fix k_2 through k_{t+1} in our analysis. Since $m_1 - m'_1$ is odd, i.e. $m_1 - m'_1$ and 2^b are co-prime, there is at most one value of k_1 satisfying Equation 2. The collision probability is therefore $\epsilon_c = 2^{-b} < 2^{1-b}$.

WHEN $m_1 - m'_1$ is even. A digest collision can be rewritten as

$$(m_1 - m'_1)k_1 + (m_1 k_2 \text{ div } 2^b) - (m'_1 k_2 \text{ div } 2^b) + (m_2 - m'_2)k_2 = y \pmod{2^b} \tag{3}$$

where

$$y = (m'_2 k_3 \text{ div } 2^b) - (m_2 k_3 \text{ div } 2^b) + \sum_{i=3}^{t} \Big[(m'_i - m_i) * k_i + (m'_i * k_{i+1} \text{ div } 2^b) - (m_i * k_{i+1} \text{ div } 2^b)\Big]$$

[3] Please note that when $m_i = m'_i$ for all $i \in \{1, \ldots, j\}$ then in the following calculation we will assume that $m_{j+1} > m'_{j+1}$.

We note that y depends only on keys $k_3,\ldots,k_{t+1}$. If we fix k_3 through k_{t+1} in our analysis, we need to find the number of pairs (k_1, k_2) such that Equation 3 is satisfied. We arrive at

$$\epsilon_c = \text{Prob}_{\left\{\substack{0\le k_1<2^b\\ 0\le k_2<2^b}\right\}} \left[(m_1 - m_1')k_1 + (m_1k_2 \text{ div } 2^b) - (m_1'k_2 \text{ div } 2^b) + (m_2 - m_2')k_2 = y \pmod{2^b}\right]$$

Let us define

$$\begin{aligned} m_1k_2 &= u2^b + v \\ m_1'k_2 &= u'2^b + v' \end{aligned}$$

Since we assumed $m_1 > m_1'$, we have $u \ge u'$ and $(m_1 - m_1')k_2 = (u - u')2^b + v - v'$.

- When $v \ge v'$: $(m_1k_2 \text{ div } 2^b) - (m_1'k_2 \text{ div } 2^b) = (m_1 - m_1')k_2 \text{ div } 2^b$
- When $v < v'$: $(m_1k_2 \text{ div } 2^b) - (m_1'k_2 \text{ div } 2^b) = [(m_1 - m_1')k_2 \text{ div } 2^b] + 1$

Let $c = m_1 - m_1'$ and $d = m_2 - m_2' \pmod{2^b}$, we then have $1 \le c < 2^b$ and:

$$\epsilon_c \le p_1 + p_2$$

where

$$p_1 = \text{Prob}_{\left\{\substack{0\le k_1<2^b\\ 0\le k_2<2^b}\right\}} \left[ck_1 + (ck_2 \text{ div } 2^b) + dk_2 = y \pmod{2^b}\right]$$

and

$$p_2 = \text{Prob}_{\left\{\substack{0\le k_1<2^b\\ 0\le k_2<2^b}\right\}} \left[ck_1 + (ck_2 \text{ div } 2^b) + dk_2 = y - 1 \pmod{2^b}\right]$$

Using Lemma 1, we have $p_1, p_2 \le 2^{-b}$, and thus $\epsilon_c \le 2^{1-b}$.

As regards distribution, since $m = m_1 \cdots m_t > 0$ as specified in Definition 3, without loss of generality we can assume that $m_1 \ge 1$. If we fix k_3 through k_{t+1} and for any $y \in \{0, \ldots, 2^b - 1\}$, then the distribution probability ϵ_d is equivalent to:

$$\epsilon_d = \text{Prob}_{\left\{\substack{0\le k_1<2^b\\ 0\le k_2<2^b}\right\}} \left[m_1k_1 + (m_1k_2 \text{ div } 2^b) + m_2k_2 = y \pmod{2^b}\right]$$

Since $1 \le m_1 < 2^b$, we can use Lemma 1 to deduce that $\epsilon_d = 2^{-b}$. □

Lemma 1. Let $1 \le c < 2^b$ and $0 \le d < 2^b$, then for any $y \in \{0, \ldots, 2^b - 1\}$ we have

$$\text{Prob}_{\left\{\substack{0\le k_1<2^b\\ 0\le k_2<2^b}\right\}} \left[ck_1 + (ck_2 \text{ div } 2^b) + dk_2 = y \pmod{2^b}\right] = 2^{-b}$$

Proof. We write $c = s2^l$ with s odd and $0 \le l < b$. Since s and 2^b are co-prime, there exist a unique inverse modulo 2^b of s, we call it s^{-1}. Our equation now becomes:

$$2^l sk_1 + (2^l sk_2 \text{ div } 2^b) + ds^{-1}sk_2 = y \pmod{2^b}$$

Let $sk_1 = \gamma \pmod{2^{b-l}}$ and $sk_2 = \alpha 2^{b-l} + \beta \pmod{2^b}$, we then have $0 \le \gamma < 2^{b-l}$ and $0 \le \alpha < 2^l$. The above equation becomes:

$$\begin{aligned} 2^l\gamma + \alpha + ds^{-1}(\alpha 2^{b-l} + \beta) &= y \pmod{2^b} \\ 2^l\gamma + \alpha(1 + ds^{-1}2^{b-l}) + \beta ds^{-1} &= y \pmod{2^b} \\ 2^l\gamma + \alpha x &= z \pmod{2^b} \end{aligned}$$

where $x = 1 + ds^{-1}2^{b-l} \pmod{2^b}$ which is always odd because $l < b$, and $z = y - \beta ds^{-1} \pmod{2^b}$. Since z is independent of γ and α, we fix z in our analysis. We can then use Lemma 2 to derive that there is a unique pair (γ, α) satisfying the above equation.

Since $0 \leq \gamma < 2^{b-l}$ and $0 \leq \alpha < 2^l$, γ and α together determine b bits of the combination of k_1 and k_2. Consequently there are at most 2^b different pairs (k_1, k_2) satisfying the condition that we require in this lemma. □

Lemma 2. Let $0 \leq l < b$ and $x \in \{1, 3, \ldots, 2^b - 1\}$ then for any $z \in \{0, \ldots, 2^b - 1\}$ there is a unique pair (γ, α) such that $0 \leq \gamma < 2^{b-l}$, $0 \leq \alpha < 2^l$, and $2^l\gamma + \alpha x = z \pmod{2^b}$.

Proof. If there exist two distinct pairs (γ, α) and (γ', α') satisfying this condition, then

$$2^l\gamma + \alpha x = 2^l\gamma' + \alpha' x = z \pmod{2^b}$$

which implies that

$$2^l(\gamma - \gamma') = (\alpha' - \alpha)x \pmod{2^b}$$

This leads to two possibilities.

- When $\alpha' = \alpha$ then $2^l(\gamma - \gamma') = 0$, which means that $2^{b-l}|(\gamma - \gamma')$. The latter is impossible because $0 \leq \gamma, \gamma' < 2^{b-l}$ and $\gamma \neq \gamma'$.
- When $\alpha' \neq \alpha$ and since x is odd, we must have $2^l|(\alpha' - \alpha)$. This is also impossible because $0 \leq \alpha, \alpha' < 2^l$.

□

REMARKS. The bound given by Theorem 1 for the distribution probability ($\epsilon_d = 2^{-b}$) is tight: let $m = 0^{b-1}1$ and any y and note that any key $k = k_1k_2$ with $k_1 = y$ satisfying this equation $digest(k, m) = y$. The bound given by Theorem 1 for the collision probability $\epsilon_c = 2^{1-b}$ also appears to be tight, i.e. it cannot be reduced to 2^{-b}. To verify this bound, we have implemented exhaustive tests on single-word messages with small value of b. For example, when $b = 7$, we look at all possible pairs of two different ($b = 7$)-bit messages in combination with all ($2b = 14$)-bit keys, the obtained collision probability is $2^{-7} \times 1.875$.

We end this section by pointing out that truncation is secure in this digest construction. For any $b' \in \{1, \ldots, b-1\}$, we define

$$\text{trunc}_{b'}(digest(k, m)) = \sum_{i=1}^{t} [m_i * k_i + (m_i * k_{i+1} \text{ div } 2^b)] \bmod 2^{b'} \tag{4}$$

where $\text{trunc}_{b'}()$ takes the first b' least significant bits of the input. We then have the following theorem whose proof is very similar to the proof of Theorem 1, and hence it is not given here.

Theorem 2. For any $n, t \geq 1$, $b \geq 1$ and any integer $b' \in \{1, \ldots, b-1\}$, $\text{trunc}_{b'}(digest())$ of Equation 4 satisfies Definition 3 with the distribution probability $\epsilon_d = 2^{-b'}$ and the collision probability $\epsilon_c = 2^{1-b'}$ on equal length inputs.

3.3 Extending *digest*()

If we want to use digest functions as the main ingredient of a message authentication code, we need to reduce the collision probability without increasing the word bitlength b that is dictated by architecture characteristics. One possibility is to hash our message with several random and independent keys, and concatenate the results. If we concatenate the results from n independent instances of the digest function, the collision probability drops from 2^{1-b} to 2^{n-nb}. This solution however requires n times as much key material.

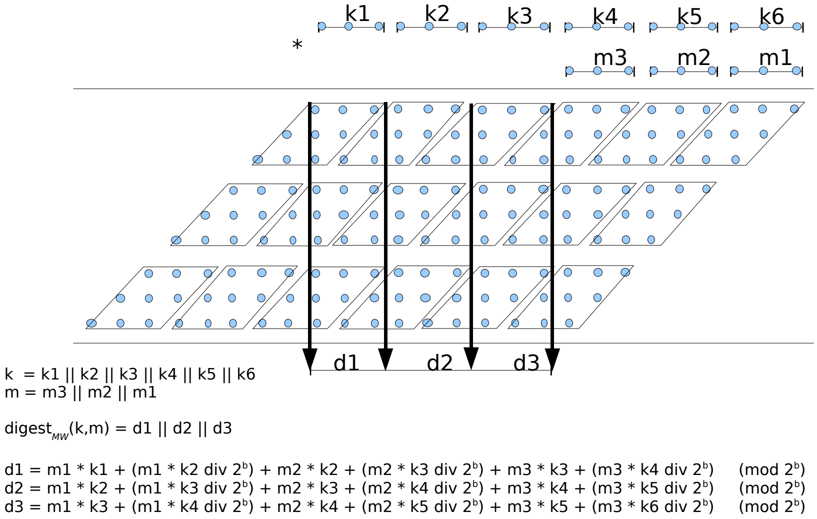

k1
k2
k3
k4
k5
k6
*
m3
m2
m1
d1
d2
d3
k = k1 || k2 || k3 || k4 || k5 || k6
m = m3 || m2 || m1
digest_MW(k,m) = d1 || d2 || d3
d1 = m1 * k1 + (m1 * k2 div 2^b) + m2 * k2 + (m2 * k3 div 2^b) + m3 * k3 + (m3 * k4 div 2^b) (mod 2^b)
d2 = m1 * k2 + (m1 * k3 div 2^b) + m2 * k3 + (m2 * k4 div 2^b) + m3 * k4 + (m3 * k5 div 2^b) (mod 2^b)
d3 = m1 * k3 + (m1 * k4 div 2^b) + m2 * k4 + (m2 * k5 div 2^b) + m3 * k5 + (m3 * k6 div 2^b) (mod 2^b)

Fig. 2. A $3b$-bit (or three-word) output $digest_{MW}(k, m)$: each parallelogram represents the expansion of a word multiplication between a b-bit key block and a b-bit message block.

A much better and well-studied approach is to use the Toeplitz-extension: given one key we left shift the key by one word to get the next key and digest again. The resulting construction is called $digest_{MW}()$, where MW stands for multiple-word output. The structure of $digest_{MW}()$ is again graphically illustrated by an example in Figure 2 that shows a close connection between $digest_{MW}()$ and the multiplicative universal hashing scheme of Dietfelbinger et al.

We define a n-blocks or $(n \times b)$-bit output $digest_{MW}(k, m)$ as follows. We still divide m into b-bit blocks $\langle m_1, \ldots, m_{t=M/b} \rangle$. However, an $(M + bn)$-bit key $k = \langle k_1, \ldots, k_{t+n} \rangle$ will be chosen randomly from $R = \{0, 1\}^{M+bn}$ to compute a nb-bit digest.

For all $i \in \{1, \ldots, n\}$, we then define:

$$d_i = digest(k_{i \cdots t+i}, m) = \sum_{j=1}^{t} [m_j k_{i+j-1} + (m_j k_{i+j} \text{ div } 2^b)] \bmod 2^b$$

And

$$digest_{MW}(k, m) = \langle d_1 \cdots \; d_n \rangle$$

The following theorem and its proof show that $digest_{MW}()$ enjoys the best bound for both collision and distribution probabilities that one could hope for.

Theorem 3. For any $n, t \geq 1$ and $b \geq 1$, $digest_{MW}()$ satisfies Definition 3 with the distribution probability $\epsilon_d = 2^{-nb}$ and the collision probability $\epsilon_c = 2^{n-nb}$ on equal length inputs.

Proof. We first consider the collision property of a digest function. For any pair of distinct messages of equal length: $m = m_1 \cdots m_t$ and $m' = m'_1 \cdots m'_t$, without loss of generality we assume that $m_1 > m'_1$. Please note that when $t = 1$ or $m_i = m'_i$ for all $i \in \{1, \ldots, t-1\}$ then in the following calculation we will assume that $m_{t+1} = m'_{t+1} = 0$.

For $i \in \{1, \ldots, n\}$, we define Equality E_i as

$$E_i : \sum_{j=1}^{t} \left[m_j k_{i+j-1} + (m_j k_{i+j} \text{ div } 2^b) \right] = \sum_{j=1}^{t} \left[m'_j k_{i+j-1} + (m'_j k_{i+j} \text{ div } 2^b) \right] \pmod{2^b}$$

and thus the collision probability is: $\epsilon_c = \text{Prob}_{\{k \in R\}}[E_1 \wedge \cdots \wedge E_n]$.

WHEN $m_1 - m'_1$ is odd. We proceed by proving that for all $i \in \{1, \ldots, n\}$

$$\text{Prob}[E_i \text{ is true} \mid E_{i+1}, \ldots, E_n \text{ are true}] \leq 2^{-b}$$

For Equality E_n, the claim is satisfied due to Theorem 1. We notice that Equalities E_{i+1} through E_n depend only on keys $k_{i+1}, \ldots, k_{n+t}$, whereas Equality E_i depends also on key k_i. Fix k_{i+1} through k_{n+t} such that Equalities E_{i+1} through E_n are satisfied. We prove that there is at most one value of k_i satisfying E_i. To achieve this we let

$$z = (m'_1 k_{i+1} \text{ div } 2^b) - (m_1 k_{i+1} \text{ div } 2^b) + \sum_{j=2}^{t} \Big[(m'_j - m_j)k_{i+j-1} + (m'_j k_{i+j} \text{ div } 2^b) - (m_j k_{i+j} \text{ div } 2^b)\Big]$$

we then rewrite Equality E_i as

$$(m_1 - m'_1)k_i = z \pmod{2^b}$$

Since we assumed $m_1 - m'_1$ is odd, there is at most one value of k_i satisfying this equation.

WHEN $m_1 - m'_1$ is even. We write $m_1 - m'_1 = 2^l s$ with s odd and $0 < l < b$, and $s' = (m'_2 - m_2)s^{-1}$. We further denote $sk_i = x_i 2^{b-l} + y_i$ for $i \in \{1, \ldots, n+t\}$, where $0 \leq x_i < 2^l$ and $0 \leq y_i < 2^{b-l}$.

For $i \in \{1, \ldots, n\}$, if we define $b_i \in \{0, 1\}$ and

$$f(y_i, x_{i+1}) = 2^l y_i + x_{i+1}[(m_2 - m'_2)s^{-1}2^{b-l} + 1] \pmod{2^b}$$

$$\begin{aligned} g(k_{i+2}, \ldots, k_{i+t}) = (m'_2 k_{i+2} \text{ div } 2^b) + \sum_{j=3}^{t} \Big[m'_j k_{i+j-1} + (m'_j k_{i+j} \text{ div } 2^b)\Big] - \\ (m_2 k_{i+2} \text{ div } 2^b) - \sum_{j=3}^{t} \Big[m_j k_{i+j-1} + (m_j k_{i+j} \text{ div } 2^b)\Big] \pmod{2^b} \end{aligned}$$

then, using similar trick as in the proof of Lemma 1, Equality E_i can be rewritten as

$$\begin{aligned} (m_1 - m'_1)k_i + ((m_1 - m'_1)k_{i+1} \text{ div } 2^b) + (m_2 - m'_2)k_{i+1} &= g(k_{i+2}, \ldots, k_{i+t}) - b_i \pmod{2^b} \\ 2^l s k_i + (2^l s k_{i+1} \text{ div } 2^b) + (m_2 - m'_2)s^{-1} s k_{i+1} &= g(k_{i+2}, \ldots, k_{i+t}) - b_i \pmod{2^b} \\ 2^l y_i + x_{i+1} + (m_2 - m'_2)s^{-1}(x_{i+1}2^{b-l} + y_{i+1}) &= g(k_{i+2}, \ldots, k_{i+t}) - b_i \pmod{2^b} \\ 2^l y_i + x_{i+1}[(m_2 - m'_2)s^{-1}2^{b-l} + 1] &= s' y_{i+1} - b_i + g(k_{i+2}, \ldots, k_{i+t}) \pmod{2^b} \\ f(y_i, x_{i+1}) &= s' y_{i+1} - b_i + g(k_{i+2}, \ldots, k_{i+t}) \pmod{2^b} \end{aligned}$$

Putting Equalities E_1 through E_n together, we have

$$\begin{aligned} E_1:&\ f(y_1, x_2) = s' y_2 - b_1 + g(k_3, \ldots, k_{1+t}) \pmod{2^b} \\ E_2:&\ f(y_2, x_3) = s' y_3 - b_2 + g(k_4, \ldots, k_{2+t}) \pmod{2^b} \\ E_3:&\ f(y_3, x_4) = s' y_4 - b_3 + g(k_5, \ldots, k_{3+t}) \pmod{2^b} \\ &\ \vdots \quad \vdots \quad \vdots \\ E_{n-1}:&\ f(y_{n-1}, x_n) = s' y_n - b_{n-1} + g(k_{n+1}, \ldots, k_{n+t-1}) \pmod{2^b} \\ E_n:&\ f(y_n, x_{n+1}) = s' y_{n+1} - b_n + g(k_{n+2}, \ldots, k_{n+t}) \pmod{2^b} \end{aligned}$$

We fix k_{n+2} through k_{t+n}. We note that there are 2^{b-t} values for y_{n+1} and two values for b_n. For each pair (y_{n+1}, b_n) there is a unique pair (y_n, x_{n+1}) satisfying Equality E_n due to Lemma 2. Similarly, for each tuple $\langle y_n, k_{n+1}, b_{n-1}, b_n \rangle$ there is also a unique pair (y_{n-1}, x_n) satisfying Equality E_{n-1}. We will continue this process until we reach the pair (y_1, x_2) in Equality E_1. Since Equalities E_1 through E_n do not depend on x_1 and there are 2^l values for x_1, there will be at most $2^l 2^n 2^{b-l} = 2^{n+b}$ different tuples $\langle k_1 \cdots k_{n+1} \rangle$ satisfying Equalities E_1 through E_n. And thus the collision probability $\epsilon_c = 2^{n+b}/2^{(n+1)b} = 2^{n-nb}$.

Similar argument also leads to our bound on the distribution probability $\epsilon_d = 2^{-nb}$. □

REMARKS. Even though Theorems 1 and 3 address the collision property of an almost universal hash function, their proofs can be easily adapted to show that our constructions are also ϵ_c-almost-Δ-universal [9] as in the case of the MMH scheme considered in the next section. The latter property requires that for every $m, m' \in X$ where $m \neq m'$ and $a \in Y$: $\Pr_{\{k \in R\}}[digest(k, m) - digest(k, m') = a] \leq \epsilon_c$.

Operation count. The advantage of this scheme is the ability to reuse the result of each word multiplication in the computation of two adjacent digest output words as seen in Figure 2 and the following pseudo-code, e.g. the multiplication $m_1 k_2$ is instrumental in the computation of both d_1 and d_2. Using the same machine as specified in subsection 3.2, each message-word therefore requires $(n+1)$ MULT and $2n$ ADD operations.

A pseudo-code for $digest_{MW}()$ on such machine may be as follows

```
$digest_{MW}(key, msg)$
For $i = 1$ to $n$
    $d[i] = 0$
    load $key[i]$
For $j = 1$ to $t$
    load $msg[j]$
    load $key[j+n]$
    $\langle High[0], Low[0] \rangle = msg[j] * key[j]$
    For $i = 1$ to $n$
        $\langle High[i], Low[i] \rangle = msg[j] * key[j+i]$
        $d[i] = d[i] + Low[i-1] + High[i]$
return $\langle d[1] \cdots d[n] \rangle$
```

4 Comparative analysis

In this section, we mainly compare our new digest scheme against well-studied universal hashing algorithms MMH of Halevi and Krawczyk [9] and NH of Black et al. [4] described in Subsections 4.1 and 4.2 respectively. Since *digest*() can be extended to produce multiple-word output as in the case of MMH and NH to build MACs, our analysis consider both single- and multiple-word output schemes. We note that NH is the building block of not only UMAC but also UHASH16 and UHASH32 [4]. For completeness, we will discuss another widely studied UHF family based on polynomial over finite field, e.g. PolyP, PolyQ, PolyR [14] and GHASH [24]. While the polynomial universal hashing schemes only require short keys, they suffer from two unpleasant properties: (1) the collision probability decreases linearly with the message length, and (2) they are less efficient, especially in software implementation, than our digest functions as well as MMH and NH due to the involved modular arithmetic operations.

The properties of the three main schemes – MMH, NH and *digest*() – are summarised in Table 1 where the upper and lower halves correspond to single-word (b bits) and respectively multiple-word (nb bits) output schemes for any $n \geq 1$. This table indicates that the security level obtained in our digest algorithm is higher than both MMH and NH with respect to the same output length. In particular, the collision probability of *digest*() is a third of MMH, while NH must double the output length to achieve the same order of security. For multiple-word output schemes, this advantage in security of our proposed digest algorithm becomes even more significant as seen in the lower half of Table 1.

Scheme	Key length	MULTs/word	ADDs/word	ϵ_c	ϵ_d	Output bitlength
digest	$M + b$	2	2	2^{1-b}	2^{-b}	b
MMH	M	1	1	6×2^{-b}	2^{2-b}	b
NH	M	1/2	3/2	2^{-b}	2^{-b}	$2b$
$digest_{MW}$	$M + nb$	$n + 1$	$2n$	2^{n-nb}	2^{-nb}	nb
MMH_{MW}	$M + (n-1)b$	n	n	$6^n \times 2^{-nb}$	2^{2n-nb}	nb
NH_{MW}	$M + 2(n-1)b$	$n/2$	$3n/2$	2^{-nb}	2^{-nb}	$2nb$

Table 1. A summary on the main properties of *digest*(), MMH and NH. MULT operates on b-bit inputs, whereas ADD operates on inputs of either b or $2b$ bits.

We end this section by providing implementation results in Table 2 of Section 4.3. As described earlier, C files which contain the implementations of NH, MMH and *digest*() as well as their multiple-word output versions can be downloaded from [1] which allows readers to test the speed of the constructions for themselves.

4.1 MMH

Fix a prime number $p \in [2^b, 2^b + 2^{b/2}]$. The b-bit output MMH universal hash function is defined for any $k = k_1, \ldots, k_t$ and $m = m_1, \ldots, m_t$ as follows

$$\text{MMH}(k, m) = \left[\left[\left[\sum_{i=1}^{t} m_i * k_i \right] \bmod 2^{2b} \right] \bmod p \right] \bmod 2^b$$

It was proved in [9] that the collision probability of MMH is $\epsilon_c = 6 \times 2^{-b}$ as opposed to only 2^{1-b} of *digest*(). By using the same proof technique presented in [9], it is also not hard to show that the distribution probability of MMH is $\epsilon_d = 2^{2-b}$, as opposed to 2^{-b} of *digest*().

Following is the pseudo-code of MMH take from [9].

```
MMH($key, msg$)
  $SumHigh = SumLow = 0$
  for $i = 1$ to $t$
    load $msg[i]$
    load $key[i]$
    $\langle ProdHigh, ProdLow \rangle = msg[i] * key[i]$
    $SumLow = SumLow + ProdLow$
    $SumHigh = SumHigh + ProdHigh + carry$
  Reduce $\langle SumHigh, SumLow \rangle$ mod $p$ and then mod $2^b$
```

For single-word output, each message word in MMH requires 1 $(b \times b)$ MULT and 1 ADD modulo 2^{2b}. We note however that this does not include the cost of the final reduction modulo p. For n-word output MMH, using "the Toeplitz matrix approach", the scheme is defined as

$$\mathrm{MMH}_{MW}(k, m) = \mathrm{MMH}(k_{1\cdots t}, m) \parallel \mathrm{MMH}(k_{2\cdots t+1}, m) \parallel \cdots \parallel \mathrm{MMH}(k_{n\cdots t+n-1}, m)$$

MMH_{MW} obtains $\epsilon_c = 6^n 2^{-nb}$ and $\epsilon_d = 2^{2n-nb}$, which are considerably weaker than $digest_{MW}()$ ($\epsilon_c = 2^{n-nb}, \epsilon_d = 2^{-nb}$).

4.2 NH

The $2b$-bit output NH universal hash function is defined for any $k = k_1, \ldots, k_t$ and $m = m_1, \ldots, m_t$, where t is even, as follows

$$\mathrm{NH}(k, m) = \sum_{i=1}^{t/2} (k_{2i-1} + m_{2i-1})(k_{2i} + m_{2i}) \bmod 2^{2b}$$

The downside of NH relative to MMH and our digest method is the level of security obtained, namely with a $2b$-bit output, which is twice the length of both *digest*() and MMH, NH was shown to have the collision probability $\epsilon_c = 2^{-b}$ and the distribution probability $\epsilon_d = 2^{-b}$, which are far from optimality. Its computational cost is however lower than the other twos, i.e. each message-word requires only 1/2 $(b \times b)$ MULT, 1 ADD modulo 2^b, and 1/2 ADD modulo 2^{2b}.

Following is the pseudo-code of NH.

```
NH($key, msg$)
    $SumHigh = SumLow = 0$
    for $i = 1$ to $t/2$
        load $msg[2i-1]$
        load $msg[2i]$
        load $key[2i-1]$
        load $key[2i]$
        $Left = msg[2i-1] + key[2i-1]$
        $Right = msg[2i] + key[2i]$
        $\langle ProdHigh, ProdLow \rangle = Left * Right$
        $SumLow = SumLow + ProdLow$
        $SumHigh = SumHigh + ProdHigh + carry$
    return $\langle SumHigh, SumLow \rangle$
```

For $2n$-word output, also using "the Toeplitz matrix approach", we have $\epsilon_c = 2^{-nb}$ and $\epsilon_d = 2^{-nb}$. Each message-word requires $n/2$ MULT and $3n/2$ ADD operations as seen below.

$$\mathrm{NH}_{MW}(k, m) = \mathrm{NH}(k_{1\cdots t}, m) \parallel \mathrm{NH}\,(k_{3\cdots t+2}, m) \parallel \cdots \parallel \mathrm{NH}(k_{2n-1\cdots t+2(n-1)}, m)$$

4.3 Implementations of MMH, NH and digest constructions

We have tested the implementations of *digest*(), MMH, NH as well as their multiple-word output versions on a workstation with a 1GHz AMD Athlon(tm) 64 X2 Dual Core Processor (4600+ or 512 KB caches) running the 2.6.30 Linux kernel. All source codes were written in C making use of GCC 4.4.1 compiler. The number of cycles elapsed during execution was measured by the *clock*() instruction in the normal way (as in UMAC [29]) in our C implementations [1].

digest			MMH			NH		
Output bitlength	ϵ_c	Speed (cpb)	Output bitlength	ϵ_c	Speed (cpb)	Output bitlength	ϵ_c	Speed (cpb)
32	2×2^{-32}	0.53	32	6×2^{-32}	0.31	64	2^{-32}	0.23
64	$2^2 \times 2^{-64}$	1.05	64	$6^2 \times 2^{-64}$	0.57	128	2^{-64}	0.39
96	$2^3 \times 2^{-96}$	1.54	96	$6^3 \times 2^{-96}$	0.76	192	2^{-96}	0.62
160	$2^5 \times 2^{-160}$	2.13	160	$6^5 \times 2^{-160}$	1.37	320	2^{-160}	1.15
256	$2^8 \times 2^{-256}$	3.44	256	$6^8 \times 2^{-256}$	2.31	512	2^{-256}	1.90

Table 2. Performance (cycles/byte) of *digest*, MMH and NH constructions. In each row, the length of NH is always twice the length of MMH and *digest*.

For comparison, we recompiled publicly available source codes for SHA-256 and SHA-512 [26] whose reported speeds on our workstation are 12.35 cpb and 8.54 cpb respectively.

For application of these primitives in MACs, normally each universal hash key is generated once out of a short seed and reused for a period of time, and hence previously reported speeds for MMH and NH within UMAC in [4, 9] and our results do not include the cost of key generation.

Table 2 shows the results of the experiments, which were averaged over a large number of random and long data inputs of at least 8 kilobytes. The speeds are in cycles/byte or cpb. Our digest constructions, at the cost of higher security, are slightly slower than MMH and NH due to extra multiplication operations, but still considerably faster than standard cryptographic hash functions SHA-256 and SHA-512.

4.4 Polynomial universal hashing schemes

Since our emphasis of this paper is on fast software implementation of universal hash functions, we have so far mainly considered UHF algorithms using simple arithmetic operations available in most ordinary computers. In this section, we will study another well-studied class of UHF based on polynomial over finite fields, including PolyP, PolyQ, PolyR [14] and GHASH within Galois Counter Mode or GCM [24].

For simplicity, we will give a simple version of polynomial universal hashing that is the core of PolyP, PolyQ, PolyR and GHASH. Let the set of all messages be $\{m = \langle m_1, \ldots, m_t\rangle; m_i \in \mathbb{F}_p\}$, here p is the largest prime number less than 2^b and the message length is $M = tb$ bits. For any key $k \in \mathbb{F}_p$, we define:

$$\text{Poly}(k, m) = m_1 + m_2k + m_3k^2 + \cdots + m_tk^{t-1} \pmod{p}$$

Such a scheme does have two nice properties as follows

- The key length of the b-bit output Poly() scheme is fixed at b bits regardless of the message length. In contrast, MMH, NH and *digest*() all require the key length to be greater than or equal to message length.
- Poly() provides collision resistance for both equal and unequal length messages. Suppose that the bit lengths of two different messages m and m' are bt and bt', then the collision probability is $max\{t-1, t'-1\}/p$. On the other hand, MMH, NH and *digest*() only ensure collision resistance for equal length data, but not unequal length messages. The latter is intuitively because unequal length messages in *digest*(), MMH and NH require unequal length keys, which make them incomparable for collision analysis.

Regarding the first property, as mentioned earlier all of the short-output constructions are usually used to build MACs which reuse a single key for a period of time. Consequently long key generation from a short seed that is done once in a while for *digest*(), MMH or NH will not affect their practical uses in message authentication codes. Without taking into account key generation, MMH, NH and digest functions are significantly faster than PolyP32, PolyQ32 and PolyP64 whose peak performance in Pentium II assembly are 3.69, 3.86 and 6.86 cpb as reported by Krovetz and Rogaway [14]. In addition to 1 MULT and 1 ADD, Poly() requires an extra reduction modulo p per each message word as seen in the pseudo-code below.[4]

The main disadvantage of a polynomial universal hashing scheme is that its collision probability depends on the length of messages, which is the opposite of MMH, NH and *digest*(). Namely, the collision probability of the above scheme is $\epsilon = (t-1)2^{-b}$ that is no where near the level of security obtained by our digest function when message is of a significant size. The security downside of polynomial universal hash functions does have a negative impact on their use in manual authentication protocols where short-output but highly secure universal hash functions are required.

Following is the pseudo-code for Poly().

```
Poly($key, msg$)
1.  load $msg[1]$
2.  $Sum = msg[1]$
3.  for $i = 2$ to $t$
4.      load $msg[i]$
5.      $Sum = (Sum + msg[i] * key)$ mod $p$
6.  return $Sum$
```

5 Short-output universal hash functions in manual authentication protocols

In addition to MAC schemes, short-output universal hash functions have found use in manual authentication protocols as explained below.

In the following scheme, parties A and B want to authenticate their public data $m_{A/B}$ to each other without the need for passwords, shared private keys as in MACs, or pre-established security infrastructures such as a PKI. Instead authentication is bootstrapped from human trust and interactions.

The authenticated data $m_{A/B}$ might include public keys, images or videos, and so can be of significant size. Using notation taken from authors' work [20–23] the N-indexed arrow ($\longrightarrow_N$) indicates an unreliable and high-bandwidth (or normal) link where messages can be maliciously altered, whereas the E-indexed arrow ($\longrightarrow_E$) represents an authentic and unspoofable (or empirical) channel. The latter is not a private channel (anyone can overhear it) and it is usually very low-bandwidth since it is implemented by humans, e.g., human conversations or manual data transfers between devices. *hash*() is a cryptographic hash function. Long random keys $k_{A/B}$ are generated by A/B, and k_A is kept secret until after k_B is revealed in Message 2. Operators $\|$ and $\oplus$ denote bitwise concatenation and exclusive-or.

A pairwise manual authentication protocol [2, 15, 17, 20]
1.$A \longrightarrow_N B : m_A, hash(A \parallel k_A)$
2.$B \longrightarrow_N A : m_B, k_B$
3.$A \longrightarrow_N B : k_A$
4.$A \longleftrightarrow_E B : h(k^*, m_A \parallel m_B)$
where $k^* = k_A \oplus k_B$

[4] In line 5 of the pseudo-code of Poly() the operation $Sum + msg[i] * key$ can overflow or be bigger than 2^{2b}, and hence reduction modulo p must be done carefully to obtain the correct result. For example, one might compute $y = msg[i] * key$ mod p first, which is followed by $Sum = (Sum + y)$ mod p.

To ensure both devices agree on the same data $m_A \parallel m_b$, their human owners manually compare the universal hash value in Message 4. As human interactions are expensive, the universal hash function needs to have a short output of $b \in [16, 32]$ bits.

As seen from the above protocol, the universal hash key k^* always varies randomly and uniformly from one to another protocol run. In other words, no value of k^* is used to hash more than one message because $k_{A/B}$ instrumental in the computation of k^* are randomly chosen in each protocol run. This is fundamentally different from MACs which use the same private key to hash multiple messages for a period of time, and hence attacks which rely on the reuse of a single private key in multiple sessions are irrelevant in manual authentication protocols. What we then want to understand is the collision and distribution properties of the universal hash function. We stress that this analysis is also applicable to group manual authentication protocols [16, 20–22, 30].

Should *digest*(), MMH or NH (or UHASH16/32) be used directly in Message 4 of the above protocol, random and fresh keys $k_{A/B}$ of similar size as $m_A \parallel m_B$ must be generated whenever the protocol is run.[5] Obviously one can generate a long random key stream from a short seed via a pseudo-random number generator, but it can be computationally expensive especially when the authenticated data $m_{A/B}$ are of a significant size. Of course we can use one of the polynomial universal hashing functions (e.g. PolyP32, PolyQ32 or PolyR16_32 all defined in [14]) which require a short key. But since humans only can compare short value over the empirical channel, it is intolerable that the security bound of the universal hash function degrades linearly along with the length of data being authenticated.

One possibility suggested in [2, 8, 25] is to truncate the output of a cryptographic hash function to the b least significant bits:

$$h(k, m) = \text{trunc}_b\left(hash(k \parallel m)\right)$$

Although it can be computationally infeasible to search for a full cryptographic hash collision, it is not clear whether the truncated solution is sufficiently secure because the definition of a hash function does not normally specify the distribution of individual groups of bits.

What we therefore propose is a combination of cryptographic hashing and short-output universal hash functions. We want to stress that among MMH, NH and *digest*(), the least preferable scheme would be NH because it needs to double output length to achieve the same order of security as MMH and *digest*(). The length of universal hash functions must be short in manual authentication protocols because humans can only compare short strings efficiently and accurately.

Without loss of generality, we use our digest method in the following construction which is also applicable to MMH and NH. Let *hash*() be a B-bit cryptotgraphic hash function, e.g. SHA-2 or SHA-3. First the input key is split into two parts of unequal lengths $k = k_1 \parallel k_2$, where k_1 is $B + b$ bits and k_2 is at least 80 bits. Then our modified digest construction *digest*′() which takes an arbitrarily length message m is computed as follows

$$digest'(k, m) = digest(k_1, hash(m \parallel k_2))$$

We hash the concatenation of m and k_2 to make it much harder for the intruder to search for hash collision because a large number of bits of the hash input will not be controlled by the intruder. Consequently the intruder cannot carry out effective off-line searching.

We denotes θ_c the hash collision probability on random messages of *hash*(), and it should be clear that $\theta_c \gg 2^{-b}$ given that $b \in [16, 32]$. The following theorem will demonstrate that this

[5] Suppose that the bitlengths of input data and output are M and b then *digest*() requires $M + b$ bits and both MMH and NH requires M bits for the key.

construction preserves both the collision probability except a tiny bias due to the hash function and the distribution probability of $digest()$ regardless of what $hash()$ is. It also removes the restriction on equal length input messages because the hash function $hash()$ always produces a fixed length value.[6]

Theorem 4. $digest'()$ satisfies Definition 3 with the distribution probability $\epsilon_d = 2^{-b}$ and the collision probability $\epsilon_c = 2^{1-b} + \theta_c$.

Proof. Let l_1 and l_2 denote the bitlengths of keys k_1 and k_2 respectively.

We first consider collision property of $digest'()$. For any pair of distinct messages m and m', as key k_2 varies uniformly and randomly the probability that $hash(m \parallel k_2) = hash(m' \parallel k_2)$ is bounded above by θ_c. So there are two possibilities:

- When $hash(m \parallel k_2) = hash(m' \parallel k_2)$ then $digest(k_1, hash(m \parallel k_2)) = digest(k_1, hash(m' \parallel k_2))$ for any key $k_1 \in \{0,1\}^{l_1}$.
- When $hash(m \parallel k_2) \neq hash(m' \parallel k_2)$ then $digest(k_1, hash(m \parallel k_2)) = digest(k_1, hash(m' \parallel k_2))$ with probability 2^{1-b}.

Consequently the collision probability of $digest'()$ is

$$\theta_c + (1 - \theta_c)2^{1-b} < \theta_c + 2^{1-b}$$

As regards distribution probability of $digest'()$, we fix message m of arbitrarily length and a b-bit value y in our analysis.

For each value of k_2, there will be at most 2^{l_1-b} different keys k_1 such that

$$digest(k_1, hash(m \parallel k_2)) = y$$

Since there are 2^{l_2} different keys k_2, there will be at most $2^{l_1-b}2^{l_2} = 2^{l_1+l_2-b}$ different pairs (k_1, k_2) or different keys k such that $digest(k_1, hash(m \parallel k_2)) = y$. The distribution probability of $digest'()$ is therefore 2^{-b} □

We end this section by pointing out that the shortness of UHF output required in manual authentication protocols further implies that UHFs with optimal (or nearly optimal) collision probability are much more sought here than in message authentication codes. Although our proposed $digest'()$ scheme is very near to optimality, we might want to go further. To our knowledge, this is possible but at the expense of involving arithmetic that computers less like to do than word multiplication and addition even when the input data is short. These are bit-wise matrix multiplications in the well-studied Toeplitz matrix hashing construction of [12, 18] that we mentioned in Footnote 1 and finite fields modular reductions in polynomial universal hashing schemes of [5, 11, 28]. Both of these are discussed in Annexes A and B.

Acknowledgements:

Long Nguyen's work on this paper was supported by a research grant from the US Office of Naval Research. Andrew William Roscoe's was partially supported by funding from the US Office of Naval Research.

[6] We note that there is a subtle difference between $digest'()$ and PolyR16_32() of [14] which is defined as follows PolyR16_32(k, m) = PolyQ32$(k_1,$ PolyQ16$(k_2, m_2) \parallel m_1)$. Here PolyQ32 and PolyQ16 produce 32- and 16-bit outputs respectively. The idea here is to hash short messages directly with PolyQ16, but hash significant longer messages with a hybrid scheme. As a result PolyR16_32 is faster than PolyQ32, but its collision probability is still dependent on message length. This is not the case with $digest'()$, which partly explains why we need to a rather longer key of $K = B + b + 80$ bits to reach $\epsilon_c = 2^{1-b}$ regardless of the message length.

The authors would like to thank Dr. Andrew Ker at the Oxford University Department of Computer Science for his help with statistical analysis of the digest constructions.

Progresses on the security proof of our digest functions were first made when Long Nguyen visited Professor Bart Preneel and Dr. Frederik Vercauteren at the Computer Security and Industrial Cryptography (COSIC) research group at the Katholieke Universiteit Leuven in September and October 2010. The authors would like to thank them for their time and support as well as Drs Nicky Mouha and Antoon Bosselaers at COSIC for pointing out the relevance of the multiplicative universal hashing scheme of Dietzfelbinger et al.[7] and the literature on hash function implementation and speed measurement for benchmarking.

We also received helpful comments from many anonymous referees as well as had fruitful discussions and technical feedbacks from Professor Serge Vaudenay and Dr. Atefeh Mashatan when Long Nguyen visited the Security and Cryptography Laboratory (LASEC) at the Swiss Federal Institute of Technologies (EPFL) in February and March 2011. The feedbacks significantly improve the technical quality and presentation of the paper. Long Nguyen would like to thank them for their time and support.

References

1. `http://www.cs.ox.ac.uk/publications/publication5935-abstract.html`
2. *Simple Pairing White Paper.* See: `www.bluetooth.com/NR/rdonlyres/0A0B3F36-D15F-4470-85A6-F2CCFA26F70F/0/SimplePairing_WP_V10r00.pdf`
3. `http://software.intel.com/en-us/articles/carry-less-multiplication-and-its-usage-for-computing-the-gcm-mode/`
4. J. Black, S. Halevi, H. Krawczyk, T. Krovetz, P. Rogaway. *UMAC: Fast and Secure Message Authentication.* CRYPTO, LNCS vol. 1666, pp. 216-233, 1999.
5. B. den Boer. *A simple and key-economical unconditional authentication scheme.* Journal of Computer Security **2** (1993), 65-71.
6. J.L. Carter and M.N. Wegman. *Universal Classes of Hash Functions.* Journal of Computer and System Sciences, **18** (1979), 143-154.
7. M. Dietzfelbinger, T. Hagerup, J. Katajainen, and M. Penttonen. *A reliable randomized algorithm for the closest-pair problem.* Journal Algorithms, **25** (1997), 19-51.
8. C. Gehrmann, C. Mitchell and K. Nyberg. *Manual Authentication for Wireless Devices.* RSA Cryptobytes, vol. 7, no. 1, pp. 29-37, 2004.
9. S. Halevi and H. Krawczyk. *MMH: Software Message Authentication in the Gbit/second Rates.* FSE, LNCS vol. 1267, pp. 172-189, 1997.
10. ISO/IEC 9798-6, L.H. Nguyen, ed., 2010, *Information Technology – Security Techniques – Entity authentication – Part 6: Mechanisms using manual data transfer.*
11. T. Johansson, G.A. Kabatianskii, and B. Smeets. *On the relation between A-Codes and Codes correcting independent errors.* Advances in Cryptology, EUROCRYPT 1993, LNCS vol. 765, 1-11.
12. H. Krawczyk. *LFSR-based Hashing and Authentication.* Advances in Cryptology, CRYPTO 1994, LNCS vol. 839, 129-139.
13. H. Krawczyk. *New Hash Functions For Message Authentication.* Advances in Cryptology - Eurocrypt 1995, LNCS vol. 921, pp. 301-310.
14. T. Krovetz and P. Rogaway. em Fast Universal Hashing with Small Keys and no Preprocessing: the PolyR Construction. Information Security and Cryptology - ICICS 2000, LNCS vol. 2015, pp. 73-89, Springer, 2000.
15. S. Laur and K. Nyberg. *Efficient Mutual Data Authentication Using Manually Authenticated Strings.* LNCS vol. 4301, pp. 90-107, 2006.
16. S. Laur and S. Pasini. *SAS-Based Group Authentication and Key Agreement Protocols.* In Public Key Cryptography - PKC 2008, 11th International Workshop on Practice and Theory in Public-Key Cryptography, pp. 197-213.
17. A.Y. Lindell, Comparison-based key exchange and the security of the numeric comparison mode in Bluetooth v2.1, in: *Proceedings of the Cryptographers' Track at the RSA Conference 2009 on Topics in Cryptology*, Lecture Notes in Computer Science, Vol. 5473, M. Fischlin, ed., Springer, 2009, pp. 66-83.
18. Y. Mansour, N. Nisan and P. Tiwari. *The Computational Complexity of Universal Hashing.* Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, pp. 235-243, 1990.
19. A. Mashatan and D. Stinson. *Practical Unconditionally Secure Two-channel Message Authentication.* Designs, Codes and Cryptography **55** (2010), 169-188.

20. L.H. Nguyen and A.W. Roscoe. *Authentication protocols based on low-bandwidth unspoofable channels: A comparative survey.* Journal of Computer Security **19**(1): 139-201 (2011).
21. L.H. Nguyen and A.W. Roscoe. *Efficient group authentication protocol based on human interaction* FCS-ARSPA 2006, pp. 9-31.
22. L.H. Nguyen and A.W. Roscoe. *Authenticating ad-hoc networks by comparison of short digests.* Information and Computation **206**(2-4) (2008), 250-271.
23. L.H. Nguyen and A.W. Roscoe. *Separating two roles of hashing in one-way message authentication.* FCS-ARSPA-WITS 2008, pp. 195-210.
24. National Institute of Standards and Technology. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC.* NIST Special Publication 800-38D, November, 2007.
25. S. Pasini and S. Vaudenay. *SAS-based Authenticated Key Agreement.* Public Key Cryptography - PKC 2006: The 9th international workshop on theory and practice in public key cryptography, LNCS vol. 3958, pp. 395-409.
26. Please see: `http://www.aarongifford.com/computers/sha.html`
27. D.R. Stinson. *Universal Hashing and Authentication Codes.* Advances in Cryptology - Crypto 1991, LNCS vol. 576, pp. 74-85, 1992.
28. R. Taylor. *An Integrity Check Value Algorithm for Stream Ciphers.* Advances in Cryptology, CRYPTO 1993. LNCS vol. 773, Springer-Verlag, pp. 40-48, 1994.
29. The source code and performance of UMAC can be found on this website: `http://fastcrypto.org/umac/`
30. J. Valkonen, N. Asokan and K. Nyberg. *Ad Hoc Security Associations for Groups.* In Proceedings of the Third European Workshop on Security and Privacy in Ad hoc and Sensor Networks 2006. LNCS vol. 4357, pp. 150-164.
31. M.N. Wegman and J.L. Carter. *New Hash Functions and Their Use in Authentication and Set Equality.* Journal of Computer and System Sciences, **22** (1981), 265-279.

A Toeplitz universal hashing

We first give the definition of a Toeplitz matrix.

Definition 4. A Toeplitz matrix A is a (not necessary square) matrix where each left-to-right diagonal is fixed, i.e. for all pairs of indexes (i, j): $A_{i,j} = A_{i+1,j+1}$.

If we want to compute a b-bit universal hash of a M-bit message m, then $(M + b - 1)$-bit key k is drawn randomly from $R = \{0, 1\}^{M+b-1}$. We can generate a Toeplitz matrix $A(k)$ of M rows and b columns from key k, i.e. we assume a linear map from $(\mathbb{F}_2)^{M+b-1}$ to the set of Toeplitz matrices in $(\mathbb{F}_2)^{M\times b}$.

Krawczyk [12] and Mansour [18] independently introduce the following scheme, where the symbol '×' in Equation 5 represents a product of vector m and matrix $A(k)$ over $\mathbb{F}_2$.

$$h^T(k, m) = m \times A(k) \tag{5}$$

If key k is drawn randomly from R, then the collision probability is 2^{-b} which is optimal. For use in manual authentication protocols of Section 5, we define $h(k, m) = h^T(k_1, hash(m \parallel k_2))$ where $k = k_1 \parallel k_2$. This obtains $\epsilon_c = 2^{-b} + \theta_c$ where θ_c is the hash collision probability of $hash()$.

B Polynomial universal hashing

We first define the following n-bit output polynomial universal hashing scheme $\mathrm{PH}_{n,p}$ adapted from [5, 11, 28], where p is the largest prime number less than 2^n. This unversal hash function takes a n-bit key $k \in \mathbb{F}_p$ and a $2n$-bit data $m = m_1 \parallel m_2$, and produces an output in $\mathbb{F}_p$.

$$\mathrm{PH}_{n,p}(k, m) = k * m_1 + m_2 \pmod p$$

It is not difficult to show that the collision probability of this construction is $1/p$.

Suppose that we can hash an arbitrarily long message m into a $4b$-bit value by using a cryptographic hash function then our construction uses two different instances of the above polynomial hashing scheme, namely PH_{b,p_1} and PH_{2b,p_2} where p_1 and p_2 are the biggest prime numbers less than 2^b and 2^{2b} respectively.

$$h(k,m) = \mathrm{PH}_{b,p_1}(k_1, \mathrm{PH}_{2b,p_2}(k_2, hash(m \parallel k_3)))$$

Here $k = k_1 \parallel k_2 \parallel k_3$, where $k_1 \in \mathbb{F}_{p_1}$, $k_2 \in \mathbb{F}_{p_2}$ and k_3 is at least 80 bits.

The collision probability of this construction is therefore $\epsilon_c = 1/p_1 + 1/p_2 + \theta_c$, where θ_c denotes the hash collision probability on random messages of $hash()$. Since $p_2 \gg p_1$ and $1/p_1 \gg \theta_c$, we can deduce that $\epsilon_c \approx 1/p_1 \approx 2^{-b}$.

Lapin: An Efficient Authentication Protocol Based on Ring-LPN

Stefan Heyse[1], Eike Kiltz[1], Vadim Lyubashesvky[2],
Christof Paar[1], and Krzysztof Pietrzak[3]*

[1] Ruhr-Universität Bochum
[2] INRIA / ENS, Paris
[3] IST Austria

Abstract. We propose a new authentication protocol that is provably secure based on a *ring* variant of the learning parity with noise (LPN) problem. The protocol follows the design principle of the LPN-based protocol from Eurocrypt'11 (Kiltz et al.), and like it, is a two round protocol secure against *active* attacks. Moreover, our protocol has small communication complexity and a very small footprint which makes it applicable in scenarios that involve low-cost, resource-constrained devices.
Performance-wise, our protocol is more efficient than previous LPN-based schemes, such as the many variants of the Hopper-Blum (HB) protocol and the aforementioned protocol from Eurocrypt'11. Our implementation results show that it is even comparable to the standard challenge-and-response protocols based on the AES block-cipher. Our basic protocol is roughly 20 times slower than AES, but with the advantage of having 10 times smaller code size. Furthermore, if a few hundred bytes of non-volatile memory are available to allow the storage of some off-line pre-computations, then the online phase of our protocols is only twice as slow as AES.

Keywords: HB protocols, RFID authentication, LPN problem, Ring-LPN problem

1 Introduction

Lightweight shared-key authentication protocols, in which a tag authenticates itself to a reader, are extensively used in resource-constrained devices such as radio-frequency identification (RFID) tags or smart cards. The straight-forward approach for constructing secure authentications schemes is to use low-level symmetric primitives such as block-ciphers, e.g. AES [DR02]. In their most basic form, the protocols consist of the reader sending a short challenge c and the tag responding with $\mathsf{AES}_K(c)$, where K is the shared secret key. The protocol is secure if AES fulfils a strong, *interactive* security assumption, namely that it behaves like a strong pseudo-random function.

Authentication schemes based on AES have some very appealing features: they are extremely fast, consist of only 2 rounds, and have very small communication complexities. In certain scenarios, however, such as when low-cost and resource-constrained devices are involved, the relatively large gate-count and code size used to implement AES may pose a problem. One approach to overcome the restrictions presented by low-weight devices is to construct a low-weight block cipher (e.g. PRESENT [BKL$^+$07]), while another approach has been to deviate entirely from block-cipher based constructions and build a *provably-secure* authentication scheme based on the hardness of some mathematical problem. In this work, we concentrate on this second approach.

Ideally, one would like to construct a scheme that incorporates all the beneficial properties of AES-type protocols, while also acquiring the additional provable security and smaller code description characteristics. In the past decade, there have been proposals that achieved some, but not all, of these criteria. Most of these proposals are extensions and variants of the Hopper-Blum (HB) protocol, recently a protocol following a different blueprint has been proposed by Kiltz et al. [KPC$^+$11]. Our proposal can be seen as a continuation of this line of research that contains all the advantages enjoyed by LPN-based protocols, while at the same time, getting even closer to enjoying the benefits of AES-type schemes.

OVERVIEW OF OUR RESULTS. In this work we present a new symmetric authentication protocol which (i) is provably-secure against active attacks (as defined in [JW05]) based on the Ring-LPN assumption,

* Supported by the European Research Council / ERC Starting Grant (259668-PSPC)

a natural variant of the standard LPN (learning parity with noise) assumption; (ii) consists of 2 rounds; (iii) has small communication complexity (approximately 1300 bits); (iv) has efficiency comparable to AES-based challenge-response protocols (depending on the scenario), but with a much smaller code size. To demonstrate the latter we implemented the tag part of our new protocol in a setting of high practical relevance – a low-cost 8-bit microcontroller which is a typical representative of a CPU to be found on lightweight authentication tokens, and compared its performance (code size and running time) with an AES implementation on the same platform.

PREVIOUS WORKS. Hopper and Blum [HB00,HB01] proposed a 2-round authentication protocol that is secure against *passive* adversaries based on the hardness of the LPN problem (we remind the reader of the definition of the LPN problem in Section 1.2). The characteristic feature of this protocol is that it requires very little workload on the part of the tag and the reader. Indeed, both parties only need to compute vector inner products and additions over F_2, which makes this protocol (thereafter named HB) a good candidate for lightweight applications.

Following this initial work, Juels and Weis constructed a protocol called HB^+ [JW05] which they proved to be secure against more realistic, so called *active* attacks. Subsequently, Katz et al. [KS06a], [KS06b,KSS10] provided a simpler security proof for HB^+ as well as showed that it remains secure when executed in parallel. Unlike the HB protocol, however, HB^+ requires three rounds of communication between tag and reader. From a practical aspect, 2 round authentication protocols are often advantageous over 3 round protocols. They often show a lower latency which is especially pronounced on platforms where the establishment of a communication in every directions is accompanied by a fixed initial delay. An additional drawback of both HB and HB^+ is that their communication complexity is on the order of hundreds of thousands of bits, which makes them almost entirely impractical for lightweight authentication tokens because of timing and energy constraints. (The contactless transmission of data on RFIDs or smart cards typically requires considerably more energy than the processing of the same data.)

To remedy the overwhelming communication requirement of HB^+, Gilbert et al. proposed the three-round $\mathsf{HB}^\sharp$ protocol [GRS08a]. A particularly practical instantiation of this protocol requires fewer than two thousand bits of communication, but is no longer based on the hardness of the LPN problem. Rather than using independent randomness, the $\mathsf{HB}^\sharp$ protocol utilized a Toeplitz matrix, and is thus based on a plausible assumption that the LPN problem is still hard in this particular scenario.

A feature that the $\mathsf{HB}, \mathsf{HB}^+$, and $\mathsf{HB}^\sharp$ protocols have in common is that at some point the reader sends a random string r to the tag, which then must reply with $\langle r, s\rangle + e$, the inner product of r with the secret s plus some small noise e. The recent work of Kiltz et al. [KPC$^+$11] broke with this approach, and they were able to construct the first 2-round LPN-based authentication protocol (thereafter named EC11) that is secure against active attacks. In their challenge-response protocol, the reader sends some challenge bit-string c to the tag, who then answers with a noisy inner product of a random r (which the tag chooses itself) and a session-key $K(c)$, where $K(c)$ selects (depending on c) half of the bits from the secret s. Unfortunately, the EC11 protocol still inherits the large communication requirement of HB and HB^+. Furthermore, since the session key $K(c)$ is computed using bit operations, it does not seem to be possible to securely instantiate EC11 over structured (and hence more compact) objects such as Toeplitz matrices (as used in $\mathsf{HB}^\sharp$ [GRS08a]).

1.1 Our contributions

PROTOCOL. In this paper we propose a variant of the EC11 protocol from [KPC$^+$11] which uses an "algebraic" derivation of the session key $K(c)$, thereby allowing to be instantiated over a carefully chosen ring $\mathsf{R} = \mathsf{F}_2[X]/(f)$. Our scheme is no longer based on the hardness of LPN, but rather on the hardness of a natural generalization of the problem to rings, which we call Ring-LPN(see Section 3 for the definition of the problem.) The general overview of our protocol is quite simple. Given a challenge c from the reader, the tag answers with $(r, z = r \cdot K(c) + e) \in \mathsf{R} \times \mathsf{R}$, where r is a random ring element, e is a

Table 1. Summary of implementation results

Protocol	Time (cycles)		Code size
	online	offline	(bytes)
Ours: reducible f (§5.1)	$30,000$	$82,500$	$1,356$
Ours: irreducible f (§5.2)	$21,000$	$174,000$	459
AES-based [LLS09,Tik]	$10,121$	0	$4,644$

low-weight ring element, and $K(c) = sc + s'$ is the session key that depends on the shared secret key $K = (s, s') \in \mathsf{R}^2$ and the challenge c. The reader accepts if $e' = r \cdot K(c) - z$ is a polynomial of low weight, cf. Figure 1 in Section 4. Compared to the HB and HB^+ protocols, ours has one less round and a dramatically lower communication complexity. Our protocol has essentially the same communication complexity as $\mathsf{HB}^\sharp$, but still retains the advantage of one fewer round. And compared to the two-round EC11 protocol, ours again has the large savings in the communication complexity. Furthermore, it inherits from EC11 the simple and tight security proof that, unlike three-round protocols, does not use rewinding.

We remark that while our protocol is provably secure against active attacks, we do not have a proof of security against man-in-the-middle ones. Still, as argued in [KSS10], security against active attacks is sufficient for many use scenarios (see also [JW05,KW05,KW06]). We would like to mention that despite man-in-the-middle attacks being outside our "security model", we think that it is still worthwhile investigating whether such attacks do in fact exist, because it presently seems that all previous man-in-the middle attacks against HB-type schemes along the lines of Gilbert et al. [GRS05] and of Ouafi et al. [OOV08] do not apply to our scheme. In Appendix A, however, we do present a man-in-the-middle attack that works in time approximately $n^{1.5} \cdot 2^{\lambda/2}$ (where n is the dimension of the secret and λ is the security parameter) when the adversary can influence on the order of $n^{1.5} \cdot 2^{\lambda/2}$ interactions between the reader and the tag. To resist this attack, one could simply double the security parameter, but we believe that even for $\lambda = 80$ (and $n > 512$, as it is currently set in our scheme) this attack is already impractical because of the extremely large number of interactions that the adversary will have to observe and modify.

IMPLEMENTATION. We demonstrate that our protocol is indeed practical by providing a lightweight implementation of the tag part of the protocol. (The reader is typically not run on a constrained device and therefore we do not consider its performance.) The target platform was an AVR ATmega163 [Atm] based smart card. The ATmega163 is a small 8-bit microcontroller which is a typical representative of a CPU to be found on lightweight authentication tokens. The main metrics we consider are run time and code size. We compare our results with a challenge-response protocol using an AES implementation optimized for the target platform. A major advantage of our protocol is its very small code size. The most compact implementation requires only about 460 bytes of code, which is an improvement by factor of about 10 over AES-based authentication. Given that EEPROM or FLASH memory is often one of the most precious resources on constrained devices, our protocol can be attractive in certain situations. The drawback of our protocol over AES on the target platform is an increase in clock cycles for one round of authentication. However, if we have access to a few hundred bytes of non-volatile data memory, our protocol allows precomputations which make the on-line phase only a factor two or three slower than AES. But even without precomputations, the protocol can still be executed in a few 100 msec, which will be sufficient for many real-world applications, e.g. remote keyless entry systems or authentication for financial transactions. Table 1 gives a summary of the results, see Section 5 for details.

We would like to stress at this point that our protocol is targeting lightweight tags that are equipped with (small) CPUs. For ultra constrained tokens (such as RFIDs in the price range of a few cents targeting the EPC market) which consist nowadays of a small integrated circuit, even compact AES implementations are often considered too costly. (We note that virtually all current commercially available low-end RFIDs do not have any crypto implemented.) However, tokens which use small microcontrollers are far more common, e.g., low-cost smart cards, and they do often require strong authentication. Also, it can be speculated that computational RFIDs such as the WISP [Wik] will become more common in the future, and hence software-friendly authentication methods that are highly efficient such as the protocol provided here will be needed.

1.2 LPN, Ring-LPN, and Related Problems

The security of our protocols relies on the new Ring Learning Parity with Noise (Ring-LPN) problem which is a natural extension of the standard Learning Parity with Noise (LPN) problem to rings. It can also be seen as a particular instantiation of the Ring-LWE (Learning with Errors over Rings) problem that was recently shown to have a strong connection to lattices [LPR10]. We will now briefly describe and compare these hardness assumptions, and we direct the reader to Section 3 for a formal definition of the Ring-LPN problem.

The decision versions of these problems require us to distinguish between two possible oracles to which we have black-box access. The first oracle has a randomly generated secret vector $s \in \mathsf{F}_2^n$ which it uses to produce its responses. In the LPN problem, each query to the oracle produces a uniformly random matrix[4] $A \in \mathsf{F}_2^{n\times n}$ and a vector $As + e = t \in \mathsf{F}_2^n$ where e is a vector in F_2^n each of whose entries is an independently generated Bernoulli random variable with probability of 1 being some public parameter τ between 0 and $1/2$. The second oracle in the LPN problem outputs a uniformly-random matrix $A \in \mathsf{F}_2^{n\times n}$ and a uniformly random vector $t \in \mathsf{F}_2^n$.

The only difference between LPN and Ring-LPN is in the way the matrix A is generated (both by the first and second oracle). While in the LPN problem, all its entries are uniform and independent, in the Ring-LPN problem, only its first column is generated uniformly at random in F_2^n. The remaining n columns of A depend on the first column and the underlying ring $\mathsf{R} = \mathsf{F}_2[X]/(f(X))$. If we view the first column of A as a polynomial $r \in \mathsf{R}$, then the i^{th} column (for $0 \leq i \leq n-1$) of A is just the vector representation of rX^i in the ring R. Thus when the oracle returns $As + e$, this corresponds to it returning the polynomial $r \cdot s + e$ where the multiplication of polynomials r and s (and the addition of e) is done in the ring R. The $\mathsf{Ring\text{-}LPN}^{\mathsf{R}}$ assumption states that it is hard to distinguish between the outputs of the first and the second oracle described above. In Section 3, we discuss how the choice of the ring R affects the security of the problem.

While the standard Learning Parity with Noise (LPN) problem has found extensive use as a cryptographic hardness assumption (e.g., [HB01,JW05,GRS08b,GRS08a,ACPS09,KSS10]), we are not aware of any constructions that employed the Ring-LPN problem. There have been some previous works that considered some relatively similar “structured” versions of LPN. The $\mathsf{HB}^\sharp$ authentication protocol of Gilbert et al. [GRS08a] made the assumption that for a random Toeplitz matrix $S \in \mathsf{F}_2^{m\times n}$, a uniformly random vector $a \in \mathsf{F}_2^n$, and a vector $e \in \mathsf{F}_2^m$ whose coefficients are distributed as Ber_τ, the output $(a, Sa + e)$ is computationally indistinguishable from (a, t) where t is uniform over F_2^m.

Another related work, as mentioned above, is the recent result of Lyubashevsky et al. [LPR10], where it is shown that solving the decisional Ring-LWE (Learning with Errors over Rings) problem is as hard as quantumly solving the worst case instances of the shortest vector problem in *ideal* lattices. The Ring-LWE problem is quite similar to Ring-LPN, with the main difference being that the ring R is defined as $\mathsf{F}_q[X]/(f(X))$ where $f(X)$ is a cyclotomic polynomial and q is a prime such that $f(X)$ splits completely into $deg(f(X))$ distinct factors over F_q.

Unfortunately, the security proof of our authentication scheme does not allow us to use a polynomial $f(X)$ that splits into low-degree factors, and so we cannot base our scheme on lattice problems. For a similar reason (see the proof of our scheme in Section 4 for more details), we cannot use samples that come from a Toeplitz matrix as in [GRS08a]. Nevertheless, we believe that the Ring-LPN assumption is very natural and will find further cryptographic applications, especially for constructions of schemes for low-cost devices.

[4] In the more common description of the LPN problem, each query to the oracle produces one random sample in F_2^n. For comparing LPN to Ring-LPN, however, it is helpful to consider the oracle as returning a matrix of n random independent samples on each query.

2 Definitions

2.1 Rings and Polynomials

For a polynomial $f(X)$ over F_2, we will often omit the indeterminate X and simply write f. The degree of f is denoted by $deg(f)$. For two polynomials a, f in $\mathsf{F}_2[X]$, $a \bmod f$ is defined to be the unique polynomial r of degree less than $deg(f)$ such that $a = fg + r$ for some polynomial $g \in \mathsf{F}_2[X]$. The elements of the ring $\mathsf{F}_2[X]/(f)$ will be represented by polynomials in $\mathsf{F}_2[X]$ of maximum degree $deg(f) - 1$. In this paper, we will only be considering rings $\mathsf{R} = \mathsf{F}_2[X]/(f)$ where the polynomial f factors into *distinct* irreducible factors over F_2. For an element a in the ring $\mathsf{F}_2[X]/(f)$, we will denote by $\widehat{a}$, the CRT (Chinese Remainder Theorem) representation of a with respect to the factors of f. In other words, if $f = f_1 \dots f_m$ where all f_i are irreducible, then

$$\widehat{a} \doteq (a \bmod f_1, \dots, a \bmod f_m).$$

If f is itself an irreducible polynomial, then $\widehat{a} = a$. Note that an element $\widehat{a} \in \mathsf{R}$ has a multiplicative inverse iff, for all $1 \leq i \leq m$, $a \neq 0 \bmod f_i$. We denote by R^* the set of elements in R that have a multiplicative inverse.

2.2 Distributions

For a distribution D over some domain, we write $r \xleftarrow{\$} D$ to denote that r is chosen according to the distribution D. For a domain Y, we write $U(Y)$ to denote the uniform distribution over Y. Let Ber_τ be the Bernoulli distribution over F_2 with parameter (bias) $\tau \in \,]0, 1/2[$ (i.e., $\Pr[x = 1] = \tau$ if $x \leftarrow \mathsf{Ber}_\tau$). For a polynomial ring $\mathsf{R} = \mathsf{F}_2[X]/(f)$, the distribution $\mathsf{Ber}_\tau^{\mathsf{R}}$ denotes the distribution over the polynomials of R, where each of the coefficients of the polynomial is drawn independently from Ber_τ. For a ring R and a polynomial $s \in \mathsf{R}$, we write $\Lambda_\tau^{\mathsf{R},s}$ to be the distribution over $\mathsf{R} \times \mathsf{R}$ whose samples are obtained by choosing a polynomial $r \xleftarrow{\$} U(\mathsf{R})$ and another polynomial $e \xleftarrow{\$} \mathsf{Ber}_\tau^{\mathsf{R}}$, and outputting $(r, rs + e)$.

2.3 Authentication Protocols

An authentication protocol Π is an interactive protocol executed between a Tag $\mathcal{T}$ and a reader $\mathcal{R}$, both PPT algorithms. Both hold a secret x (generated using a key-generation algorithm KG executed on the security parameter λ in unary) that has been shared in an initial phase. After the execution of the authentication protocol, $\mathcal{R}$ outputs either $\mathtt{accept}$ or $\mathtt{reject}$. We say that the protocol has completeness error ε_c if for all $\lambda \in \mathbb{N}$, all secret keys x generated by $\mathsf{KG}(1^\lambda)$, the honestly executed protocol returns $\mathtt{reject}$ with probability at most ε_c. We now define different security notions of an authentication protocol.

PASSIVE ATTACKS. An authentication protocol is secure against *passive* attacks, if there exists no PPT adversary $\mathcal{A}$ that can make the reader $\mathcal{R}$ return $\mathtt{accept}$ with non-negligible probability after (passively) observing any number of interactions between reader and tag.

ACTIVE ATTACKS. A stronger notion for authentication protocols is security against *active* attacks. Here the adversary $\mathcal{A}$ runs in two stages. First, she can interact with the honest tag a polynomial number of times (with concurrent executions allowed). In the second phase $\mathcal{A}$ interacts with the reader only, and wins if the reader returns $\mathtt{accept}$. Here we only give the adversary one shot to convince the verifier.[5] An authentication protocol is (t, q, ε)-*secure against active adversaries* if every PPT $\mathcal{A}$, running in time at most t and making q queries to the honest reader, has probability at most ε to win the above game.

[5] By using a hybrid argument one can show that this implies security even if the adversary can interact in $k \geq 1$ independent instances concurrently (and wins if the verifier accepts in at least one instance). The use of the hybrid argument looses a factor of k in the security reduction.

3 Ring-LPN and its Hardness

The decisional $\mathsf{Ring\text{-}LPN}^{\mathsf{R}}$ (Ring Learning Parity with Noise in ring R) assumption, formally defined below, states that it is hard to distinguish uniformly random samples in $\mathsf{R} \times \mathsf{R}$ from those sampled from $\Lambda_\tau^{\mathsf{R},s}$ for a uniformly chosen $s \in \mathsf{R}$.

Definition 1 ($\mathsf{Ring\text{-}LPN}^{\mathsf{R}}$)**.** *The (decisional)* $\mathsf{Ring\text{-}LPN}_\tau^{\mathsf{R}}$ *problem is* (t, q, ε)*-hard if for every distinguisher* $\mathcal{D}$ *running in time* t *and making* q *queries,*

$$\left|\Pr\left[s \xleftarrow{\$} \mathsf{R}\ :\ \mathcal{D}^{\Lambda_\tau^{\mathsf{R},s}} = 1\right] - \Pr\left[\ \mathcal{D}^{U(\mathsf{R}\times\mathsf{R})} = 1\right]\right| \leq \varepsilon.$$

3.1 Hardness of LPN and Ring-LPN

One can attempt to solve Ring-LPN using standard algorithms for LPN, or by specialized algorithms that possibly take advantage of Ring-LPN's additional structure. Some work towards constructing the latter type of algorithm has recently been done by Hanrot et al. [HLPS11], who show that in certain cases, the algebraic structure of the Ring-LPN and Ring-LWE problems makes them vulnerable to certain attacks. These attacks essentially utilize a particular relationship between the factorization of the polynomial $f(X)$ and the distribution of the noise.

Ring-LPN with an irreducible $f(X)$ When $f(X)$ is irreducible over F_2, the ring $\mathsf{F}_2[X]/(f)$ is a field. For such rings, the algorithm of Hanrot et al. does not apply, and we do not know of any other algorithm that takes advantage of the added algebraic structure of this particular Ring-LPN instance. Thus to the best of our knowledge, the most efficient algorithms for solving this problem are the same ones that are used to solve LPN, which we will now very briefly recount.

The computational complexity of the LPN problem depends on the length of the secret n and the noise distribution Ber_τ. Intuitively, the larger the n and the closer τ is to $1/2$, the harder the problem becomes. Usually the LPN problem is considered for constant values of τ somewhere between 0.05 and 0.25. For such constant τ, the fastest asymptotic algorithm for the LPN problem, due to Blum et al. [BKW03], takes time $2^{\Omega(n/\log n)}$ and requires approximately $2^{\Omega(n/\log n)}$ samples from the LPN oracle. If one has access to fewer samples, then the algorithm will perform somewhat worse. For example, if one limits the number of samples to only polynomially-many, then the algorithm has an asymptotic complexity of $2^{\Omega(n/\log\log n)}$ [Lyu05]. In our scenario, the number of samples available to the adversary is limited to n times the number of executions of the authentication protocol, and so it is reasonable to assume that the adversary will be somewhat limited in the number of samples he is able to obtain (perhaps at most 2^{40} samples), which should make our protocols harder to break than solving the Ring-LPN problem. Levieil and Fouque [LF06] made some optimizations to the algorithm of Blum et al. and analyzed its precise complexity. To the best of our knowledge, their algorithm is currently the most efficient one and we will refer to their results when analyzing the security of our instantiations.

In Section 5, we base our scheme on the hardness of the $\mathsf{Ring\text{-}LPN}^{\mathsf{R}}$ problem where the ring is $\mathsf{R} = \mathsf{F}_2[X]/(X^{532} + X + 1)$ and $\tau = 1/8$. According to the analysis of [LF06], an LPN problem of dimension 512 with $\tau = 1/8$ would require 2^{77} memory (and thus at least that much time) to solve when given access to approximately as many samples (see [LF06, Section 5.1]). Since our dimension is somewhat larger and the number of samples will be limited in practice, it is reasonable to assume that this instantiation has 80-bit security.

Ring-LPN with a reducible $f(X)$ For efficiency purposes, it is sometimes useful to consider using a polynomial $f(X)$ that is not irreducible over F_2. This will allow us to use the CRT representation of the elements of $\mathsf{F}_2[X]/(f)$ to perform multiplications, which in practice turns out to be more efficient. Ideally, we would like the polynomial f to split into as many small-degree polynomials f_i as possible, but there are some constraints that are placed on the factorization of f both by the security proof, and the possible weaknesses that a splittable polynomial introduces into the Ring-LPN problem.

If the polynomial f splits into $f = \prod_{i=1}^{m} f_i$, then it may be possible to try and solve the Ring-LPN problem modulo some f_i rather than modulo f. Since the degree of f_i is smaller than the degree of f, the resulting Ring-LPN problem may end up being easier. In particular, when we receive a sample $(r, rs+e)$ from the distribution $\Lambda_\tau^{\mathsf{R},s}$, we can rewrite it in CRT form as

$$(\widehat{r}, \widehat{rs+e}) = ((r \bmod f_1, rs + e \bmod f_1), \ldots,\\ (r \bmod f_m, rs + e \bmod f_m)),$$

and thus for every f_i, we have a sample

$$(r \bmod f_i, (r \bmod f_i)(s \bmod f_i) + e \bmod f_i),$$

where all the operations are in the ring (or field) $\mathsf{F}_2[X]/(f_i)$. Thus solving the (decision) Ring-LPN problem in $\mathsf{F}_2[X]/(f)$ reduces to solving the problem in $\mathsf{F}_2[X]/(f_i)$. The latter problem is in a smaller dimension, since $deg(s) > deg(s \bmod f_i)$, but the error distribution of $(e \bmod f_i)$ is quite different than that of e. While each coefficient of e is distributed independently as Ber_τ, each coefficient of $(e \bmod f_i)$ is distributed as the distribution of a sum of certain coefficients of e, and therefore the new error is larger.[6] Exactly which coefficients of e, and more importantly, how many of them, combine to form every particular coefficient of e' depends on the polynomial f_i. For example, if

$$f(X) = (X^3 + X + 1)(X^3 + X^2 + 1)$$

and $e = \sum\limits_{i=0}^{5} e_i X^i$, then,

$$e' = e \bmod (X^3 + X + 1) = (e_0 + e_3 + e_5) + (e_1 + e_3 + e_4 + e_5)X + (e_2 + e_4 + e_5)X^2,$$

and thus every coefficient of the error e' is comprised of at least 3 coefficients of the error vector e, and thus $\tau' > \frac{1}{2} - \frac{(1-2\tau)^3}{2}$.

In our instantiation of the scheme with a reducible $f(X)$ in Section 5, we used the $f(X)$ such that it factors into f_i's that make the operations in CRT form relatively fast, while making sure that the resulting Ring-LPN problem modulo each f_i is still around 2^{80}-hard.

4 Authentication Protocol

In this section we describe our new 2-round authentication protocol and prove its active security under the hardness of the Ring-LPN problem. Detailed implementation details will be given in Section 5.

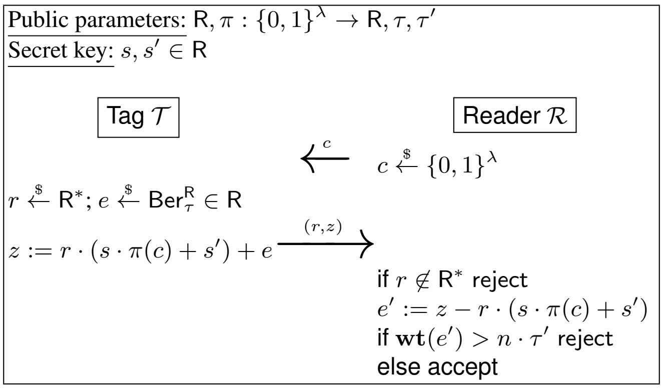
4.1 The Protocol

Our authentication protocol is defined over the ring $\mathsf{R} = \mathsf{F}_2[X]/(f)$ and involves a "suitable" mapping $\pi : \{0,1\}^\lambda \to \mathsf{R}$. We call π *suitable* for ring R if for all $c, c' \in \{0,1\}^\lambda$, $\pi(c) - \pi(c') \in \mathsf{R} \setminus \mathsf{R}^*$ iff $c = c'$. We will discuss the necessity and existence of such mappings after the proof of Theorem 1

- Public parameters. The authentication protocol has the following public parameters, where τ, τ' are constants and n depend on the security parameter λ.

R, n	ring $\mathsf{R} = \mathsf{F}_2[X]/(f)$, $\deg(f) = n$
$\pi : \{0,1\}^\lambda \to \mathsf{R}$	mapping
$\tau \in (0, 1/2)$	parameter of Bernoulli distribution
$\tau' \in (\tau, 1/2)$	acceptance threshold

[6] If we have k elements $e_1, \ldots, e_k \overset{\$}{\leftarrow} \mathsf{Ber}_\tau$, then a simple calculation shows that the element $e' = e_1 + \ldots + e_k$ is distributed as $\mathsf{Ber}_{\tau'}$ where $\tau' = \frac{1}{2} - \frac{(1-2\tau)^k}{2}$.



Public parameters: $\mathsf{R}, \pi : \{0,1\}^\lambda \to \mathsf{R}, \tau, \tau'$

Secret key: $s, s' \in \mathsf{R}$

Tag $\mathcal{T}$

Reader $\mathcal{R}$

$\xleftarrow{c}$ $\quad c \xleftarrow{\$} \{0,1\}^\lambda$

$r \xleftarrow{\$} \mathsf{R}^*; e \xleftarrow{\$} \mathsf{Ber}_\tau^\mathsf{R} \in \mathsf{R}$

$z := r \cdot (s \cdot \pi(c) + s') + e \xrightarrow{(r,z)}$

if $r \notin \mathsf{R}^*$ reject

$e' := z - r \cdot (s \cdot \pi(c) + s')$

if $\mathbf{wt}(e') > n \cdot \tau'$ reject

else accept

Fig. 1. Two-round authentication protocol with active security from the $\mathsf{Ring\text{-}LPN}^\mathsf{R}$ assumption.

- Key Generation. Algorithm $\mathsf{KG}(1^\lambda)$ samples $s, s' \xleftarrow{\$} \mathsf{R}$ and returns s, s' as the secret key.
- Authentication Protocol. The Reader $\mathcal{R}$ and the Tag $\mathcal{T}$ share secret value $s, s' \in \mathsf{R}$. To be authenticated by a Reader, the Tag and the Reader execute the authentication protocol from Figure 1.

4.2 Analysis

For our analysis we define for $x, y \in]0,1[$ the following constant:

$$c(x,y) := \left(\frac{x}{y}\right)^x \left(\frac{1-x}{1-y}\right)^{1-x}.$$

We now state that our protocol is secure against active adversaries. Recall that active adversaries can arbitrarily interact with a Tag oracle in the first phase and tries to impersonate the Reader in the 2nd phase.

Theorem 1. *If ring mapping π is suitable for ring R and the $\mathsf{Ring\text{-}LPN}_\mathsf{R}$ problem is (t, q, ε)-hard then the authentication protocol from Figure 1 is (t', q, ε')-secure against active adversaries, where*

$$t' = t - q \cdot \exp(R) \qquad \varepsilon' = \varepsilon + q \cdot 2^{-\lambda} + c(\tau', 1/2)^{-n} \tag{4.1}$$

and $\exp(R)$ is the time to perform $O(1)$ exponentiations in R. Furthermore, the protocol has completeness error $\varepsilon_c(\tau, \tau', n) \approx c(\tau', \tau)^{-n}$.

Proof. The completeness error $\varepsilon_c(\tau, \tau', n)$ is (an upper bound on) the probability that an honestly generated Tag gets rejected. In our protocol this is exactly the case when the error e' has weight $\geq n \cdot \tau'$, i.e.

$$\varepsilon_c(\tau, \tau', n) = \Pr[\mathbf{wt}(e') > n \cdot \tau' \;:\; e \xleftarrow{\$} \mathsf{Ber}_\tau^\mathsf{R}]$$

Levieil and Fouque [LF06] show that one can approximate this probability as $\varepsilon_c \approx c(\tau', \tau)^{-n}$.

To prove the security of the protocol against active attacks we proceed in sequences of games. Game_0 is the security experiment describing an active attack on our scheme by an adversary $\mathcal{A}$ making q queries and running in time t', i.e.

- Sample the secret key $s, s' \xleftarrow{\$} \mathsf{R}$.
- (1st phase of active attack) $\mathcal{A}$ queries the tag $\mathcal{T}$ on $c \in \{0,1\}^\lambda$ and receives (r, z) computed as illustrated in Figure 1.
- (2nd phase of active attack) $\mathcal{A}$ gets a random challenge $c^* \xleftarrow{\$} \{0,1\}^\lambda$ and outputs (r, z). $\mathcal{A}$ wins if the reader $\mathcal{R}$ accepts, i.e. $\mathbf{wt}(z - r \cdot (s \cdot \pi(c^*) + s')) \leq n \cdot \tau'$.

By definition we have $\Pr[\mathcal{A} \text{ wins in } \mathsf{Game}_0] \leq \varepsilon'$.

Game_1 is as Game_0, except that all the values (r, z) returned by the Tag oracle in the first phase (in return to a query $c \in \{0,1\}^\lambda$) are uniform random elements $(r, z) \in \mathsf{R}^2$. We now show that if $\mathcal{A}$ is successful against Game_0, then it will also be successful against Game_1.

Claim. $|\Pr[\mathcal{A} \text{ wins in } \mathsf{Game}_1] - \Pr[\mathcal{A} \text{ wins in } \mathsf{Game}_0]| \leq \varepsilon + q \cdot 2^{-\lambda}$

To prove this claim, we construct an adversary $\mathcal{D}$ (distinguisher) against the Ring-LPN problem which runs in time $t = t' + \exp(\mathsf{R})$ and has advantage

$$\varepsilon \geq |\Pr[\mathcal{A} \text{ wins in } \mathsf{Game}_1] - \Pr[\mathcal{A} \text{ wins in } \mathsf{Game}_0]| - q \cdot 2^{-\lambda}$$

$\mathcal{D}$ has access to a Ring-LPN oracle $\mathcal{O}$ and has to distinguish between $\mathcal{O} = \Lambda_\tau^{\mathsf{R},s}$ for some secret $s \in \mathsf{R}$ and $\mathcal{O} = U(\mathsf{R} \times \mathsf{R})$.

- $\mathcal{D}$ picks a random challenge $c^* \overset{\$}{\leftarrow} \{0,1\}^\lambda$ and $a \overset{\$}{\leftarrow} \mathsf{R}$. Next, it runs $\mathcal{A}$ and simulates its view with the unknown secret s, s', where $s \in \mathsf{R}$ comes from the oracle $\mathcal{O}$ and s' is implicitly defined as $s' := -\pi(c^*) \cdot s + a \in \mathsf{R}$.
- In the 1st phase, $\mathcal{A}$ can make q (polynomial many) queries to the Tag oracle. On query $c \in \{0,1\}^\lambda$ to the Tag oracle, $\mathcal{D}$ proceeds as follows. If $\pi(c) - \pi(c^*) \not\in \mathsf{R}^*$, then abort. Otherwise, $\mathcal{D}$ queries its oracle $\mathcal{O}()$ to obtain $(r', z') \in \mathsf{R}^2$. Finally, $\mathcal{D}$ returns (r, z) to $\mathcal{A}$, where

$$r := r' \cdot (\pi(c) - \pi(c^*))^{-1}, \quad z := z' + ra. \tag{4.2}$$

- In the 2nd phase, $\mathcal{D}$ uses $c^* \in \{0,1\}^\lambda$ to challenge $\mathcal{A}$. On answer (r, z), $\mathcal{D}$ returns 0 to the Ring-LPN game if $\mathbf{wt}(z - r \cdot a) > n \cdot \tau'$ or $r \not\in \mathsf{R}^*$, and 1 otherwise. Note that $s\pi(c^*) + s' = (\pi(c^*) - \pi(c^*))s + a = a$ and hence the above check correctly simulates the output of a reader with the simulated secret s, s'.

Note that the running time of $\mathcal{D}$ is that of $\mathcal{A}$ plus $O(q)$ exponentiations in R.

Let bad be the event that for at least one query c made by $\mathcal{A}$ to the Tag oracle, we have that $\pi(c) - \pi(c^*) \not\in \mathsf{R}^*$. Since c^* is uniform random in R and hidden from $\mathcal{A}$'s view in the first phase we have by the union bound over the q queries

$$\begin{aligned} \Pr[\mathsf{bad}] &\leq q \cdot \Pr_{c^* \in \{0,1\}^\lambda}[\pi(c) - \pi(c^*) \in \mathsf{R} \setminus \mathsf{R}^*] \\ &= q \cdot 2^{-\lambda}. \end{aligned} \tag{4.3}$$

The latter inequality holds because π is suitable for R.

Let us now assume bad does not happen. If $\mathcal{O} = \Lambda_\tau^{\mathsf{R},s}$ is the real oracle (i.e., it returns (r', z') with $z' = r's + e$) then by the definition of (r, z) from (4.2),

$$z = (r's + e) + ra = r(\pi(c)s - \pi(c^*)s + a) + e = r(s\pi(c) + s') + e.$$

Hence the simulation perfectly simulates $\mathcal{A}$'s view in Game_0. If $\mathcal{O} = U(\mathsf{R} \times \mathsf{R})$ is the random oracle then (r, z) are uniformly distributed, as in Game_1. That concludes the proof of Claim 4.2.

We next upper bound the probability that $\mathcal{A}$ can be successful in Game_1. This bound will be information theoretic and even holds if $\mathcal{A}$ is computationally unbounded and can make an unbounded number of queries in the 1st phase. To this end we introduce the minimal soundness error, ε_{ms}, which is an upper bound on the probability that a tag (r, z) chosen independently of the secert key is valid, i.e.

$$\varepsilon_{\text{ms}}(\tau', n) := \max_{(z,r) \in \mathsf{R} \times \mathsf{R}^*} \Pr_{s,s' \overset{\$}{\leftarrow} \mathsf{R}}[\mathbf{wt}(\underbrace{z - r \cdot (s \cdot \pi(c^*) + s')}_{e'}) \leq n\tau']$$

As $r \in \mathsf{R}^*$ and $s' \in \mathsf{R}$ is uniform, also $e' = z - r \cdot (s \cdot \pi(c^*) + s'$ is uniform, thus $\varepsilon_{\mathrm{ms}}$ is simply

$$\varepsilon_{\mathrm{ms}}(\tau', n) := \Pr_{e' \xleftarrow{\$} \mathsf{R}} [\mathbf{wt}(e') \leq n\tau']$$

Again, it was shown in [LF06] that this probability can be approximated as

$$\varepsilon_{\mathrm{ms}}(\tau', n) \approx c(\tau', 1/2)^{-n}. \tag{4.4}$$

Clearly, $\varepsilon_{\mathrm{ms}}$ is a trivial lower bound on the advantage of $\mathcal{A}$ in forging a valid tag, by the following claim in Game_1 one cannot do any better than this.

Claim. $\Pr[\mathcal{A}$ wins in $\mathsf{Game}_1] = \varepsilon_{\mathrm{ms}}(\tau', n)$

To see that this claim holds one must just observe that the answers $\mathcal{A}$ gets in the first phase of the active attack in Game_1 are independent of the secret s, s'. Hence $\mathcal{A}$'s advantage is $\varepsilon_{\mathrm{ms}}(\tau', n)$ by definition.

Claims 4.2 and 4.2 imply (4.1) and conclude the proof of Theorem 1.

We require the mapping $\pi : \{0,1\}^\lambda \rightarrow \mathsf{R}$ used in the protocol to be *suitable* for R, i.e. for all $c, c' \in \{0,1\}^\lambda$, $\pi(c) - \pi(c') \in \mathsf{R} \setminus \mathsf{R}^*$ iff $c = c'$. In Section 5 we describe efficient suitable maps for any $\mathsf{R} = \mathsf{F}_2[X]/(f)$ where f has no factor of degree $\leq \lambda$. This condition is necessary, as no suitable mapping exists if f has a factor f_i of degree $\leq \lambda$: in this case, by the pigeonhole principle, there exist distinct $c, c' \in \{0,1\}^\lambda$ such that $\pi(c) = \pi(c') \mod f_i$, and thus $\pi(c) - \pi(c') \in \mathsf{R} \setminus \mathsf{R}^*$.

We stress that for our security proof we need π to be suitable for R, since otherwise (4.3) is no longer guaranteed to hold. It is an interesting question if this is inherent, or if the security of our protocol can be reduced to the $\mathsf{Ring\text{-}LPN}^{\mathsf{R}}$ problem for arbitrary rings $\mathsf{R} = \mathsf{F}_2[X]/(f)$, or even $\mathsf{R} = \mathsf{F}_q[X]/(f)$ (This is interesting since, if f has factors of degree $\ll \lambda$, the protocol could be implemented more efficiently and even become based on the worst-case hardness of lattice problems). Similarly, it is unclear how to prove security of our protocol instantiated with Toeplitz matrices.

5 Implementation

There are two objectives that we pursue with the implementation of our protocol. First, we will show that the protocol is in fact practical with concrete parameters, even on extremely constrained CPUs. Second, we investigate possible application scenarios where the protocol might have additional advantages. From a practical point of view, we are particularly interested in comparing our protocol to classical symmetric challenge-response schemes employing AES. Possible advantages of the protocol at hand are (i) the security properties and (ii) improved implementation properties. With respect to the former aspect, our protocol has the obvious advantage of being provably secure under a reasonable and static hardness assumption. Even though AES is arguably the most trusted symmetric cipher, it is "merely" computationally secure with respect to known attacks.

In order to investigate implementation properties, constrained microprocessors are particularly relevant. We chose an 8-bit AVR ATmega163 [Atm] based smartcard, which is widely used in myriads of embedded applications. It can be viewed as a typical representative of a CPU used in tokens that are in need for an authentication protocol, e.g., computational RFID tags or (contactless) smart cards. The main metrics we consider for the implementation are run-time and code size. We note at this point that in many lightweight crypto applications, code size is the most precious resource once the run-time constraints are fulfilled. This is due to the fact that EEPROM or flash memory is often heavily constrained. For instance, the WISP, a computational RFID tag, has only 8 kBytes of program memory [Wik,MSP].

We implemented two variants of the protocol described in Section 4. The first variant uses a ring $\mathsf{R} = \mathsf{F}_2[X]/(f)$, where f splits into five irreducible polynomials; the second variant uses a field, i.e., f is irreducible. For both implementations, we chose parameters which provide a security level of $\lambda = 80$ bits, i.e., the parameters are chosen such that ε' in (4.1) is bounded by 2^{-80} and the completeness ε_c is bounded by 2^{-40}. This security level is appropriate for the lightweight applications which we are targeting.

5.1 Implementation with a Reducible Polynomial

From an implementation standpoint, the case of reducible polynomial is interesting since one can take advantage of arithmetic based on the Chinese Remainder Theorem.

PARAMETERS. To define the ring $\mathsf{R} = \mathsf{F}_2[X]/(f)$, we chose the reducible polynomial f to be the product of the $m = 5$ irreducible pentanomials specified by the following powers with non-zero coefficients: $(127, 8, 7, 3, 0)$, $(126, 9, 6, 5, 0)$, $(125, 9, 7, 4, 0)$, $(122, 7, 4, 3, 0)$, $(121, 8, 5, 1, 0)$[7]. Hence f is a polynomial of degree $n = 621$. We chose $\tau = 1/6$ and $\tau' = .29$ to obtain minimal soundness error $\varepsilon_{\text{ms}} \approx c(\tau', 1/2)^{-n} \leq 2^{-82}$ and completeness error $\varepsilon_{\text{c}} \leq 2^{-42}$. From the discussion of Section 3 the best known attack on $\mathsf{Ring\text{-}LPN}_\tau^{\mathsf{R}}$ with the above parameters has complexity $> 2^{80}$. The mapping $\pi : \{0,1\}^{80} \to \mathsf{R}$ is defined as follows. On input $c \in \{0,1\}^{80}$, for each $1 \leq i \leq 5$, pad $c \in \{0,1\}^{80}$ with $\deg(f_i) - 80$ zeros and view the result as coefficients of an element $v_i \in \mathsf{F}_2[X]/(f_i)$. This defines $\pi(c) = (v_1, \ldots, v_5)$ in CRT representation. Note that, for fixed $c, c^* \in \{0,1\}^{80}$, we have that $\pi(c) - \pi(c^*) \in \mathsf{R} \setminus \mathsf{R}^*$ iff $c = c^*$ and hence π is *suitable* for R.

IMPLEMENTATION DETAILS. The main operations are multiplications and additions of polynomials that are represented by 16 bytes. We view the CRT-based multiplication in three stages. In the first stage, the operands are reduced modulo each of the five irreducible polynomials. This part has a low computational complexity. Note that only the error e has to be chosen in the ring and afterwards transformed to CRT representation. It is possible to save the secret key (s, s') and to generate r directly in the CRT representation. This is not possible for e because e has to come from $\mathsf{Ber}_\tau^{\mathsf{R}}$. In the second stage, one multiplication in each of the finite fields defined by the five pentanomials has to be performed. We used the right-to-left comb multiplication algorithm from [HMV03]. For the multiplication with $\pi(c)$ we exploit the fact that only the first 80 coefficients can be non-zero. Hence we wrote one function for *normal* multiplication and one for *sparse* multiplication. The latter is more than twice as fast as the former. The subsequent reduction takes care of the special properties of the pentanomials, thus code reuse is not possible for the different fields. The third stage, constructing the product polynomial in the ring, is shifted to the prover (RFID reader) which normally has more computational power than the tag $\mathcal{T}$. Hence the response (r, z) is sent in CRT form to the reader. If non-volatile storage — in our case we need $2 \cdot 5 \cdot 16 = 160$ bytes — is available we can heavily reduce the response time of the tag. At an arbitrary point in time, choose e and r according to their distribution and precompute $\mathit{tmp}_1 = r \cdot s$ and $\mathit{tmp}_2 = r \cdot s' + e$. When a challenge c is received afterwards, tag $\mathcal{T}$ only has to compute $z = \mathit{tmp}_1 \cdot \pi(c) + \mathit{tmp}_2$. Because $\pi(c)$ is sparse, the tag can use the *sparse* multiplication and response very quickly. The results of the implementation are shown in Table 2 in Section 5.3. Note that all multiplication timings given already include the necessary reductions and addition of a value according to Figure 1.

5.2 Implementation with an Irreducible Polynomial

PARAMETERS. To define the field $\mathsf{F} = \mathsf{F}_2[X]/(f)$, we chose the irreducible trinomial $f(X) = X^{532} + X + 1$ of degree $n = 532$. We chose $\tau = 1/8$ and $\tau' = .27$ to obtain minimal soundness error $\varepsilon_{\text{ms}} \approx c(\tau', 1/2)^{-n} \leq 2^{-80}$ and completeness error $\varepsilon_{\text{c}} \approx 2^{-55}$. From the discussion in Section 3 the best known attack on $\mathsf{Ring\text{-}LPN}_\tau^{\mathsf{F}}$ with the above parameters has complexity $> 2^{80}$. The mapping $\pi : \{0,1\}^{80} \to \mathsf{F}$ is defined as follows. View $c \in \{0,1\}^{80}$ as $c = (c_1, \ldots, c_{16})$ where c_i is a number between 1 and 32. Define the coefficients of the polynomial $v = \pi(c) \in \mathsf{F}$ as zero except all positions i of the form $i = 16 \cdot (j-1) + c_j$, for some $j = 1, \ldots, 16$. Hence $\pi(c)$ is sparse, i.e., it has exactly 16 non-zero coefficients. Since π is injective and F is a field, the mapping π is suitable for F.

IMPLEMENTATION DETAILS. The main operation for the protocol is now a 67-byte multiplication. Again we used the right-to-left comb multiplication algorithm from [HMV03] and an optimized reduction algorithm. Like in the reducible case, the tag can do similar precomputations if $2 \cdot 67 = 134$ bytes non-volatile

[7] $(127, 8, 7, 3, 0)$ refers to the polynomial $X^{127} + X^8 + X^7 + X^3 + 1$.

storage are available. Because of the special type of the mapping $v = \pi(c)$, the gain of the *sparse* multiplication is even larger than in the reducible case. Here we are a factor of 7 faster, making the response time with precomputations faster, although the field is larger. The results are shown in Table 3 in Section 5.3.

5.3 Implementation Results

All results presented in this section consider only the clock cycles of the actual arithmetic functions. The communication overhead and the generation of random bytes is excluded because they occur in every authentication scheme, independent of the underlying cryptographic functions. The time for building e from $\mathsf{Ber}_{\tau}^{\mathsf{R}}$ out of the random bytes and converting it to CRT form is included in *Overhead*. Table 2 and Table 3 shows the results for the ring based and field based variant, respectively.

Table 2. Results for the ring based variant w/o precomputation

Aspect	**time** **in cycles**	**code size** **in bytes**
Overhead	$17,500$	264
Mul	$5 \times 13,000$	164
sparse Mul	$5 \times 6,000$	170
total	$112,500$	1356

The overall code size is not the sum of the other values because, as mentioned before, the same multiplication code is used for all *normal* and *sparse* multiplications, respectively, while the reduction code is different for every field ($\approx$ 134 byte each). The same code for reduction is used independently of the type of the multiplication for the same field. If precomputation is acceptable, the tag can answer the challenge after approximately $30,000$ clock cycles, which corresponds to a 15 msec if the CPU is clocked at 2 MHz.

Table 3. Results for the field based variant w/o precomputation

Aspect	**time** **in cycles**	**code size** **in bytes**
Overhead	$3,000$	150
Mul	$150,000$	161
sparse Mul	$21,000$	148
total	$174,000$	459

For the field-based protocol, the overall performance is slower due to the large operands used in the multiplication routine. But due to the special mapping $v = \pi(c)$, here the tag can do a sparse multiplications in only $21,000$ clocks cycles. This allows the tag to respond in 10.5 msec at 2 MHz clock rate if non-volatile storage is available.

As mentioned in the introduction, we want to compare our scheme with a conventional challenge-response authentication protocol based on AES. The tag's main operation in this case is one AES encryption. The implementation in [LLS09] states $8,980$ clock cycles for one encryption on a similar platform, but unfortunately no code size is given; [Tik] reports 10121 cycles per encryption and a code size of 4644 bytes.[8] In comparison with these highly optimized AES implementations, our scheme is around eleven times slower when using the ring based variant without precomputations. If non-volatile storage allows

[8] An internet source [Poe] claims to encrypt in 3126 cycles with code size of 3098 bytes but since this is unpublished material we do not consider it in our comparison.

precomputations, the ring based variant is only three times slower than AES. But the code size is by a factor of two to three smaller, making it attractive for Flash constrained devices. The field based variant without precomputations is 17 to 19 times slower than AES, but with precompuations it is only twice as slow as AES, while only consuming one tenths of the code size. From a practical point of view, it is important to note that even our slowest implementation is executed in less than 100 msec if the CPU is clocked at 2 MHz. This response time is sufficient in many application scenarios. (For authentications involving humans, a delay of 1 sec is often considered acceptable.)

The performance drawback compared to AES is not surprising, but it is considerably less dramatic compared to asymmetric schemes like RSA or ECC [GPW$^+$04]. But exploiting the special structure of the multiplications in our scheme and using only a small amount of non-volatile data memory provides a response time in the same order of magnitude as AES, while keeping the code size much smaller.

6 Conclusions and open Problems

We proposed a new [KPC$^+$11]-like authentication protocol with provable security against active attacks based on the Ring-LPN assumption, consisting of only two rounds, and having small communication complexity. Furthermore, our implementations on an 8-bit AVR ATmega163 based smartcard demonstrated that it has very small code size and its efficiency can be of the same order as traditional AES-based authentication protocols. Overall, we think that its features make it very applicable in scenarios that involve low-cost, resource-constrained devices.

Our protocol cannot be proved secure against man-in-the-middle (MIM) attacks, but using a recent transformation from [DKPW12] we can get a MIM secure scheme with small extra cost (one application of a universal hash function.) Still, finding a more direct construction which achieves MIM security (or proving that the current protocol already has this property) but doesn't require any hashing remains an interesting open problem.

We believe that the Ring-LPN assumption is very natural and will find further cryptographic applications, especially for constructions of schemes for low-cost devices. In particular, we think that if the LPN-based line of research is to lead to a practical protocol in the future, then the security of this protocol will be based on a hardness assumption with some “extra algebraic structure”, such as Ring-LPN in this work, or LPN with Toeplitz matrices in the work of Gilbert et al. [GRS08a]. More research, however, needs to be done on understanding these problems and their computational complexity. In terms of Ring-LPN, it would be particularly interesting to find out whether there exists an equivalence between the decision and the search versions of the problem similar to the reductions that exist for LPN [BFKL93,Reg09,KS06a] and Ring-LWE [LPR10].

7 Acknowledgements.

We would like to thank the anonymous referees of this conference and those of the *ECRYPT Workshop on Lightweight Cryptography* for very useful comments, and in particular for the suggestion that the scheme is somewhat vulnerable to a man-in-the-middle attack whenever an adversary observes two reader challenges that are the same. We hope that the attack we described in Appendix A corresponds to what the reviewer had in mind. This work was partially supported by the European Research Council.

References

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai, *Fast cryptographic primitives and circular-secure encryption based on hard learning problems*, CRYPTO 2009 (Shai Halevi, ed.), LNCS, vol. 5677, Springer, August 2009, pp. 595–618.

[Atm] Atmel, *ATmega163 datasheet*, ”www.atmel.com/atmel/acrobat/doc1142.pdf”.

[BFKL93] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton, *Cryptographic primitives based on hard learning problems*, CRYPTO, 1993, pp. 278–291.

[BKL$^+$07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe, *PRESENT: An ultra-lightweight block cipher*, CHES 2007 (Pascal Paillier and Ingrid Verbauwhede, eds.), LNCS, vol. 4727, Springer, September 2007, pp. 450–466.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman, *Noise-tolerant learning, the parity problem, and the statistical query model*, J. ACM **50** (2003), no. 4, 506–519.

[DKPW12] Yevgeniy Dodis, Eike Kiltz, Krzysztof Pietrzak, and Daniel Wichs, *Message authentication, revisited*, EUROCRYPT, 2012.

[DR02] Joan Daemen and Vincent Rijmen, *The design of rijndael: AES - the advanced encryption standard*, Springer, 2002.

[GPW$^+$04] Nils Gura, Arun Patel, Arvinderpal W, Hans Eberle, and Sheueling Chang Shantz, *Comparing elliptic curve cryptography and RSA on 8-bit CPUs*, Cryptographic Hardware and Embedded Systems - CHES 2004, 2004, pp. 119–132.

[GRS05] Henri Gilbert, Matt Robshaw, and Herve Sibert, *An active attack against HB+ – a provably secure lightweight authentication protocol*, Cryptology ePrint Archive, Report 2005/237, 2005, `http://eprint.iacr.org/`.

[GRS08a] Henri Gilbert, Matthew J. B. Robshaw, and Yannick Seurin, *HB$^\sharp$: Increasing the security and efficiency of HB$^+$*, EUROCRYPT 2008 (Nigel P. Smart, ed.), LNCS, vol. 4965, Springer, April 2008, pp. 361–378.

[GRS08b] ———, *How to encrypt with the LPN problem*, ICALP 2008, Part II (Luca Aceto, Ivan Damgard, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, eds.), LNCS, vol. 5126, Springer, July 2008, pp. 679–690.

[HB00] N. Hopper and M. Blum, *A secure human-computer authentication scheme*, Tech. Report CMU-CS-00-139, Carnegie Mellon University, 2000.

[HB01] Nicholas J. Hopper and Manuel Blum, *Secure human identification protocols*, ASIACRYPT 2001 (Colin Boyd, ed.), LNCS, vol. 2248, Springer, December 2001, pp. 52–66.

[HLPS11] Guillaume Hanrot, Vadim Lyubashevsky, Chris Peikert, and Damien Stehlé, *Personal communication*, 2011.

[HMV03] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone, *Guide to elliptic curve cryptography*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[JW05] Ari Juels and Stephen A. Weis, *Authenticating pervasive devices with human protocols*, CRYPTO 2005 (Victor Shoup, ed.), LNCS, vol. 3621, Springer, August 2005, pp. 293–308.

[KPC$^+$11] Eike Kiltz, Krzysztof Pietrzak, David Cash, Abhishek Jain, and Daniele Venturi, *Efficient authentication from hard learning problems*, EUROCRYPT, 2011, pp. 7–26.

[KS06a] Jonathan Katz and Ji Sun Shin, *Parallel and concurrent security of the HB and HB+ protocols*, EUROCRYPT 2006 (Serge Vaudenay, ed.), LNCS, vol. 4004, Springer, May / June 2006, pp. 73–87.

[KS06b] Jonathan Katz and Adam Smith, *Analyzing the HB and HB+ protocols in the "large error" case*, Cryptology ePrint Archive, Report 2006/326, 2006, `http://eprint.iacr.org/`.

[KSS10] Jonathan Katz, Ji Sun Shin, and Adam Smith, *Parallel and concurrent security of the HB and HB+ protocols*, Journal of Cryptology **23** (2010), no. 3, 402–421.

[KW05] Ziv Kfir and Avishai Wool, *Picking virtual pockets using relay attacks on contactless smartcard*, Security and Privacy for Emerging Areas in Communications Networks, International Conference on **0** (2005), 47–58.

[KW06] Ilan Kirschenbaum and Avishai Wool, *How to build a low-cost, extended-range RFID skimmer*, Proceedings of the 15th USENIX Security Symposium (SECURITY 2006), USENIX Association, August 2006, pp. 43–57.

[LF06] Éric Levieil and Pierre-Alain Fouque, *An improved LPN algorithm*, SCN 06(Roberto De Prisco and Moti Yung, eds.), LNCS, vol. 4116, Springer, September 2006, pp. 348–359.

[LLS09] Hyubgun Lee, Kyounghwa Lee, and Yongtae Shin, *AES implementation and performance evaluation on 8-bit microcontrollers*, CoRR **abs/0911.0482** (2009).

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev, *On ideal lattices and learning with errors over rings*, EUROCRYPT 2010 (Henri Gilbert, ed.), LNCS, vol. 6110, Springer, May 2010, pp. 1–23.

[Lyu05] Vadim Lyubashevsky, *The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem*, APPROX-RANDOM, 2005, pp. 378–389.

[MSP] *MSP430 datasheeet.*

[OOV08] Khaled Ouafi, Raphael Overbeck, and Serge Vaudenay, *On the security of HB$^\#$ against a man-in-the-middle attack*, ASIACRYPT, 2008, pp. 108–124.

[Poe] B. Poettering, *AVRAES: The AES block cipher on AVR controllers*, "`http://point-at-infinity.org/avraes/`".

[Reg09] Oded Regev, *On lattices, learning with errors, random linear codes, and cryptography*, J. ACM **56** (2009), no. 6.

[Tik] Jeff Tikkanen, *AES implementation on AVR ATmega328p*, "`http://cs.ucsb.edu/\~koc/cs178/projects/JT/avr\_aes.html`".

[Wik] WISP Wiki, *WISP 4.0 DL hardware*, "`http://wisp.wikispaces.com/WISP+4.0+DL`".

A Man-in-the-Middle Attack

In this section, we sketch a man-in-the-middle attack against the protocol in Figure 1 that recovers the secret key in time approximately $O\left(n^{1.5} \cdot 2^{\lambda/2}\right)$ when the adversary is able to insert himself into that many valid interactions between the reader and the tag. For a ring $\mathsf{R} = \mathsf{F}_2[X]/(f)$ and a polynomial $g \in \mathsf{R}$, define the vector $\boldsymbol{g}$ to be a vector of dimension $deg(f)$ whose i^{th} coordinate is the X^i coefficient of g. Similarly, for a polynomial $h \in \mathsf{R}$, let $Rot(h)$ be a $deg(f) \times deg(f)$ matrix whose i^{th} column (for $0 \leq i < deg(f)$) is $\overrightarrow{h \cdot X^i}$, or in other words, the coefficients of the polynomial $h \cdot X^i$ in the ring R. From this description, one can check that for two polynomials $g, h \in \mathsf{R}$, the product $\overrightarrow{g \cdot h} = Rot(g) \cdot \boldsymbol{h} \bmod 2 = Rot(h) \cdot \boldsymbol{g} \bmod 2$.

We now move on to describing the attack. The i^{th} (successful) interaction between a reader $\mathcal{R}$ and a tag $\mathcal{T}$ consists of the reader sending the challenge c_i, and the tag replying with the pair (r_i, z_i) where $z_i - r_i \cdot (s \cdot \pi(c_i) + s')$ is a low-weight polynomial of weight at most $n \cdot \tau'$. The adversary who is observing this interaction will forward the challenge c_i untouched to the tag, but reply to the reader with the ordered pair $(r_i, z_i' = z_i + e_i)$ where e_i is a vector that is strategically chosen with the hope that the vector $z_i' - r_i \cdot (s \cdot \pi(c_i) + s')$ is *exactly* of weight $n \cdot \tau'$. It's not hard to see that it's possible to choose such a vector e_i so that the probability of $z_i' - r_i \cdot (s \cdot \pi(c_i) + s')$ being of weight $n \cdot \tau'$ is approximately $1/\sqrt{n}$. The response (r_i, z_i') will still be valid, and so the reader will accept. By the birthday bound, after approximately $2^{\lambda/2}$ interactions, there will be a challenge c_j that is equal to some previous challenge c_i. In this case, the adversary replies to the reader with (r_i, z_i''), where the polynomial z_i'' is just the polynomial z_i' whose first bit (i.e. the constant coefficient) is flipped. What the adversary is hoping for is that the reader accepted the response (r_i, z_i') but rejects (r_i, z_i''). Notice that the only way this can happen is if the first bit of z_i' is equal to the first bit of $r_i \cdot (s \cdot \pi(c_i) + s')$, and thus flipping it, increases the error by 1 and makes the reader reject. We now explain how finding such a pair of responses can be used to recover the secret key.

Since the polynomial expression $z_i' - r_i \cdot (s \cdot \pi(c_i) + s') = z_i' - r_i \cdot \pi(c_i) \cdot s - r_i \cdot s'$ can be written as matrix-vector multiplications as

$$\boldsymbol{z_i'} - Rot(r_i \cdot \pi(c_i)) \cdot \boldsymbol{s} - Rot(r_i) \cdot \boldsymbol{s'} \bmod 2,$$

if we let the first bit of $\boldsymbol{z_i'}$ be β_i, the first row of $Rot(r_i \cdot \pi(c_i))$ be $\boldsymbol{a_i}$ and the first row of $Rot(r_i)$ be $\boldsymbol{b_i}$, then we obtain the linear equation

$$\langle \boldsymbol{a_i}, \boldsymbol{s} \rangle + \langle \boldsymbol{b_i}, \boldsymbol{s'} \rangle = \beta_i.$$

To recover the entire secret s, s', the adversary needs to repeat the above attack until he obtains $2n$ linearly-independent equations (which can be done with $O(n)$ successful attacks), and then use Gaussian elimination to recover the full secret.

Higher-Order Masking Schemes for S-Boxes

Claude Carlet[1], Louis Goubin[2], Emmanuel Prouff[3], Michael Quisquater[2], and Matthieu Rivain[4]

[1] LAGA, Université de Paris 8
claude.carlet@univ-paris8.fr
[2] Université de Versailles St-Quentin-en-Yvelines
louis.goubin@prism.uvsq.fr
michael.quisquater@prism.uvsq.fr
[3] Oberthur Technologies
e.prouff@oberthur.com
[4] CryptoExperts
matthieu.rivain@cryptoexperts.com

Abstract. Masking is a widely used countermeasure against side-channel attacks. The principle is to randomly split every sensitive intermediate variable occurring in the computation into $d+1$ shares, where d is called the *masking order* and plays the role of a security parameter. The main issue while applying masking to protect a block cipher implementation is to design an efficient scheme for the s-box computations. Actually, masking schemes with arbitrary order only exist for Boolean circuits and for the AES s-box. Although any s-box can be represented as a Boolean circuit, applying such a strategy leads to inefficient implementation in software. The design of an efficient and generic higher-order masking scheme was hence until now an open problem. In this paper, we introduce the first masking schemes which can be applied in software to efficiently protect any s-box at any order. We first describe a general masking method and we introduce a new criterion for an s-box that relates to the best efficiency achievable with this method. Then we propose concrete schemes that aim to approach the criterion. Specifically, we give optimal methods for the set of *power functions*, and we give efficient heuristics for the general case. As an illustration we apply the new schemes to the DES and PRESENT s-boxes and we provide implementation results.

1 Introduction

Side-channel analysis is a class of cryptanalytic attacks that exploit the physical environment of a cryptosystem to recover some *leakage* about its secrets. It is often more efficient than a cryptanalysis mounted in the so-called *black-box model* where no leakage occurs. In particular, *continuous side-channel attacks* in which the adversary gets information at each invocation of the cryptosystem are especially threatening. Common attacks as those exploiting the running-time [19], the power consumption [20] or the electromagnetic radiations [12] of a cryptographic computation fall into this class.

Many implementations of block ciphers have been practically broken by continuous side-channel analysis — see for instance [6, 20, 22, 24] — and securing them has been a longstanding issue for the embedded systems industry. A sound approach is to use *secret sharing* [3, 32], often called *masking* in the context of side-channel attacks. This approach consists in splitting each sensitive variable of the implementation (*i.e.* variables depending on the secret key) into $d+1$ shares, where d is called the *masking order*. It has been shown that the complexity of mounting a successful side-channel attack against a masked implementation increases exponentially with the masking order [7]. Starting from this observation, the design of efficient masking schemes for different ciphers has become a foreground issue.

The DES cipher has been the focus of first designs, with the notable work of Goubin and Patarin in [14]. Further schemes have been subsequently published, in particular for the AES cipher, applying masking in hardware or software with different area-time-memory trade-offs [2, 4, 23, 25, 28, 31]. All these schemes deal with *first-order masking*, namely the intermediate variables are split in two shares (a mask and a masked variable). As a result, they only thwart *first order* side-channel attacks in which the adversary exploits the leakage of a single intermediate computation. During the last years, several works have demonstrated that this defense strategy was not sufficient for long term security purpose and that *higher-order attacks* could be successfully performed against cryptographic implementations (see *e.g.* [24]). This has raised the need for secure and efficient higher-order masking schemes.

Higher-Order Masking. The principle of higher-order masking is to split every sensitive variable x occurring during the computation into $d+1$ shares $x_0, \ldots, x_d$ in such a way that the following relation is satisfied for a group operation $\perp$:

$$x_0 \perp x_1 \perp \cdots \perp x_d = x \ . \tag{1}$$

In the rest of the paper, we shall consider that $\perp$ is the addition over some field of characteristic 2. Usually, the d shares $x_1, \ldots, x_d$ (called *the masks*) are randomly picked up and the last one x_0 (called *the masked variable*) is processed such that it satisfies (1). When d random masks are involved per sensitive variable the masking is said to be *of order* d. The tuple $(x_i)_i$ is further called a *dth-order encoding of* x.

When higher-order masking is involved to protect a block cipher implementation, a so-called *masking scheme* must be designed to enable the computation on masked data. Such a scheme must ensure that the final shares correspond to the expected ciphertext on the one hand, and it must ensure the dth-order security property for the chosen order d on the other hand. The latter property states that every tuple of d or less intermediate variables is independent of any sensitive variable. When satisfied, it guarantees that no attack of order lower than or equal to d is possible.

Most block cipher structures (*e.g.* AES or DES) are iterative, meaning that they apply several times a same transformation, called *round*, to an internal state initially filled with the plaintext. The round itself is composed of a key addition, one or several linear transformation(s) and one or several non-linear s-box(es). Key addition and linear transformations are easily handled as linearity enables to process each share independently. The main difficulty in designing masking schemes for block ciphers hence lies in masking the s-box(es).

Masking and S-Boxes. Whereas many solutions have been proposed to deal with the case of first-order masking (see *e.g.* [2, 4, 23, 27]), only a few solutions exist for the higher-order case. A scheme has been proposed by Schramm and Paar in [31] which generalizes the (first-order) table recomputation method described in [2,23]. Although the authors apply their method in the particular case of an AES implementation, it is generic and can be applied to protect any s-box. Unfortunately, this scheme has been shown to be vulnerable to a 3rd-order attack whatever the chosen masking order [8]. In other words, it only provides 2nd-order security. Further schemes were proposed by Rivain, Dottax and Prouff in [28] with formal security proofs but still limited to 2nd-order security.

The first scheme achieving dth-order security for an arbitrary chosen d has been designed by Ishai, Sahai and Wagner in [15]. The here-called *ISW scheme* consists in masking the Boolean representation of an algorithm which is composed of logical operations NOT and AND. Securing a NOT for any order d is straightforward since $x = \bigoplus_i x_i$ implies $\mathrm{NOT}(x) = \mathrm{NOT}(x_0) \oplus x_1 \cdots \oplus x_d$. The main contribution of [15] is a method to secure the AND operation for any arbitrary order d (the description of this scheme is recalled in Section 2.1). Although the ISW scheme is an important theoretical result, its practical application faces some issues. At the hardware level, the obtained circuits may have prohibitive area requirements, especially for being used in embedded systems (privileged targets of side-channel attacks). Moreover, Mangard *et al.*

have shown in [21, 22] that masking at the hardware level is sensitive to *glitches* which induce unpredicted flaws in masked circuits. Preventing glitches can be done thanks to synchronization elements (*e.g.* registers or latches) [26] or by performing additional sharing [25] but in both cases, the circuit size is still significantly increased. On the other hand, a direct application of the ISW scheme to secure an s-box computation in software would consist in taking the Boolean representation of the s-box and in processing every logical operation successively in a masked way. Since the Boolean representation of common s-boxes involves a huge number of logical operations, the resulting implementation would likely be inefficient.

In the particular case of AES, a solution has been proposed by Rivain and Prouff in [29] to efficiently mask the s-box processing at any order. Specifically, the authors use the algebraic structure of the AES s-box, which is the composition of an affine function over $\mathbb{F}_2^8$ with the power function $x \mapsto x^{254}$ over $\mathbb{F}_{256}$, and they show that it can be expressed as a sequence of operations involving a few linear functions over $\mathbb{F}_2^8$ (easy to mask) and four multiplications over $\mathbb{F}_{256}$. The latter are secured by applying the ISW scheme (generalized to $\mathbb{F}_{256}$). Subsequently, Kim, Hong and Lim have presented in [16] an extension of Rivain and Prouff's scheme, which is based on the tower-field approach from [30]. On the other hand, Genelle, Prouff and Quisquater have proposed in [13] a higher-order scheme based on the alternate use of Boolean masking and multiplicative masking. Although schemes in [16] and [13] achieve better performances than [29], they are still restricted to the AES s-box and their generalization to any s-box (or subclasses) is an open issue.

Our Contribution. The present paper introduces the first higher-order masking scheme which can be applied to efficiently protect any s-box processing in software. We first give a general method that extends the Rivain and Prouff approach to mask any s-box and we introduce a new criterion for an s-box that relates to the best efficiency achievable with our method. Then we give concrete schemes that aim to approach the so-called *masking complexity.* Specifically, we give optimal methods for the set of *power functions*, and we give efficient heuristics for the general case. As an illustration we apply our scheme to the DES and PRESENT s-boxes and we provide implementation results.

2 Higher-Order Masking of any S-Box

In this section, we describe a general method to mask any s-box and we introduce a related *masking complexity* criterion.

2.1 General Method

An s-box is a function from $\{0,1\}^n$ to $\{0,1\}^m$ with $m \leq n$ and n small (typically $n \in \{4, 6, 8\}$). We shall use the terminology of (n, m) s-box when the dimensions need to be specified. To design a higher-order masking scheme for such a function, our approach is to express it as a sequence of affine functions over $\mathbb{F}_2^n$, and multiplications over $\mathbb{F}_{2^n}$. Such a strategy is always possible since any (n, m) s-box can be represented by a polynomial function $x \mapsto \sum_{i=0}^{2^n-1} a_i x^i$ over $\mathbb{F}_{2^n}$ where the a_i are constant coefficients in $\mathbb{F}_{2^n}$. The a_i can be obtained from the s-box look-up table by applying Lagrange's Interpolation Theorem. When m is strictly lower than n, the m-bit outputs can be embedded into $\mathbb{F}_{2^n}$ by padding them to n-bit outputs (*e.g.* by setting most significant bits to 0). The padding is then removed after the polynomial evaluation. We recall hereafter the Lagrange Interpolation Theorem applied to our context.

Theorem 1 (Lagrange Interpolation). *Let* S *be a function* $\mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$*. Then, for every* $x \in \mathbb{F}_{2^n}$*, we have:*

$$\mathrm{S}(x) = \sum_{\alpha \in \mathbb{F}_{2^n}} \mathrm{S}(\alpha)\ell_\alpha(x) \ , \tag{2}$$

where, for every $\alpha \in \mathbb{F}_{2^n}$*,* ℓ_α *is defined as:*

$$\ell_\alpha(x) = \prod_{\substack{\beta \in \mathbb{F}_{2^n} \\ \beta \neq \alpha}} \frac{x - \beta}{\alpha - \beta} \ . \tag{3}$$

Remark 1. The ℓ_α are called the *Lagrange basis polynomials* and satisfy $\ell_\alpha(x) = 1$ if $x = \alpha$ and $\ell_\alpha(x) = 0$ otherwise. In particular, every ℓ_α is a monic polynomial of degree $2^n - 1$, and we have $\ell_\alpha(x) = (x + \alpha)^{2^n - 1} + 1$. Moreover, the coefficients of $\mathrm{S}(x)$ can be directly computed from the Mattson-Solomon polynomial by:

$$a_i = \begin{cases} \mathrm{S}(0) & \text{if } i = 0 \\ \sum_{k=0}^{2^n-2} \mathrm{S}(\alpha^k)\alpha^{-ki} & \text{if } 1 \leq i \leq 2^n - 2 \\ \mathrm{S}(1) + \sum_{i=0}^{2^n-2} a_i & \text{if } i = 2^n - 1 \end{cases}$$

for every primitive element α of $\mathbb{F}_{2^n}$.

The polynomial representation of an s-box is based on four kinds of operations over $\mathbb{F}_{2^n}$: additions, scalar multiplications (*i.e.* multiplications by constants), squares, and regular multiplications (*i.e.* of two different variables). Except for the latter, all these operations are $\mathbb{F}_2^n$*-linear* (or $\mathbb{F}_2^n$*-affine*), that is the corresponding function over $\mathbb{F}_2^n$ are linear (resp. affine). The processing of any s-box can then be performed as a sequence of $\mathbb{F}_2^n$-affine functions (themselves composed of additions, squares and scalar multiplications over $\mathbb{F}_{2^n}$) and of regular multiplications over $\mathbb{F}_{2^n}$, called *nonlinear multiplications* in the following. Masking an s-box processing can hence be done by masking every affine function and every nonlinear multiplication independently. We recall hereafter how this can be done for each category.

Masking of $\mathbb{F}_2^n$*-affine functions.* Let $x = \sum_i x_i$ be a shared variable. Every affine function g with additive part c_g satisfies:

$$g(x) = \begin{cases} \sum_{i=0}^{d} g(x_i) & \text{if } d \text{ is even,} \\ c_g + \sum_{i=0}^{d} g(x_i) & \text{if } d \text{ is odd.} \end{cases}$$

The masked processing of g then simply consists in evaluating g for every share x_i, and possibly correcting one of them by addition of c_g. Such a processing clearly achieves dth-order security as the shares are all processed independently.

Masking of nonlinear multiplications. Every nonlinear multiplication can be processed by using the ISW scheme. Let $a, b \in \mathbb{F}_{2^n}$ and let $(a_i)_{0 \leq i \leq d}$ and $(b_i)_{0 \leq i \leq d}$ be dth-order encoding of a and b. To securely compute a dth-order encoding $(c_i)_{0 \leq i \leq d}$ of $c = ab$, the ISW method over $\mathbb{F}_{2^n}$ performs as follows:[5]

1. For every $0 \leq i < j \leq d$, pick up a random value $r_{i,j}$ in $\mathbb{F}_{2^n}$.
2. For every $0 \leq i < j \leq d$, compute $r_{j,i} = (r_{i,j} + a_i b_j) + a_j b_i$.
3. For every $0 \leq i \leq d$, compute $c_i = a_i b_i + \sum_{j \neq i} r_{i,j}$.

[5] The use of brackets indicates the order in which the operations are performed, which is mandatory for the security of the scheme.

It can be checked that the obtained shares are a sound encoding of c. Namely, we have:

$$\sum_{i=0}^{d} c_i = \big(\sum_{i=0}^{d} a_i\big)\big(\sum_{i=0}^{d} b_i\big) = ab = c.$$

In [15] it is shown that the above computation achieves $(d/2)$th-order security. A tighter security proof is given in [29] which shows that dth-order security is actually achieved as long as the masks of the two inputs are independent. Therefore, we shall refresh the masks before a masked multiplication when necessary. This can be done using a refreshing procedure as proposed in [29] (see Algorithm 2 in appendix).

Remark 2. Another method to process a masked multiplication at an arbitrary order is used in [10] to achieve provable security under specific leakage assumptions. However this method requires more operations and more random bits than the ISW scheme does. For this reason, the ISW scheme should be preferred in a usual dth-order security model.

2.2 Masking Complexity

The scheme described in the previous section secures the computation of any (n, m) s-box S by masking its polynomial representation over $\mathbb{F}_{2^n}$. The evaluation of such a polynomial is composed of $\mathbb{F}_2^n$-affine functions g and of nonlinear multiplications. The masked processing of each $\mathbb{F}_2^n$-affine function g merely involves $d + 1$ evaluations of g itself, while it involves $(d + 1)^2$ field multiplications, $2d(d + 1)$ field additions and the generation of $nd(d + 1)/2$ random bits for each nonlinear multiplication. The masked processing of $\mathbb{F}_2^n$-affine functions hence quickly becomes negligible compared to the masked processing of nonlinear multiplications as d grows. This observation motivates the following definition of the *masking complexity* for an s-box.

Definition 1 (Masking Complexity). *Let m and n be two integers such that $m \leq n$. The* masking complexity *of a (n, m) s-box is the minimal number of nonlinear multiplications required to evaluate its polynomial representation over $\mathbb{F}_{2^n}$.*

The following proposition directly results from this definition.

Proposition 1. *The masking complexity of an s-box is invariant when composed with $\mathbb{F}_2^n$-affine bijections in input and/or in output.*

Remark 3. Since field isomorphisms are $\mathbb{F}_2$-linear bijections, the choice of the irreducible polynomial to represent field elements does not impact the masking complexity of an s-box.

In the next sections, we address the issue of finding polynomial evaluations of an s-box that aim at minimizing the number of nonlinear multiplications. Those constructions will enable us to deduce upper bounds on the masking complexity of an s-box. We first study the case of power functions whose polynomial representation has a single monomial (*e.g.* the AES s-box). For these functions, we exhibit the exact masking complexity by deriving addition chains with minimal number of nonlinear multiplications. We then address the general case and provide efficient heuristics to evaluate any s-box with a low number of nonlinear multiplications.

3 Optimal Masking of Power Functions

In this section, we consider s-boxes for which the polynomial representation over $\mathbb{F}_{2^n}$ is a single monomial. These s-boxes are usually called *power functions* in the literature. We describe a generic method to compute the masking complexity of such s-boxes. Our method involves the notion of *cyclotomic class*.

Definition 2. *Let $\alpha \in [0; 2^n - 2]$. The* cyclotomic class *of α is the set C_α defined by:*

$$C_\alpha = \{\alpha \cdot 2^i \bmod 2^n - 1;\ i \in [0; n-1]\}.$$

We have the following proposition.

Proposition 2. *Let $\mu(m)$ denote the multiplicative order of 2 modulo m and let φ denote the Euler's totient function. For every divisor δ of $2^n - 1$, the number of distinct cyclotomic classes $C_\alpha \subseteq [0; 2^n - 2]$ with $\gcd(\alpha, 2^n - 1) = \delta$ is $\varphi\big(\frac{2^n-1}{\delta}\big)/\mu\left(\frac{2^n-1}{\delta}\right)$. It follows that the total number of distinct cyclotomic classes of $[0; 2^n - 2]$ equals:*

$$\sum_{\delta|(2^n-1)} \frac{\varphi(\delta)}{\mu(\delta)} \ .$$

Proof. Proposition 2 can be deduced from the following facts:

- An integer $\alpha \in [0; 2^n-2]$ satisfies $\gcd(\alpha, 2^n-1) = \delta$ if and only if $\alpha = \delta\beta$, with $\gcd(\beta, \frac{2^n-1}{\delta}) = 1$. There are thus $\varphi\big(\frac{2^n-1}{\delta}\big)$ integers $\alpha \in [0; 2^n - 2]$ such that $\gcd(\alpha, 2^n - 1) = \delta$.
- For any α such that $\gcd(\alpha, 2^n - 1) = \delta$ (hence of the form $\alpha = \delta\beta$ with $\gcd(\beta, \frac{2^n-1}{\delta}) = 1$), we have $\alpha \cdot 2^i \equiv \alpha \cdot 2^j \bmod 2^n - 1$ if and only if $\beta \cdot 2^i \equiv \beta \cdot 2^j \bmod \frac{2^n-1}{\delta}$, that is, if and only if $2^i \equiv 2^j \bmod \frac{2^n-1}{\delta}$. Hence C_α has cardinality $\#C_\alpha = \mu\big(\frac{2^n-1}{\delta}\big)$.

The set of integers $\alpha \in [0; 2^n - 2]$ such that $\gcd(\alpha, 2^n - 1) = \delta$ is partitioned into cyclotomic classes, each of them having cardinality $\mu\big(\frac{2^n-1}{\delta}\big)$. Hence the number of such cyclotomic classes is $\varphi\big(\frac{2^n-1}{\delta}\big)/\mu\left(\frac{2^n-1}{\delta}\right)$. It follows that the total number of distinct cyclotomic classes of $[0; 2^n - 2]$ equals $\sum_{\delta|(2^n-1)} \varphi\big(\frac{2^n-1}{\delta}\big)/\mu\left(\frac{2^n-1}{\delta}\right) = \sum_{\delta|(2^n-1)} \varphi(\delta)/\mu(\delta)$.

□

The study of cyclotomic classes is interesting in our context since a power x^α can be computed from a power x^β without any nonlinear multiplication if and only if α and β lie in the same cyclotomic class. Hence, all the power functions with exponents within a given cyclotomic class have the same masking complexity and computing the masking complexity for all the power functions over $\mathbb{F}_{2^n}$ thus amounts to compute this complexity for each cyclotomic class over $\mathbb{F}_{2^n}$. In what follows, we perform such a computation for fields $\mathbb{F}_{2^n}$ of small dimensions n.

To compute the masking complexity for an element in a cyclotomic class, we use the following observation: determining the masking complexity of a power function $x \mapsto x^\alpha$ amounts to find the addition chain for α with the least number of additions which are not doublings (see [17] for an introduction to addition chains). This kind of addition chain is usually called a 2-*addition chain*.[6] Let $(\alpha_i)_i$ denote some addition chain. At step i, it is possible to obtain any element within the cyclotomic classes $(C_{\alpha_j})_{j \le i}$ using doublings only. As we are interested in finding the addition chain with the least number of additions which are not doublings, the problem we need to solve is the following: given some $\alpha \in C_\alpha$, find the shortest chain $C_{\alpha_0} \to C_{\alpha_1} \to \cdots \to C_{\alpha_k}$ where $C_{\alpha_0} = C_1$, $C_{\alpha_k} = C_\alpha$ and for every $i \in [1; k]$, there exists $j, \ell < i$ such that $\alpha_i = \alpha'_j + \alpha'_\ell$ where $\alpha'_j \in C_{\alpha_j}$ and $\alpha'_\ell \in C_{\alpha_\ell}$.

We shall denote by $\mathcal{M}^n_k$ the class of exponents α such that $x \mapsto x^\alpha$ has a masking complexity equal to k. The family of classes $(\mathcal{M}^n_k)_k$ is a partition of $[0; 2^n - 2]$ and each $\mathcal{M}^n_k$ is the union of one or several cyclotomic classes. For a small dimension n, we can proceed by exhaustive search

[6] This problem has been studied in the general setting where the multiplication by q (and not specifically by 2) is considered *free* and the obtained addition chains are called *q-addition chains* [33]. The purpose is to find efficient exponentiation methods in $\mathbb{F}_q$ (as in such field the Frobenius map $x \mapsto x^q$ is efficient). To the best of our knowledge, apart from a specific application to the SFLASH signature algorithm in [1], the case of 2-addition chains has not been particularly investigated.

to determine the shortest 2-addition chain(s) for each cyclotomic class. We implemented such an exhaustive search from which we obtained the masking complexity classes $\mathcal{M}_k^n$ for $n \leq 11$ (note that in practice most s-boxes have dimension $n \leq 8$). Table 1 summarizes the obtained results for $n \in \{4, 6, 8\}$ (usual dimensions). Results for other dimensions are summarized in Appendix A. Additionally, Table 2 gives the optimal 2-addition chains (in exponential notation) corresponding to every cyclotomic class for $n = 8$.

Table 1. Cyclotomic classes for $n \in \{4, 6, 8\}$ w.r.t. the masking complexity k.

k	Cyclotomic classes in $\mathcal{M}_k^n$
$n = 4$	
0	$C_0 = \{0\}$, $C_1 = \{1, 2, 4, 8\}$
1	$C_3 = \{3, 6, 12, 9\}$, $C_5 = \{5, 10\}$
2	$C_7 = \{7, 14, 13, 11\}$
$n = 6$	
0	$C_0 = \{0\}$, $C_1 = \{1, 2, 4, 8, 16, 32\}$
1	$C_3 = \{3, 6, 12, 24, 48, 33\}$, $C_5 = \{5, 10, 20, 40, 17, 34\}$, $C_9 = \{9, 18, 36\}$
2	$C_7 = \{7, 14, 28, 56, 49, 35\}$, $C_{11} = \{11, 22, 44, 25, 50, 37\}$, $C_{13} = \{13, 26, 52, 41, 19, 38\}$, $C_{15} = \{15, 30, 29, 27, 23\}$, $C_{21} = \{21, 42\}$, $C_{27} = \{27, 54, 45\}$
3	$C_{23} = \{23, 46, 29, 58, 53, 43\}$, $C_{31} = \{31, 62, 61, 59, 55, 47\}$
$n = 8$	
0	$C_0 = \{0\}$, $C_1 = \{1, 2, 4, 8, 16, 32, 64, 128\}$
1	$C_3 = \{3, 6, 12, 24, 48, 96, 192, 129\}$, $C_5 = \{5, 10, 20, 40, 80, 160, 65, 130\}$, $C_9 = \{9, 18, 36, 72, 144, 33, 66, 132\}$, $C_{17} = \{17, 34, 68, 136\}$
2	$C_7 = \{7, 14, 28, 56, 112, 224, 193, 131\}$, $C_{11} = \{11, 22, 44, 88, 176, 97, 194, 133\}$, $C_{13} = \{13, 26, 52, 104, 208, 161, 67, 134\}$, $C_{15} = \{15, 30, 60, 120, 240, 225, 195, 135\}$, $C_{19} = \{19, 38, 76, 152, 49, 98, 196, 137\}$, $C_{21} = \{21, 42, 84, 168, 81, 162, 69, 138\}$, $C_{25} = \{25, 50, 100, 200, 145, 35, 70, 140\}$, $C_{27} = \{27, 54, 108, 216, 177, 99, 198, 141\}$, $C_{37} = \{37, 74, 148, 41, 82, 164, 73, 146\}$, $C_{45} = \{45, 90, 180, 105, 210, 165, 75, 150\}$, $C_{51} = \{51, 102, 204, 153\}$, $C_{85} = \{85, 170\}$
3	$C_{23} = \{23, 46, 92, 184, 113, 226, 197, 139\}$, $C_{29} = \{29, 58, 116, 232, 209, 163, 71, 142\}$, $C_{31} = \{31, 62, 124, 248, 241, 227, 199, 143\}$, $C_{39} = \{39, 78, 156, 57, 114, 228, 201, 147\}$, $C_{43} = \{43, 86, 172, 89, 178, 101, 202, 149\}$, $C_{47} = \{47, 94, 188, 121, 242, 229, 203, 151\}$, $C_{53} = \{53, 106, 212, 169, 83, 166, 77, 154\}$, $C_{55} = \{55, 110, 220, 185, 115, 230, 205, 155\}$, $C_{59} = \{59, 118, 236, 217, 179, 103, 206, 157\}$, $C_{61} = \{61, 122, 244, 233, 211, 167, 79, 158\}$, $C_{63} = \{63, 126, 252, 249, 243, 231, 207, 159\}$, $C_{87} = \{87, 174, 93, 186, 117, 234, 213, 171\}$, $C_{91} = \{91, 182, 109, 218, 181, 107, 214, 173\}$, $C_{95} = \{95, 190, 125, 250, 245, 235, 215, 175\}$, $C_{111} = \{111, 222, 189, 123, 246, 237, 219, 183\}$, $C_{119} = \{119, 238, 221, 187\}$
4	$C_{127} = \{127, 254, 253, 251, 247, 239, 223, 191\}$

It is interesting to note that for every n, the *inverse function* $x \mapsto x^{2^n-2}$ related to the cyclotomic class $C_{2^{n-1}-1}$ always has the highest masking complexity. In particular, the inverse function $x \mapsto x^{254}$ (for $n = 8$) used in the AES has a masking complexity of 4 as it was conjectured in [29].

4 Efficient Heuristics for General S-Boxes

We now address the general case of an s-box having a polynomial representation $\sum_{j=0}^{2^n-1} a_j x^j$ over $\mathbb{F}_{2^n}$. A straightforward solution is to successively compute every power x^j using $x^j = (x^{j/2})^2$ if j is even and $x^j = x^{j-1}x$ if j is odd, while updating the polynomial value by adding the monomial $a_j x^j$ at every step. Such a method requires $2^{n-1} - 1$ nonlinear multiplications. As

Table 2. Optimal 2-addition chains (in exponential notation) for cyclotomic classes for $n = 8$.

k	2-addition chains with k nonlinear multiplications
1	$x^3 \leftarrow x \times x^2$ – $x^5 \leftarrow x \times x^4$ $x^9 \leftarrow x \times x^8$ – $x^{17} \leftarrow x \times x^{16}$
2	$x^7 \leftarrow x \times x^2 \times x^4$ – $x^{11} \leftarrow x \times x^2 \times x^8$ $x^{13} \leftarrow x \times x^4 \times x^8$ – $x^{15} \leftarrow x^3 \times (x^3)^4$ $x^{19} \leftarrow x \times x^2 \times x^{16}$ – $x^{21} \leftarrow x \times x^4 \times x^{16}$ $x^{27} \leftarrow x^3 \times (x^3)^8$ – $x^{37} \leftarrow x \times x^4 \times x^{32}$ $x^{45} \leftarrow x^5 \times (x^5)^8$ – $x^{51} \leftarrow x^3 \times (x^3)^{16}$ $x^{85} \leftarrow x^5 \times (x^5)^{16}$
3	$x^{23} \leftarrow x \times x^2 \times x^4 \times x^{16}$ – $x^{29} \leftarrow x \times x^4 \times x^8 \times x^{16}$ $x^{31} \leftarrow x^3 \times (x^3)^4 \times x^{16}$ – $x^{29} \leftarrow x \times x^2 \times x^4 \times x^{32}$ $x^{43} \leftarrow x \times x^2 \times x^8 \times x^{32}$ – $x^{47} \leftarrow x^3 \times (x^3)^4 \times x^{32}$ $x^{53} \leftarrow x \times x^2 \times x^{16} \times x^{32}$ – $x^{55} \leftarrow x^3 \times x^4 \times (x^3)^{16}$ $x^{59} \leftarrow x^3 \times (x^3)^8 \times x^{32}$ – $x^{59} \leftarrow x^5 \times x^{16} \times (x^5)^8$ $x^{63} \leftarrow x^7 \times (x^7)^8$ – $x^{87} \leftarrow x^2 \times x^5 \times (x^5)^{16}$ $x^{91} \leftarrow x^3 \times (x^3)^8 \times x^{64}$ – $x^{95} \leftarrow x^5 \times (x^5)^2 \times (x^5)^{16}$ $x^{111} \leftarrow x^3 \times (x^3)^4 \times (x^3)^{32}$ – $x^{63} \leftarrow x^7 \times (x^7)^{16}$
4	$x^{127} \leftarrow x^3 \times (x^3)^4 \times (x^3)^{16} \times x^{64}$

we show hereafter, less naive methods exist that substantially lower the number of nonlinear multiplications. We propose two different methods and then compare their efficiency.

4.1 Cyclotomic Method

Let q denote the number of distinct cyclotomic classes of $[0; 2^n - 2]$. The polynomial representation of S can be written as:

$$\mathrm{S}(x) = a_0 + \Big(\sum_{i=1}^{q} Q_i(x)\Big) + a_{2^n-1}\, x^{2^n-1} \ ,$$

where the Q_i are polynomials such that every Q_i has powers from a single cyclotomic class C_{α_i}, namely we can write $Q_i(x) = \sum_j a_{i,j} x^{\alpha_i 2^j}$ for some coefficients $a_{i,j}$ in $\mathbb{F}_{2^n}$. Let us then denote L_i the linearized polynomial $L_i(x) = \sum_j a_{i,j} x^{2^j}$ which is a $\mathbb{F}_2^n$-linear function of x. We have $Q_i(x) = L_i(x^{\alpha_i})$ by definition. The *cyclotomic method* simply consists in deriving the powers x^{α_i} for each cyclotomic class C_{α_i} as well as x^{2^n-1} if $a_{2^n-1} \neq 0$, and in evaluating $\mathrm{S}(x) = a_0 + \Big(\sum_{i=1}^{q} L_i(x^{\alpha_i})\Big) + a_{2^n-1}\, x^{2^n-1}$. The powers x^{α_i} can each be derived with a single nonlinear multiplication. This is obvious for the α_i lying in $\mathcal{M}_1^n$. Then it is clear that every power x^{α_i} with $\alpha_i \in \mathcal{M}_{k+1}^n$ can be derived with a single multiplication from the powers $(x^{\alpha_i})_{\alpha_i \in \mathcal{M}_k^n}$. The power x^{2^n-1} can then be derived with a single nonlinear multiplication from the power x^{2^n-2}. The cyclotomic method hence involves a number of nonlinear multiplications equal to the number of cyclotomic classes, minus 2 (as x^0 and x^1 are obtained without nonlinear multiplication), plus 1 (to derive x^{2^n-1}). By Proposition 2, we then have the following result.

Proposition 3 (Cyclotomic Method). *Let m and n be two positive integers such that $m \leq n$. The masking complexity of every (n, m) s-box is upper-bounded by:*

$$\sum_{\delta | (2^n-1)} \frac{\varphi(\delta)}{\mu(\delta)} - 1 \ .$$

An (n,m) s-box S is said to be *balanced* if for every $y \in \{0,1\}^m$, the number of preimages of y for S is constant to 2^{n-m}. The following lemma gives a well-known folklore result.

Lemma 1. *Let m and n be two positive integers such that $m \leq n$. The polynomial representation of every balanced (n,m) s-box has degree strictly lower than $2^n - 1$.*

Proof. Since Lagrange basis polynomials are all monic of degree $2^n - 1$, the coefficient a of the power to the $2^n - 1$ in the polynomial representation of S satisfies $a = \sum_{\alpha \in \mathbb{F}_{2^n}} \mathrm{S}(\alpha)$, which equals 0 if S is balanced. □

When the polynomial representation of the s-box has degree strictly lower than $2^n - 1$, the cyclotomic method saves one nonlinear multiplication since the power x^{2^n-1} is not required. Namely, we have the following corollary of Proposition 3.

Corollary 1 (Cyclotomic Method). *Let m and n be two positive integers such that $m \leq n$ and let S be a (n,m) s-box. If S is balanced, then the masking complexity of S is upper-bounded by:*

$$\sum_{\delta | (2^n-1)} \frac{\varphi(\delta)}{\mu(\delta)} - 2 \ .$$

4.2 Parity-Split Method

The *parity-split method* is composed of two stages. The first stage derives a set of powers $(x^j)_{j \leq q}$ for some q using the straightforward method described in the introduction of this section. The second stage essentially consists in an application of the Knuth-Eve polynomial evaluation algorithm [9,18] which is based on a recursive use of the following lemma.

Lemma 2. *Let n and t be two positive integers and let Q be a polynomial of degree t over $\mathbb{F}_{2^n}[x]$. There exist two polynomials Q_1 and Q_2 of degree upper-bounded by $\lfloor t/2 \rfloor$ over $\mathbb{F}_{2^n}[x]$ such that:*

$$Q(x) = Q_1(x^2) + Q_2(x^2)x \ . \tag{4}$$

By applying Lemma 2 to the polynomial representation of S, we get $\mathrm{S}(x) = Q_1(x^2) + Q_2(x^2)x$, where Q_1 and Q_2 are two polynomials of degrees upper-bounded by $2^{n-1} - 1$. We deduce that S can be computed based on the set of powers $(x^{2j})_{j \leq 2^{n-1}-1}$ plus a single multiplication by x. Then, applying Lemma 2 again to the polynomials Q_1 and Q_2 both of degrees upper bounded by $2^{n-1} - 1$, we get two new pairs of polynomials (Q_{11}, Q_{12}) and (Q_{21}, Q_{22}) such that $Q_1(x^2) = Q_{11}(x^4) + Q_{12}(x^4)x^2$ and $Q_2(x^2) = Q_{21}(x^4) + Q_{22}(x^4)x^2$. The degrees of the new polynomials are upper bounded by $2^{n-2} - 1$. We then deduce that S can be computed based on the set of powers $(x^{4j})_{j \leq 2^{n-2}-1}$ plus 1 multiplication by x and 2 multiplications by x^2. Eventually, by applying Lemma 2 recursively r times, we get an evaluation of S involving evaluations in x^{2^r} of polynomials of degrees upper-bounded by $2^{n-r} - 1$, plus $\sum_{i=0}^{r-1} 2^i = 2^r - 1$ multiplications by powers of x of the form x^{2^i} with $i \leq r - 1$. The overall evaluation of S hence requires $2^r - 1$ nonlinear multiplications (the x^{2^i} being obtained with squares only) plus the evaluation in x^{2^r} of polynomials of degrees upper-bounded by $2^{n-r} - 1$. The latter evaluation can be performed by first deriving all the powers $(x^{2^r j})_{j \leq 2^{n-r}-1}$ and then evaluating the polynomials (which only involves scalar multiplications and additions once the powers have been derived). For every $j \leq 2^{n-r} - 1$, the powers $(x^{2^r j})_{j \leq 2^{n-r}-1}$ can be computed successively from $y = x^{2^r}$ by $y^j = (y^{j/2})^2$ if j is even and $y^j = y^{j-1}x$ if j is odd. This takes some squares plus $2^{n-r-1} - 1$ nonlinear multiplications (*i.e.* one per odd integer in $[3, 2^{n-r} - 1]$).

We then deduce the following proposition.

Proposition 4. *Let m and n be two positive integers such that $m \leq n$. The masking complexity of every (n, m) s-box is upper-bounded by:*

$$\min_{0 \leq r \leq n} (2^{n-r-1} + 2^r) - 2 = \begin{cases} 3 \cdot 2^{(n/2)-1} - 2 & \textit{if } n \textit{ is even,} \\ 2^{(n+1)/2} - 2 & \textit{if } n \textit{ is odd.} \end{cases} \tag{5}$$

Note that the value of r for which the minimum is reached in (5) is $r = \lfloor \frac{n}{2} \rfloor$.

4.3 Comparison

Table 3 summarizes the number of nonlinear multiplications obtained by the cyclotomic method (for balanced s-boxes) and by the parity-split method. We see that the cyclotomic method works better for small dimensions ($n \leq 5$) and the parity-split method for higher dimensions ($n \geq 6$). Furthermore, the superiority of the parity-split method becomes significant as n grows.

Table 3. Number of nonlinear multiplications w.r.t. the evaluation method.

Method $\backslash$ n	3	4	5	6	7	8	9	10	11
Cyclotomic	1	3	5	11	17	33	53	105	192
Parity-Split	2	4	6	10	14	22	30	46	62

We emphasize that these bounds may not be optimal, namely they may be higher than the maximum masking complexity of (n, m) s-boxes. We let open the issue of finding more efficient (or provably optimal) methods in the general case for further research.

5 Application to DES and PRESENT

In this section we apply the proposed methods to the s-boxes of two different block ciphers: the well-known and still widely used Data Encryption Standard (DES) [11], and the lightweight block cipher PRESENT [5]. The former uses eight different $(6, 4)$ s-boxes and the latter uses a single $(4, 4)$ s-box. According to Table 3, we shall prefer the parity-split method for the DES s-boxes (10 nonlinear multiplications), and the cyclotomic method for the PRESENT s-box (3 nonlinear multiplications).

5.1 Parity-Split Method on DES S-boxes

The parity-split method on a DES s-box uses a polynomial representation of the s-box over $\mathbb{F}_{64}$ which satisfies:

$$\begin{aligned} S \,:\, x \,\longmapsto\, & Q_0(x^8) + Q_1(x^8) \cdot x^4 + \big(Q_2(x^8) + Q_3(x^8) \cdot x^4\big) \cdot x^2 \\ & + \Big(Q_4(x^8) + Q_5(x^8) \cdot x^4 + \big(Q_6(x^8) + Q_7(x^8) \cdot x^4\big) \cdot x^2\Big) \cdot x \end{aligned} \tag{6}$$

where the Q_i are degree-7 polynomials, namely, there exist coefficients $a_{i,j}$ for $0 \leq i, j \leq 7$ such that:

$$Q_i(x^8) = a_{i,0} + a_{i,1}x^8 + a_{i,2}x^{16} + a_{i,3}x^{24} + a_{i,4}x^{32} + a_{i,5}x^{40} + a_{i,6}x^{48} + a_{i,7}x^{56} \ .$$

We first derive the powers x^{8j} for $j = 1, 2, \ldots, 7$, which is done at the cost of 3 nonlinear multiplications by:

$$x^8 \leftarrow ((x^2)^2)^2;\ x^{16} \leftarrow (x^8)^2;\ x^{24} \leftarrow x^8 \cdot x^{16};\ x^{32} \leftarrow (x^{16})^2;$$
$$x^{40} \leftarrow x^8 \cdot x^{32};\ x^{48} \leftarrow (x^{24})^2;\ x^{56} \leftarrow x^8 \cdot x^{48};$$

Then we evaluate each polynomial $Q_i(x^8)$ as a linear combination of the above powers. Finally, we evaluate (6) at the cost of 7 nonlinear multiplications and a few additions. The nonlinear multiplications are computed using the ISW scheme over $\mathbb{F}_{64}$ such as recalled in Section 2.1. A detailed algorithm for the overall masked s-box evaluation is given in Appendix B. Moreover the log/alog tables for the multiplication over $\mathbb{F}_{64}$ and for the $a_{i,j}$ coefficients for the first DES s-box are given in Appendix C.

5.2 Cyclotomic Method on PRESENT S-box

The cyclotomic method on the PRESENT s-box starts from the straightforward polynomial representation of the s-box over $\mathbb{F}_{16}$:

$$\mathrm{S} \;:\; x \;\longmapsto\; a_0 + a_1 x + a_2 x^2 + \cdots + a_{14} x^{14} \;,$$

(where the degree is indeed strictly lower than 15 by Lemma 1). We then have:

$$\mathrm{S}(x) = a_0 + L_1(x) + L_3(x^3) + L_5(x^5) + L_7(x^7) \;. \tag{7}$$

where:

$$\begin{aligned} L_1 : x \;&\mapsto\; a_1 x + a_2 x^2 + a_4 x^4 + a_8 x^8 \\ L_3 : x \;&\mapsto\; a_3 x + a_6 x^2 + a_{12} x^4 + a_9 x^8 \\ L_5 : x \;&\mapsto\; a_5 x + a_{10} x^2 \\ L_7 : x \;&\mapsto\; a_7 x + a_{14} x^2 + a_{13} x^4 + a_{11} x^8 \end{aligned}$$

and the L_i are $\mathbb{F}_2^4$-linear.

We first derive the powers x^3, x^5, and x^7, which is done at the cost of 3 nonlinear multiplications by: $x^3 \leftarrow x \cdot x^2$; $x^5 \leftarrow x^3 \cdot x^2$; $x^7 \leftarrow x^5 \cdot x^2$. Then we evaluate (7) which costs a few linear transformations and additions. A detailed algorithm for the overall masked s-box evaluation is given in Appendix B. Moreover the look-up tables for the multiplication over $\mathbb{F}_{16}$ and for the L_i transformations are given in Appendix C.

5.3 Implementation Results

In this section, we give implementation results for our scheme applied to DES and PRESENT s-boxes. For comparison, we also give performances of some higher-order masking schemes for the AES s-box, as well as performances of existing schemes for DES and PRESENT s-boxes at orders 1 and 2. For the AES s-box processing, we implemented Rivain and Prouff's method [29] and its improvement by Kim *et al.* [16]. We did not implement Genelle *et al.* 's scheme [13] since it addresses the masking of an overall AES and is not interesting while focusing on a single s-box processing. Regarding existing schemes for DES and PRESENT s-boxes, we implemented the generic methods proposed in [27] (for $d = 1$) and in [28] (for $d = 2$). We also implemented the improvement of these schemes described in [28, §3.3] that consists in treating two 4-bit outputs at the same time.[7] Note that we did not implement the table re-computation method (for $d = 1$) since it only makes sense for an overall cipher and not for a single s-box processing.

Table 4 lists the timing/memory performances of the different implementations. We wrote the codes in assembly language for an `8051` based 8-bit architecture with bit-addressable memory. `ROM` consumptions (*i.e.* code sizes) are not listed since they are not prohibitive.

[7] This improvement is only described in [28] for $d = 2$ but it can be applied likewise to the 1st-order scheme of [27].

Table 4. Comparison of secure s-box implementations

	Method	Reference	cycles	RAM (bytes)
		First Order Masking		
1.	AES s-box	[29]	533	10
2.	AES s-box	[16]	320	14
3.	DES s-box	Simple version [27]	1096	2
4.	DES s-box	Improved version [27] & [28]	439	14
5.	DES s-box	this paper	4100	50
6.	PRESENT s-box	Simple Version [27]	281	2
7.	PRESENT s-box	Improved Version [27] & [28]	231	14
4.	PRESENT s-box	this paper	220	18
		Second Order Masking		
1.	AES s-box	[29]	832	18
2.	AES s-box	[16]	594	24
3.	DES s-box	Simple version [28]	1045	69
4.	DES s-box	Improved version [28]	652	39
5.	DES s-box	this paper	7000	78
6.	PRESENT s-box	Simple Version [28]	277	21
7.	PRESENT s-box	Improved Version [28]	284	15
8.	PRESENT s-box	this paper	400	31
		Third Order Masking		
1.	AES s-box	[29]	1905	28
2.	AES s-box	[16]	965	38
3.	DES s-box	this paper	10500	108
4.	PRESENT s-box	this paper	630	44

As expected, the cyclotomic method is very efficient when applied to protect the PRESENT s-box. The small input dimension of the s-box indeed implies a low masking complexity (equal to 3). Moreover, it enables to tabulate the multiplication over $\mathbb{F}_{16}$. At first order, it is even slightly better than the method in [27] (or its improvement). At second order, the cost of the secure multiplications involved in the cyclotomic method is approximatively doubled, which explains that the overall cost is multiplied by 1.8. This makes it less efficient than [27] and [28], which are less impacted by the increase of the masking order from 1 to 2. At third order, our method is the only one. The number of cycles staying small (630), Table 4 shows that achieving resistance against 3rd-order side-channel analysis is realistic for an implementation of PRESENT on a 8051 architecture. For DES s-boxes, the parity-split method is less efficient than the state-of-the art methods for $d = 1, 2$. This is an expected consequence of the high number of nonlinear multiplications (here 10) achieved with the parity-split method in dimension 6 and of the fact that the field multiplications can no longer be tabulated (and must therefore be computed thanks to log/alog look-up tables). At third order, the timing efficiency of the method becomes very low. The masked s-box processing is 5 (resp. 10) times slower than the efficiency of the AES s-box protected thanks to [16] (resp. [29]), though its input dimension is smaller.

The ranking of the timing efficiencies for AES, DES and PRESENT s-boxes is correlated to the number of nonlinear multiplications in the used scheme (3, 4-5, and 10, for PRESENT, AES and DES respectively) which underline the soundness of the masking complexity criterion. Therefore, while selecting an s-box for a block cipher design, one should favor an s-box with small masking complexity if side-channel attacks are taken into account.

6 Discussion

In previous sections we have introduced the first schemes that can be used to mask any s-box at any order with fair performances in software. In particular, these schemes enable to apply higher-order masking on random s-boxes (*e.g.* the DES s-boxes) which have no specific mathematical structure. Prior to our work, the only existing methods were the circuit-oriented proposals of Ishai *et al.* [15] and of Faust *et al.* [10]. The main purpose of these works was a proof of concept for applying higher-order masking to circuits with formal security proofs, but they did not address efficient implementation. A direct application of [15] or [10] to a block cipher consists in taking its Boolean representation and in replacing every XOR and AND with $O(d)$ and $O(d^2)$ logical operations respectively (where d is the masking order). Applying such a strategy in software leads to inefficient implementation as the Boolean representation of an s-box includes a huge number of nonlinear gates (with a $O(d^2)$ factor to be protected). Compared to these techniques, our schemes achieve significant improvements. These are obtained by starting from the field representation of the s-box and applying methods to significantly reduce the number of nonlinear multiplications compared to the Boolean representation of the s-box. For instance, we have shown that a DES s-box can be computed with 10 nonlinear multiplications whereas its Boolean representation involves several dozens of logical AND operations.

We believe that our work opens up new avenues for research in block cipher implementations and side-channel security. In particular, the issue of designing s-boxes with low masking complexity and good cryptographic criteria is still to be investigated. On the other hand, our work could be extended to take into account more general definitions of the masking complexity. Indeed Definition 1 is software oriented and hence does not encompass the hardware case. As discussed above, the complexity of masking in hardware merely depends on the number of nonlinear gates [10, 15], that is on the number of nonlinear multiplications in the (n-variate) s-box representation over $\mathbb{F}_2$, the so-called *algebraic normal form*. One may also want to minimize the number of nonlinear multiplications in the (ℓ-variate) s-box representation over $\mathbb{F}_{2^k}$ for some k (and $\ell = \lceil n/k \rceil$). This approach has actually already been followed in [16], where Kim *et al.* speeds up the scheme in [29] by using the fact that the AES s-box can be processed with 5 nonlinear multiplications over $\mathbb{F}_{16}$ rather than 4 nonlinear multiplications over $\mathbb{F}_{256}$. Although requiring an additional nonlinear multiplication, the resulting implementation is faster since multiplications over $\mathbb{F}_{16}$ can be tabulated while multiplications over $\mathbb{F}_{256}$ are computed based on the slower log/alog approach. These observations motivate the following — more general — definition of the masking complexity.

Definition 3 (Masking Complexity). *Let m, n and k be three integers such that $m, k \leq n$. The* masking complexity *of a (n, m) s-box over $\mathbb{F}_{2^k}$ is the minimal number of nonlinear multiplications required to evaluate its polynomial representation over $\mathbb{F}_{2^k}$.*

Here again, the masking complexity is independent of the representation of $\mathbb{F}_{2^k}$ since one can go from one representation to another without any nonlinear multiplication. The issue of finding efficient methods with respect to the masking complexity over a smaller field $\mathbb{F}_{2^k}$ is left open for further researches.

7 Conclusion

In this paper we have introduced new generic higher-order masking schemes for s-boxes with efficient software implementation. Specifically, we have extended the Rivain and Prouff's approach for the AES s-box to any s-box. The method consists in masking the polynomial representation of the s-box over $\mathbb{F}_{2^n}$ where n is the input dimension. As argued, the complexity of this method mainly depends on the number of nonlinear multiplications involved in the polynomial representation (*i.e.* multiplications which are not squares nor scalar multiplications). We have then introduced the masking complexity parameter for an s-box as the minimal number of nonlinear multiplications required for its evaluation. We have provided the exact values of this parameter for the set of power functions and upper bounds for all s-boxes. Namely, we have presented optimal methods to mask power functions and efficient heuristics for the general case. Eventually we have applied our schemes to the DES s-boxes and to the PRESENT s-box and we have provided implementation results. Our work stresses interesting open issues for further research. Among them the design of s-boxes taking into account the masking complexity criterion and the extension of our approach to masking over $\mathbb{F}_{2^k}$ with $k < n$ (*e.g.* for efficient hardware implementations) are of particular interest.

References

1. M.-L. Akkar, N. Courtois, R. Duteuil, and L. Goubin. A Fast and Secure Implementation of Sflash. In Y. Desmedt, editor, *Public Key Cryptography – PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 267–278. Springer, 2003.
2. M.-L. Akkar and C. Giraud. An Implementation of DES and AES, Secure against Some Attacks. In Ç. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 2001.
3. G. Blakley. Safeguarding cryptographic keys. In *National Comp. Conf.*, volume 48, pages 313–317, New York, June 1979. AFIPS Press.
4. J. Blömer, J. G. Merchan, and V. Krummel. Provably Secure Masking of AES. In M. Matsui and R. Zuccherato, editors, *Selected Areas in Cryptography – SAC 2004*, volume 3357 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2004.
5. A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.
6. E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.
7. S. Chari, C. Jutla, J. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In M. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
8. J.-S. Coron, E. Prouff, and M. Rivain. Side Channel Cryptanalysis of a Higher Order Masking Scheme. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 28–44. Springer, 2007.
9. J. Eve. The evaluation of polynomials. *Comm. ACM*, 6(1):17–21, 1964.
10. S. Faust, T. Rabin, L. Reyzin, E. Tromer, and V. Vaikuntanathan. Protecting Circuits from Leakage: the Computationally-Bounded and Noisy Cases. In H. Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 135–156. Springer, 2010.
11. FIPS PUB 46. *The Data Encryption Standard.* National Bureau of Standards, Jan. 1977.
12. K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic Analysis: Concrete Results. In Ç. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.
13. L. Genelle, E. Prouff, and M. Quisquater. Thwarting Higher-Order Side Channel Analysis with Additive and Multiplicative Maskings. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems, 13th International Workshop – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 240–255. Springer, 2011.
14. L. Goubin and J. Patarin. DES and Differential Power Analysis – The Duplication Method. In Ç. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.
15. Y. Ishai, A. Sahai, and D. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In D. Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.
16. H. Kim, S. Hong, and J. Lim. A Fast and Provably Secure Higher-Order Masking of AES S-Box. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems, 13th International Workshop – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 95–107. Springer, 2011.

17. D. Knuth. *The Art of Computer Programming*, volume 2. Addison Wesley, third edition, 1988.
18. D. E. Knuth. Evaluation of polynomials by computers. *Comm. ACM*, 5(12):595–599, 1962.
19. P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
20. P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
21. S. Mangard, T. Popp, and B. M. Gammel. Side-Channel Leakage of Masked CMOS Gates. In A. Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.
22. S. Mangard, N. Pramstaller, and E. Oswald. Successfully Attacking Masked AES Hardware Implementations. In J. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.
23. T. Messerges. Securing the AES Finalists against Power Analysis Attacks. In B. Schneier, editor, *Fast Software Encryption – FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 2000.
24. T. Messerges. Using Second-order Power Analysis to Attack DPA Resistant Software. In Ç. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 2000.
25. S. Nikova, V. Rijmen, and M. Schläffer. Secure Hardware Implementation of Non-linear Functions in the Presence of Glitches. In P. J. Lee and J. H. Cheon, editors, *Information Security and Cryptology – ICISC 2008*, volume 5461 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2008.
26. T. Popp, M. Kirschbaum, T. Zefferer, and S. Mangard. Evaluation of the Masked Logic Style MDPL on a Prototype Chip. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2007.
27. E. Prouff and M. Rivain. A Generic Method for Secure SBox Implementation. In S. Kim, M. Yung, and H.-W. Lee, editors, *Information Security Applications – WISA 2007*, volume 4867 of *Lecture Notes in Computer Science*, pages 227–244. Springer, 2008.
28. M. Rivain, E. Dottax, and E. Prouff. Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis. In T. Baignères and S. Vaudenay, editors, *Fast Software Encryption – FSE 2008*, Lecture Notes in Computer Science, pages 127–143. Springer, 2008.
29. M. Rivain and E. Prouff. Provably Secure Higher-Order Masking of AES. In S. Mangard and F.-X. Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010.
30. A. Satoh, S. Morioka, K. Takano, and S. Munetoh. A Compact Rijndael Hardware Architecture with S-Box Optimization. In E. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 239–254. Springer, 2001.
31. K. Schramm and C. Paar. Higher Order Masking of the AES. In D. Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2006.
32. A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, Nov. 1979.
33. J. von zur Gathen. Efficient and Optimal Exponentiation in Finite Fields. *Computational Complexity*, 1:360–394, 1991.

A Masking Complexity of Power Functions

Table 5 summarizes the masking complexity classes $(\mathcal{M}_k^n)_k$ for dimensions n in the set $\{3, 5, 7, 9, 10, 11\}$.

Table 5. Cyclotomic classes for $n \in \{3, 5, 7, 9, 10, 11\}$ w.r.t. the masking complexity k.

k	Cyclotomic classes in $\mathcal{M}_k^n$
$n = 3$	
0	$C_0 = \{0\}$, $C_1 = \{1, 2, 4\}$
1	$C_3 = \{3, 6, 5\}$
$n = 5$	
0	$C_0 = \{0\}$, $C_1 = \{1, 2, 4, 8, 16\}$
1	$C_3 = \{3, 6, 12, 24, 17\}$, $C_5 = \{5, 10, 20, 9, 18\}$
2	$C_7 = \{7, 14, 28, 25, 19\}$, $C_{11} = \{11, 22, 13, 26, 21\}$, $C_{15} = \{15, 30, 29, 27, 23\}$
$n = 7$	
0	$C_0 = \{0\}$, $C_1 = \{1, 2, 4, 8, 16, 32, 64\}$
1	$C_3 = \{3, 6, 12, 24, 48, 96, 65\}$, $C_5 = \{5, 10, 20, 40, 80, 33, 66\}$, $C_9 = \{9, 18, 36, 72, 17, 34, 68\}$
2	$C_7 = \{7, 14, 28, 56, 112, 97, 67\}$, $C_{11} = \{11, 22, 44, 88, 49, 98, 69\}$, $C_{13} = \{13, 26, 52, 104, 81, 35, 70\}$, $C_{15} = \{15, 30, 60, 120, 113, 99, 71\}$, $C_{19} = \{19, 38, 76, 25, 50, 100, 73\}$, $C_{21} = \{21, 42, 84, 41, 82, 37, 74\}$, $C_{27} = \{27, 54, 108, 89, 51, 102, 77\}$, $C_{43} = \{43, 86, 45, 90, 53, 106, 85\}$
3	$C_{23} = \{23, 46, 92, 57, 114, 101, 75\}$, $C_{29} = \{29, 58, 116, 105, 83, 39, 78\}$, $C_{31} = \{31, 62, 124, 121, 115, 103, 79\}$, $C_{47} = \{47, 94, 61, 122, 117, 107, 87\}$, $C_{55} = \{55, 110, 93, 59, 118, 109, 91\}$, $C_{63} = \{63, 126, 125, 123, 119, 111, 95\}$
$n = 9$	
0	C_0, C_1
1	C_3, C_5, C_9, C_{17}
2	C_7, C_{11}, C_{13}, C_{15}, C_{19}, C_{21}, C_{25}, C_{27}, C_{35}, C_{37}, C_{41}, C_{45}, C_{51}, C_{73}, C_{75}, C_{83}, C_{85}
3	C_{23}, C_{29}, C_{31}, C_{39}, C_{43}, C_{47}, C_{53}, C_{55}, C_{57}, C_{59}, C_{61}, C_{63}, C_{75}, C_{77}, C_{79}, C_{87}, C_{91}, C_{93}, C_{95}, C_{103}, C_{107}, C_{109}, C_{111}, C_{117}, C_{119}, C_{123}, C_{125}, C_{127}, C_{171}, C_{175}, C_{183}, C_{187}, C_{219}
4	C_{191}, C_{223}, C_{239}
$n = 10$	
0	C_0, C_1
1	C_3, C_5, C_9, C_{17}, C_{33}
2	C_7, C_{11}, C_{13}, C_{15}, C_{19}, C_{21}, C_{25}, C_{27}, C_{35}, C_{37}, C_{41}, C_{45}, C_{49}, C_{51}, C_{69}, C_{73}, C_{85}, C_{99}, C_{147}, C_{165}
3	C_{23}, C_{29}, C_{31}, C_{39}, C_{43}, C_{47}, C_{53}, C_{55}, C_{57}, C_{59}, C_{61}, C_{63}, C_{71}, C_{75}, C_{77}, C_{79}, C_{83}, C_{87}, C_{89}, C_{91}, C_{93}, C_{95}, C_{101}, C_{103}, C_{105}, C_{107}, C_{109}, C_{111}, C_{115}, C_{117}, C_{119}, C_{121}, C_{123}, C_{125}, C_{149}, C_{151}, C_{155}, C_{157}, C_{167}, C_{171}, C_{173}, C_{175}, C_{179}, C_{181}, C_{183}, C_{187}, C_{189}, C_{205}, C_{207}, C_{213}, C_{215}, C_{219}, C_{221}, C_{231}, C_{235}, C_{237}, C_{245}, C_{255}, C_{341}, C_{347}, C_{363}, C_{447}, C_{495}
4	C_{127}, C_{159}, C_{191}, C_{223}, C_{239}, C_{247}, C_{251}, C_{253}, C_{343}, C_{351}, C_{367}, C_{375}, C_{379}, C_{383}, C_{439}, C_{479}, C_{511}
$n = 11$	
0	C_0, C_1
1	C_3, C_5, C_9, C_{17}, C_{33}
2	C_7, C_{11}, C_{13}, C_{15}, C_{19}, C_{21}, C_{25}, C_{27}, C_{35}, C_{37}, C_{41}, C_{45}, C_{49}, C_{51}, C_{67}, C_{69}, C_{73}, C_{81}, C_{85}, C_{99}, C_{137}, C_{153}, C_{163}, C_{165}, C_{293}
3	C_{23}, C_{29}, C_{31}, C_{39}, C_{43}, C_{47}, C_{53}, C_{55}, C_{57}, C_{59}, C_{61}, C_{63}, C_{71}, C_{75}, C_{77}, C_{79}, C_{83}, C_{87}, C_{89}, C_{91}, C_{93}, C_{95}, C_{101}, C_{103}, C_{105}, C_{107}, C_{109}, C_{111}, C_{113}, C_{115}, C_{117}, C_{119}, C_{121}, C_{123}, C_{125}, C_{139}, C_{141}, C_{143}, C_{147}, C_{149}, C_{151}, C_{155}, C_{157}, C_{167}, C_{169}, C_{171}, C_{173}, C_{175}, C_{179}, C_{181}, C_{185}, C_{187}, C_{189}, C_{199}, C_{201}, C_{203}, C_{205}, C_{207}, C_{211}, C_{213}, C_{217}, C_{219}, C_{221}, C_{229}, C_{231}, C_{243}, C_{245}, C_{255}, C_{295}, C_{299}, C_{301}, C_{307}, C_{309}, C_{311},C_{315}, C_{317}, C_{331}, C_{333}, C_{335}, C_{343}, C_{347}, C_{359}, C_{363}, C_{365}, C_{379}, C_{411}, C_{423}, C_{427}, C_{429}, C_{339}, C_{341}, C_{437}, C_{439}, C_{469}, C_{495}, C_{683}, C_{703}, C_{879}, C_{887}
4	C_{127}, C_{159}, C_{183}, C_{191}, C_{215}, C_{223}, C_{233}, C_{235}, C_{237}, C_{239}, C_{247}, C_{249}, C_{251}, C_{253}, C_{303}, C_{319}, C_{349}, C_{351}, C_{367}, C_{371}, C_{373}, C_{375}, C_{381}, C_{383}, C_{413}, C_{415}, C_{431}, C_{443}, C_{445}, C_{447}, C_{463}, C_{471}, C_{475}, C_{477}, C_{479}, C_{491}, C_{493}, C_{501}, C_{503}, C_{507}, C_{509}, C_{511}, C_{687}, C_{695}, C_{699}, C_{727}, C_{731}, C_{735}, C_{751}, C_{759}, C_{763}, C_{767}, C_{895}, C_{959}, C_{991}, C_{1023}

B Detailed Algorithms

We detail hereafter the algorithms to perform a masked DES/PRESENT s-box computation. As in [29], both algorithms use the ISW-based masked field multiplication (SecMult) and the mask refreshing procedure (RefreshMasks). We recall these procedures before giving the masked s-box algorithms.

Algorithm 1 SecMult - dth-order secure multiplication over $\mathbb{F}_{2^n}$

```
Input: shares $a_i$ satisfying $\sum_i a_i = a$, shares $b_i$ satisfying $\sum_i b_i = b$
Output: shares $c_i$ satisfying $\sum_i c_i = ab$
    for $i = 0$ to $d$
        | for $j = i + 1$ to $d$
        |     | $r_{i,j} \leftarrow^{\$} \{0,1\}^n$
        |     | $r_{j,i} \leftarrow (r_{i,j} + a_i b_j) + a_j b_i$
        | endfor
    endfor
    for $i = 0$ to $d$
        | $c_i \leftarrow a_i b_i$
        | for $j = 0$ to $d$, $j \neq i$ do $c_i \leftarrow c_i + r_{i,j}$
    endfor
    return $(c_0, \ldots, c_d)$
```

Algorithm 2 RefreshMasks

```
Input: shares $x_i$ satisfying $\sum_i x_i = x$
Output: shares $x_i$ satisfying $\sum_i x_i = x$
    for $i = 1$ to $d$
        | $tmp \leftarrow^{\$} \{0,1\}^n$
        | $x_0 \leftarrow x_0 + tmp$
        | $x_i \leftarrow x_i + tmp$
    endfor
```

Algorithm 3 Secure higher-order PRESENT s-box evaluation

```
Input: a $d$th-order encoding $(x_0, \ldots, x_d)$ of $x \in \{0,1\}^4$, look-up tables for the $L_i$
Output: a $d$th-order encoding $(t_0, \ldots, t_d)$ of $\mathrm{S}(x)$
    1. for $i = 0$ to $d$ do $y_{2,i} \leftarrow x_i^2$
    2. RefreshMasks$(y_{2,0}, \ldots, y_{2,d})$
    3. $(y_{3,0}, \ldots, y_{3,d}) \leftarrow$ SecMult$\big((x_0, \ldots, x_d), (y_{2,0}, \ldots, y_{2,d})\big)$
    4. $(y_{5,0}, \ldots, y_{5,d}) \leftarrow$ SecMult$\big((y_{2,0}, \ldots, y_{2,d}), (y_{3,0}, \ldots, y_{3,d})\big)$
    5. $(y_{7,0}, \ldots, y_{7,d}) \leftarrow$ SecMult$\big((y_{2,0}, \ldots, y_{2,d}), (y_{5,0}, \ldots, y_{5,d})\big)$
    6. $t_0 \leftarrow a_0 + L_1(x_0) + L_3(y_{3,0}) + L_5(y_{5,0}) + L_7(y_{7,0})$
    7. for $i = 1$ to $d$ do $t_i \leftarrow L_1(x_i) + L_3(y_{3,i}) + L_5(y_{5,i}) + L_7(y_{7,i})$
    8. return $(t_0, \ldots, t_d)$
```

Algorithm 4 Secure higher-order DES s-box evaluation

Input: a dth-order encoding $(x_0, \ldots, x_d)$ of $x \in \{0,1\}^6$, a table of coefficients $a_{i,j}$ for Q_i polynomials
Output: a dth-order encoding $(t_0, \ldots, t_d)$ of $\mathrm{S}(x)$

```
[Computing the $x^{8j}$ powers]
for $i = 0$ to $d$ do $y_{8,i} \leftarrow x_i^8$
RefreshMasks$(y_{8,0}, \ldots, y_{8,d})$
for $i = 0$ to $d$ do $y_{16,i} \leftarrow y_{8,i}^2$
RefreshMasks$(y_{16,0}, \ldots, y_{16,d})$
$(y_{24,0}, \ldots, y_{24,d}) \leftarrow$ SecMult$\big((y_{8,0}, \ldots, y_{8,d}), (y_{16,0}, \ldots, y_{16,d})\big)$
for $i = 0$ to $d$ do $y_{32,i} \leftarrow y_{16,i}^2$
RefreshMasks$(y_{32,0}, \ldots, y_{32,d})$
$(y_{40,0}, \ldots, y_{40,d}) \leftarrow$ SecMult$\big((y_{8,0}, \ldots, y_{8,d}), (y_{32,0}, \ldots, y_{32,d})\big)$
for $i = 0$ to $d$ do $y_{48,i} \leftarrow y_{24,i}^2$
RefreshMasks$(y_{48,0}, \ldots, y_{48,d})$
$(y_{56,0}, \ldots, y_{56,d}) \leftarrow$ SecMult$\big((y_{8,0}, \ldots, y_{8,d}), (y_{48,0}, \ldots, y_{48,d})\big)$

[Evaluating the $Q_i(x^8)$ polynomials]
for $i = 0$ to $7$ do
  | $q_{i,0} \leftarrow a_{i,0}$
  | for $k = 1$ to $d$ do $q_{i,k} \leftarrow 0$
  | for $j = 1$ to $7$ do
  |   | for $k = 0$ to $d$ do $q_{i,k} \leftarrow q_{i,k} + a_{i,j} \cdot y_{8j,k}$
  | endfor
  | RefreshMasks$(q_{i,0}, \ldots, q_{i,d})$
endfor

[Evaluating $\mathrm{S}(x)$]
for $i = 0$ to $d$ do $y_{2,i} \leftarrow x_i^2$
RefreshMasks$(y_{2,0}, \ldots, y_{2,d})$
for $i = 0$ to $d$ do $y_{4,i} \leftarrow y_{2,i}^2$
RefreshMasks$(y_{4,0}, \ldots, y_{4,d})$
$(t_0, \ldots, t_d) \leftarrow$ SecMult$\big((y_{4,0}, \ldots, y_{4,d}), (q_{7,0}, \ldots, q_{7,d})\big)$
for $i = 0$ to $d$ do $t_i \leftarrow t_i + q_{6,i}$
$(t_0, \ldots, t_d) \leftarrow$ SecMult$\big((y_{2,0}, \ldots, y_{2,d}), (t_0, \ldots, t_d)\big)$
$(s_0, \ldots, s_d) \leftarrow$ SecMult$\big((y_{4,0}, \ldots, y_{4,d}), (q_{5,0}, \ldots, q_{5,d})\big)$
for $i = 0$ to $d$ do $t_i \leftarrow t_i + s_i + q_{4,i}$
$(t_0, \ldots, t_d) \leftarrow$ SecMult$\big((x_0, \ldots, x_d), (t_0, \ldots, t_d)\big)$
$(r_0, \ldots, r_d) \leftarrow$ SecMult$\big((y_{4,0}, \ldots, y_{4,d}), (q_{3,0}, \ldots, q_{3,d})\big)$
for $i = 0$ to $d$ do $r_i \leftarrow r_i + q_{2,i}$
$(r_0, \ldots, r_d) \leftarrow$ SecMult$\big((y_{2,0}, \ldots, y_{2,d}), (r_0, \ldots, r_d)\big)$
$(p_0, \ldots, p_d) \leftarrow$ SecMult$\big((y_{4,0}, \ldots, y_{4,d}), (q_{1,0}, \ldots, q_{1,d})\big)$
for $i = 0$ to $d$ do $t_i \leftarrow t_i + r_i + p_i + q_{0,i}$

return $(t_0, \ldots, t_d)$
```

C Look-Up Tables

We detail hereafter the different look-up tables used in our implementations (in C syntax). For the DES s-boxes, the 4-bit outputs are embedded into $\mathbb{F}_{64}$ by padding their most significant bits with two 0s. The used field representations are $\mathbb{F}_{64} \equiv \mathbb{F}_2[x]/(1 + x^5 + x^6)$ for the DES s-boxes and $\mathbb{F}_{16} \equiv \mathbb{F}_2[x]/(1 + x^3 + x^4)$ for the PRESENT s-box (such that `MultGF16`$[a||b] = a \times b$). The multiplication over $\mathbb{F}_{64}$ is performed using the log/alog tables given in Figure 1 while the multiplication over $\mathbb{F}_{16}$ is performed with the multiplication table given in Figure 2. The $a_{i,j}$

coefficients for the masked computation of the first DES s-box are given in Figure 3.[8] Eventually the L_i transformations for the PRESENT s-box are given in table 4.

```
unsigned char* LogGF64[64] = {
   63, 0, 1, 58, 2, 53, 59, 39,
   3, 34, 54, 18, 60, 31, 40, 48,
   4, 43, 35, 22, 55, 15, 19, 26,
   61, 51, 32, 29, 41, 13, 49, 11,
   5, 6, 44, 7, 36, 45, 23, 8,
   56, 37, 16, 46, 20, 24, 27, 9,
   62, 57, 52, 38, 33, 17, 30, 47,
   42, 21, 14, 25, 50, 28, 12, 10
};
unsigned char* AlogGF64[64] = {
   1, 2, 4, 8, 16, 32, 33, 35,
   39, 47, 63, 31, 62, 29, 58, 21,
   42, 53, 11, 22, 44, 57, 19, 38,
   45, 59, 23, 46, 61, 27, 54, 13,
   26, 52, 9, 18, 36, 41, 51, 7,
   14, 28, 56, 17, 34, 37, 43, 55,
   15, 30, 60, 25, 50, 5, 10, 20,
   40, 49, 3, 6, 12, 24, 48, 1
};
```

Fig. 1. Log/alog tables for the multiplication for the multiplication over $\mathbb{F}_{64}$.

```
unsigned char* MultGF16[16][16] = {
   {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 7},
   {8, 9, 10, 11, 12, 13, 14, 15, 0, 2, 4, 6, 8, 10, 12, 14, 9, 11, 13, 15},
   {1, 3, 5, 7, 0, 3, 6, 5, 12, 15, 10, 9, 1, 2, 7, 4, 13, 14, 11, 8, 0, 4},
   {8, 12, 9, 13, 1, 5, 11, 15, 3, 7, 2, 6, 10, 14, 0, 5, 10, 15, 13, 8, 7},
   {2, 3, 6, 9, 12, 14, 11, 4, 1, 0, 6, 12, 10, 1, 7, 13, 11, 2, 4, 14, 8},
   {3, 5, 15, 9, 0, 7, 14, 9, 5, 2, 11, 12, 10, 13, 4, 3, 15, 8, 1, 6, 0, 8},
   {9, 1, 11, 3, 2, 10, 15, 7, 6, 14, 4, 12, 13, 5, 0, 9, 11, 2, 15, 6, 4},
   {13, 7, 14, 12, 5, 8, 1, 3, 10, 0, 10, 13, 7, 3, 9, 14, 4, 6, 12, 11, 1},
   {5, 15, 8, 2, 0, 11, 15, 4, 7, 12, 8, 3, 14, 5, 1, 10, 9, 2, 6, 13, 0},
   {12, 1, 13, 2, 14, 3, 15, 4, 8, 5, 9, 6, 10, 7, 11, 0, 13, 3, 14, 6},
   {5, 8, 12, 1, 15, 2, 10, 7, 9, 4, 0, 14, 5, 11, 10, 4, 15, 1, 13, 3, 8},
   {6, 7, 9, 2, 12, 0, 15, 7, 8, 14, 1, 9, 6, 5, 10, 2, 13, 11, 4, 12, 3}
};
```

Fig. 2. Look-up table for the multiplication over $\mathbb{F}_{16}$.

[8] Due to length constraints we could not include the coefficients for all the DES s-boxes. They will be given in an extended version of the paper.

```
unsigned char* A[8][8] = {
   {14, 5, 58, 31, 39, 36, 47, 54},
   {52, 3, 47, 28, 1, 53, 9, 52},
   {63, 7, 7, 6, 3, 1, 48, 49},
   {49, 12, 0, 9, 33, 50, 49, 3},
   {26, 3, 40, 0, 13, 8, 35, 59},
   {35, 31, 13, 38, 27, 29, 62, 61},
   {12, 27, 49, 46, 40, 50, 28, 41},
   {40, 14, 36, 44, 27, 27, 33, 0}
};
```

Fig. 3. Look-up table for the $a_{i,j}$ coefficients.

```
unsigned char* L1[16] = {0,14,13,3,3,13,14,0,8,6,5,11,11,5,6,8};
unsigned char* L3[16] = {0,14,8,6,10,4,2,12,15,1,7,9,5,11,13,3};
unsigned char* L5[16] = {0,3,4,7,0,3,4,7,11,8,15,12,11,8,15,12};
unsigned char* L7[16] = {0,10,0,10,14,4,14,4,4,14,4,14,10,0,10,0};
```

Fig. 4. Look-up tables for the L_i transformations.

Recursive Diffusion Layers for Block Ciphers and Hash Functions

Mahdi Sajadieh[1], Mohammad Dakhilalian[1], Hamid Mala[2], and Pouyan Sepehrdad[3]

[1] Cryptography & System Security Research Laboratory, Department of Electrical and Computer Engineering, Isfahan University of Technology, Isfahan, Iran
[2] Department of Information Technology Engineering, University of Isfahan, Isfahan, Iran
[3] EPFL, Lausanne, Switzerland
{sadjadieh@ec, mdalian@cc}.iut.ac.ir,
h.mala@eng.ui.ac.ir,
pouyan.sepehrdad@epfl.ch

Abstract. Many modern block ciphers use maximum distance separable (MDS) matrices as the main part of their diffusion layer. In this paper, we propose a new class of diffusion layers constructed from several rounds of Feistel-like structures whose round functions are linear. We investigate the requirements of the underlying linear functions to achieve the maximal branch number for the proposed 4 × 4 words diffusion layer. The proposed diffusion layers only require word-level XORs, rotations, and they have simple inverses. They can be replaced in the diffusion layer of the block ciphers MMB and Hierocrypt to increase their security and performance, respectively. Finally, we try to extend our results for up to 8 × 8 words diffusion layers.

Keywords: Block ciphers, Diffusion layer, Branch number, Provable security

1 Introduction

Block ciphers are one of the most important building blocks in many security protocols. Modern block ciphers are cascades of several rounds and each round consists of confusion and diffusion layers. In many block ciphers, non-linear substitution boxes (S-boxes) form the confusion layer, and a linear transformation provides the required diffusion. The diffusion layer plays an efficacious role in providing resistance against the most well-known attacks on block ciphers, such as differential cryptanalysis (DC) [2] and linear cryptanalysis (LC) [10]. The strength of a diffusion layer is usually quantified by the notion of branch number. Block ciphers exploiting diffusion layers with small branch number may suffer from critical weaknesses against DC and LC, even though their substitution layers consist of S-boxes with strong non-linear properties. Two main strategies for designing block ciphers are Feistel-like and substitution permutation network (SPN) structures. In the last 2 decades, from these two families several structures have been proposed with provable security against DC and LC. Three rounds of Feistel structure [11, 8], five rounds of RC6-like structure [6] and SDS (substitution-diffusion-substitution) structure with a perfect or almost perfect diffusion layer are examples of such structures [9].

1.1 Notations

Let $\mathbf{x}$ be an array of s n-bit elements $\mathbf{x} = [x_{0(n)}, x_{1(n)}, \cdots, x_{s-1(n)}]$. The number of non-zero elements in $\mathbf{x}$ is denoted by $w(\mathbf{x})$ and is known as the Hamming weight of $\mathbf{x}$. For a diffusion layer D applicable on $\mathbf{x}$, we have the following definitions.

Definition 1 ([4]). *The differential branch number of a linear diffusion layer D is defined as:*

$$\beta_d(D) = \min_{\mathbf{x} \neq 0}\{w(\mathbf{x}) + w(D(\mathbf{x}))\}$$

We know that the linear function D can be shown as a binary matrix $\mathbf{B}$, and D^t is a linear function obtained from $\mathbf{B}^t$, where $\mathbf{B}^t$ is the transposition of $\mathbf{B}$.

Definition 2 ([4]). *The linear branch number of a linear diffusion layer D is defined as:*

$$\beta_l(D) = \min_{\mathbf{x}\neq 0}\{w(\mathbf{x}) + w(D^t(\mathbf{x}))\}$$

It is well known that for a diffusion layer acting on s-word inputs, the maximal β_d and β_l are $s+1$ [4]. A diffusion layer D taking its maximal β_d and β_l is called a perfect or MDS diffusion layer. Furthermore, a diffusion layer with $\beta_d = \beta_l = s$ is called an almost perfect diffusion layer [9].

The following notations are used throughout this paper:

$\oplus$	: The bit-wise XOR operation
&	: The bit-wise AND operation
L_i	: Any linear function
ℓ_i	: The linear operator corresponding to the linear function L_i
$(L_1 \oplus L_2)(x)$	: $L_1(x) \oplus L_2(x)$
$L_1L_2(x)$	: $L_1(L_2(x))$
$L_1^2(x)$	: $L_1(L_1(x))$
$I(\cdot)$ function	: Identity function, $I(x) = x$
$x \gg m$ $(x \ll m)$	: Shift of a bit string x by m bits to the right (left)
$x \ggg m$ $(x \lll m)$	: Circular shift of a bit string x by m bits to the right (left)
$\lvert\cdot\rvert$	: Determinant of a matrix in $\mathsf{GF}(2)$
$a\lvert b$	: Concatenation of two bit strings a and b
$x_{(n)}$	: An n-bit value x

1.2 Our contribution

In this paper, we define the notion of a *recursive diffusion layer* and propose a method to construct such perfect diffusion layers.

Definition 3. *A diffusion layer D with s words x_i as the input, and s words y_i as the output is called a recursive diffusion layer if it can be represented in the following form:*

$$D: \begin{cases} y_0 = x_0 \oplus F_0(x_1, x_2, \ldots, x_{s-1}) \\ y_1 = x_1 \oplus F_1(x_2, x_3, \ldots, x_{s-1}, y_0) \\ \vdots \\ y_{s-1} = x_{s-1} \oplus F_{s-1}(y_0, y_1, \ldots, y_{s-2}) \end{cases} \tag{1}$$

where F_0, F_1,..., F_{s-1} are arbitrary functions.

As an example, consider a 2-round Feistel structure with a linear round function L as a recursive diffusion layer with $s = 2$. The input-output relation for this diffusion layer is:

$$D: \begin{cases} y_0 = x_0 \oplus L(x_1) \\ y_1 = x_1 \oplus L(y_0) \end{cases}$$

The quarter-round function of Salsa20 is also an example of a non-linear recursive diffusion layer [1].

$$\begin{cases} y_1 = x_1 \oplus (x_0 + x_3) \\ y_2 = x_2 \oplus (x_0 + y_1) \\ y_3 = x_3 \oplus (y_1 + y_2) \\ y_0 = x_0 \oplus (y_2 + y_3) \end{cases}$$

Also, the lightweight hash function PHOTON [5] and the block cipher LED [7] use MDS matrices based on Eq. (1). In these ciphers, an $m \times m$ MDS matrix $\mathbf{B}^m$ was designed based on the following matrix $\mathbf{B}$ for the performance purposes:

$$\mathbf{B} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & & \\ 0 & 0 & 0 & \cdots & 1 \\ Z_0 & Z_1 & Z_2 & \cdots & Z_{m-1} \end{pmatrix}$$

By matrix $\mathbf{B}$, one elements of m inputs is updated and other elements are shifted. If we use $\mathbf{B}^m$, all inputs are updated, but we must check if this matrix is MDS. One example for $m = 4$ is the PHOTON matrix working over $\mathsf{GF}(2^8)$:

$$\mathbf{B} = \begin{pmatrix} 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \\ 1\,2\,1\,4 \end{pmatrix} \Rightarrow \mathbf{B}^4 = \begin{pmatrix} 1 & 2 & 1 & 4 \\ 4 & 9 & 6 & 17 \\ 17 & 38 & 24 & 66 \\ 66 & 149 & 100 & 11 \end{pmatrix}$$

In this paper, we propose a new approach to design linear recursive diffusion layers with the maximal branch number in which F_i's are composed of one or two linear functions and a number of XOR operations. The design of the proposed diffusion layer is based on the invertibility of some simple linear functions in $\mathsf{GF}(2)$. Linear functions in this diffusion layer can be designed to be low-cost for different sizes of the input words, thus the proposed diffusion layer might be appropriate for resource-constrained devices, such as RFID tags. Although these recursive diffusion layers are not involutory, they have similar inverses with the same computational complexity.

This paper proceeds as follows: In Section 2, we introduce the general structure of our proposed recursive diffusion layer. Then, for one of its instances, we systematically investigate the required conditions for the underlying linear function to achieve the maximal branch number. In Section 3, we propose some other recursive diffusion layers with less than 8 input words and only one linear function. We use two linear functions to have perfect recursive diffusion layer for $s > 4$ in Section 4. Finally, we conclude the paper in Section 5.

2 The Proposed Diffusion Layer

In this section, we introduce a new perfect linear diffusion layer with a recursive structure. The diffusion layer D takes s words x_i for $i = \{0, 1, \ldots, s-1\}$ as input, and returns s words y_i for $i = \{0, 1, \ldots, s-1\}$ as output. So, we can represent this diffusion layer as:

$$y_0|y_1|\cdots|y_{s-1} = D(x_0|x_1|\cdots|x_{s-1})$$

The first class of the proposed diffusion layer D is represented in Fig. 1, where L is a linear function, $\alpha_k, \beta_k \in \{0, 1\}$, $\alpha_0 = 1$, and $\beta_0 = 0$.

This diffusion layer can be represented in the form of Eq. (1) in which the F_i functions are all the same and can be represented as

$$F_i(x_1, x_2, \ldots, x_{s-1}) = \bigoplus_{j=1}^{s-1} \alpha_j x_j \oplus L\left(\bigoplus_{j=1}^{s-1} \beta_j x_j\right)$$

```
Input : $s$ $n$-bit words $x_0, \dots, x_{s-1}$
Output : $s$ $n$-bit words $y_0, \dots, x_{s-1}$
for $i = 0$ to $s-1$ do
    $y_i = x_i$
end for
for $i = 0$ to $s-1$ do
    $y_i = \bigoplus_{j=0}^{s-1} \alpha_{[(j-i) \mod s]} y_j \oplus L\left(\bigoplus_{j=0}^{s-1} \beta_{[(j-i) \mod s]} y_j\right)$
end for
```

Fig. 1. The first class of the recursive diffusion layers

To guarantee the maximal branch number for D, the linear function L and the coefficients α_j and β_j must satisfy some necessary conditions. Conditions on L are expressed in this section and those of α_j's and β_j's are expressed in Section 3. The diffusion layer described by Eq. (2) is an instance that satisfies the necessary conditions on α_j and β_j with $s = 4$. In the rest of this section, we concentrate on the diffusion layers of this form and show that we can find invertible linear functions L such that D becomes a perfect diffusion layer.

$$D : \begin{cases} y_0 = x_0 \oplus x_2 \oplus x_3 \oplus L(x_1 \oplus x_3) \\ y_1 = x_1 \oplus x_3 \oplus y_0 \oplus L(x_2 \oplus y_0) \\ y_2 = x_2 \oplus y_0 \oplus y_1 \oplus L(x_3 \oplus y_1) \\ y_3 = x_3 \oplus y_1 \oplus y_2 \oplus L(y_0 \oplus y_2) \end{cases} \tag{2}$$

As shown in Fig. 2, this diffusion layer has a Feistel-like (GFN) structure, i.e.,

$$F_0(x_1, x_2, x_3) = x_2 \oplus x_3 \oplus L(x_1 \oplus x_3)$$

and for each $i > 0$, y_i is obtained by $(x_i, x_{i+1}, \dots, x_{s-1})$ and $(y_0, y_1, \dots, y_{i-1})$.

The inverse transformation, D^{-1}, has a very simple structure and does not require the inversion of the linear function L. Based on the recursive nature of D, if we start from the last equation of Eq. (2), x_3 is immediately obtained from y_i's. Then knowing x_3 and y_i's, we immediately obtain x_2 from the third line of Eq. (2). x_1 and x_0 can be obtained in the same way. Thus, the inverse of D is:

$$D^{-1} : \begin{cases} x_3 = y_3 \oplus y_1 \oplus y_2 \oplus L(y_0 \oplus y_2) \\ x_2 = y_2 \oplus y_0 \oplus y_1 \oplus L(x_3 \oplus y_1) \\ x_1 = y_1 \oplus x_3 \oplus y_0 \oplus L(x_2 \oplus y_0) \\ x_0 = y_0 \oplus x_2 \oplus x_3 \oplus L(x_1 \oplus x_3) \end{cases}$$

D and D^{-1} are different, but they have the same structure and properties. To show that D has the maximal branch number, first we introduce some lemmas and theorems.

Theorem 4 ([4]). *A Boolean function F has maximal differential branch number if and only if it has maximal linear branch number.*

As a result of Theorem 4, if we prove that the diffusion layer D represented in Eq. (2) has the maximal differential branch number, its linear branch number will be maximal too. Thus, in the following, we focus on the differential branch number.

Lemma 5. *For m linear functions L_1, L_2, ..., L_m, the proposition*

$$a \neq 0 \Rightarrow L_1(a) \oplus L_2(a) \oplus \dots \oplus L_m(a) \neq 0$$

implies that the linear function $L_1 \oplus L_2 \oplus \dots \oplus L_m$ is invertible.

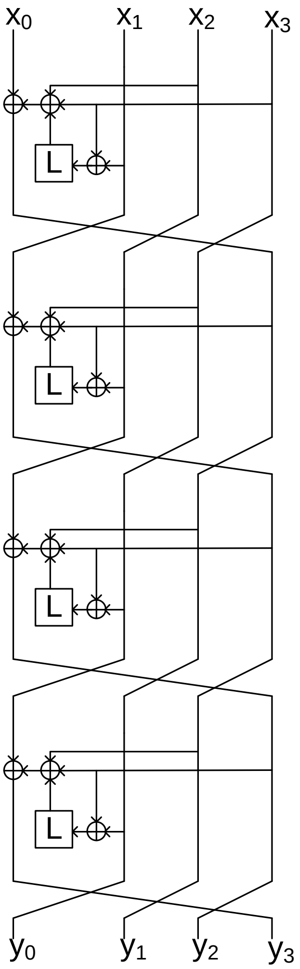


Fig. 2. The proposed recursive diffusion layer of Eq. (2)

Proof. We know that $(L_1 \oplus L_2 \oplus ... \oplus L_m)(x)$ is a linear function and it can be represented as a binary matrix $\mathbf{M}$. So, $\mathbf{M}$ is invertible if and only if $|\mathbf{M}| \neq 0$. □

Lemma 6. *Assume the linear operator ℓ_i corresponds to the linear function $L_i(x)$. If the linear operator ℓ_3 can be represented as the multiplication of two operators ℓ_1 and ℓ_2, then the corresponding linear function $L_3(x) = L_2(L_1(x))$ is invertible if and only if the linear functions $L_1(x)$ and $L_2(x)$ are invertible.*

Proof. If $L_1(x)$ and $L_2(x)$ are invertible, clearly $L_3(x)$ is invertible too. On the other hand, if $L_3(x)$ is invertible then $L_1(x)$ must be invertible, otherwise there are distinct x_1 and x_2 such that $L_1(x_1) = L_1(x_2)$. Thus, $L_3(x_1) = L_2(L_1(x_1)) = L_2(L_1(x_2)) = L_3(x_2)$ which contradicts the invertibility of $L_3(x)$. The invertibility of $L_2(x)$ is proved in the same way.

□

Example 1: We can rewrite the linear function $L_3(x) = L^3(x) \oplus x$ $(\ell_3 = \ell^3 \oplus I)$ as $L_3(x) = L_1(L_2(x))$, where $L_1(x) = L(x) \oplus x$ $(\ell_1 = \ell \oplus I)$ and $L_2(x) = L^2(x) \oplus L(x) \oplus x$ $(\ell_2 = \ell^2 \oplus \ell \oplus I)$. Thus, the invertibility of $L_3(x)$ is equivalent to the invertibility of the two linear functions $L_1(x)$ and $L_2(x)$.

Theorem 7. *For the diffusion layer represented in Eq. (2), if the four linear functions $L(x)$, $x \oplus L(x)$, $x \oplus L^3(x)$, and $x \oplus L^7(x)$ are invertible, then this diffusion layer is perfect.*

Proof. We show that the differential branch number of this diffusion layer is 5. First, the 4 words of the output are directly represented as functions of the 4 words of the input:

$$D : \begin{cases} y_0 = x_0 \oplus L(x_1) \oplus x_2 \oplus x_3 \oplus L(x_3) \\ y_1 = x_0 \oplus L(x_0) \oplus x_1 \oplus L(x_1) \oplus L^2(x_1) \oplus x_2 \oplus L^2(x_3) \\ y_2 = L^2(x_0) \oplus x_1 \oplus L(x_1) \oplus L^3(x_1) \oplus x_2 \oplus L(x_2) \oplus x_3 \oplus L^2(x_3) \oplus L^3(x_3) \\ y_3 = x_0 \oplus L^2(x_0) \oplus L^3(x_0) \oplus L(x_1) \oplus L^2(x_1) \oplus L^3(x_1) \oplus L^4(x_1) \\ \qquad \oplus L(x_2) \oplus L^2(x_2) \oplus L^2(x_3) \oplus L^4(x_3) \end{cases} \tag{3}$$

Now, we show that if the number of active (non-zero) words in the input is m, where $m = 1, 2, 3, 4$, then the number of non-zero words in the output is greater than or equal to $5 - m$. The diffusion layer represented in Eq. (2) is invertible. Consider $m = 4$, then all of the 4 words in the input are active, and we are sure at least one of the output words is active too. Thus the theorem is correct for $m = 4$. The remainder of the proof is performed for the 3 cases of $w(\varDelta(\mathbf{x})) = m$, for $m = 1, 2, 3$ separately. In each of these cases, some conditions are forced on the linear function L.

Case 1: $w(\triangle \mathbf{x}) = 1$

To study this case, first the subcase

$$(\triangle x_0 \neq 0, \triangle x_1 = \triangle x_2 = \triangle x_3 = 0 \ \text{ or } \ \triangle \mathbf{x} = \triangle x_0|0|0|0)$$

is analyzed. For this subcase, Eq. (3) is simplified to:

$$D : \begin{cases} \triangle y_0 = \triangle x_0 \\ \triangle y_1 = (I \oplus L)(\triangle x_0) \\ \triangle y_2 = L^2(\triangle x_0) \\ \triangle y_3 = (I \oplus L^2 \oplus L^3)(\triangle x_0) \end{cases}$$

If D is a perfect diffusion layer then $\triangle y_0$, $\triangle y_1$, $\triangle y_2$ and $\triangle y_3$ must be non-zero. Clearly, $\triangle y_0$ is non-zero, and based on Lemma 5, the conditions for $\triangle y_1$, $\triangle y_2$ and $\triangle y_3$ to be non-zero are that the linear functions $I \oplus L$, L^2 and $I \oplus L^2 \oplus L^3$ must be invertible. Note that based on Lemma 6, the invertibility of L^2 yields the invertibility of L. Considering Lemma 6, if the other three sub-cases are studied, it is induced that the linear functions $x \oplus L(x) \oplus L^2(x)$ and $x \oplus L(x) \oplus L^3(x)$ must also be invertible.

Case 2: $w(\triangle \mathbf{x}) = 2$

In this case, there exist exactly two active words in the input difference and we obtain some conditions on the linear function L to guarantee the branch number 5 for D. In the following, we only analyze the subcase

$$(\triangle x_0, \triangle x_1 \neq 0 \ \text{ and } \ \triangle x_2 = \triangle x_3 = 0 \ \text{ or } \ \triangle \mathbf{x} = \triangle x_0|\triangle x_1|0|0)$$

With this assumption, Eq. (3) is simplified to:

$$D : \begin{cases} \triangle y_0 = \triangle x_0 \oplus L(\triangle x_1) \\ \triangle y_1 = (I \oplus L)(\triangle x_0) \oplus (I \oplus L \oplus L^2)(\triangle x_1) \\ \triangle y_2 = L^2(\triangle x_0) \oplus (I \oplus L \oplus L^3)(\triangle x_1) \\ \triangle y_3 = (I \oplus L^2 \oplus L^3)(\triangle x_0) \oplus (L \oplus L^2 \oplus L^3 \oplus L^4)(\triangle x_1) \end{cases} \tag{4}$$

To show that $w(\triangle \mathbf{y})$ is greater than or equal to 3, we must find some conditions on L such that if one of the $\triangle y_i$'s is zero, then the other three $\triangle y_j$'s cannot be zero. Let $\triangle y_0 = 0$, then:

$$\triangle x_0 \oplus L(\triangle x_1) = 0 \Rightarrow \triangle x_0 = L(\triangle x_1)$$

If $\triangle x_0$ is replaced in the last three equations of Eq. (4), we obtain $\triangle y_1$, $\triangle y_2$ and $\triangle y_3$ as follows:

$$\begin{cases} \triangle y_1 = \triangle x_1 \\ \triangle y_2 = \triangle x_1 \oplus L(\triangle x_1) \\ \triangle y_3 = L^2(\triangle x_1) \end{cases}$$

Obviously, $\triangle y_1$ is not zero. Furthermore, for $\triangle y_2$ and $\triangle y_3$ to be non-zero, considering Lemma 5, we conclude that the functions $x \oplus L(x)$ and $L^2(x)$ must be invertible. This condition was already obtained in the Case 1. We continue this procedure for $\triangle y_1 = 0$.

$$\begin{aligned} &\triangle y_1 = \triangle x_0 \oplus L(\triangle x_0) \oplus x_1 \oplus L(\triangle x_1) \oplus L^2(\triangle x_1) = 0 \Rightarrow \\ &\triangle x_0 \oplus L(\triangle x_0) = x_1 \oplus L(\triangle x_1) \oplus L^2(\triangle x_1) \end{aligned}$$

From the previous subcase, we know that if $\triangle y_0 = 0$ then $\triangle y_1 \neq 0$. Thus we conclude that, $\triangle y_0$ and $\triangle y_1$ cannot be simultaneously zero. Therefore, by contraposition we obtain that if $\triangle y_1 = 0$ then $\triangle y_0 \neq 0$. So, we only check $\triangle y_2$ and $\triangle y_3$. From the third equation in Eq. (4), we have:

$$(I \oplus L)(\triangle y_2) = L^2(\triangle x_1) \oplus L^3(\triangle x_1) \oplus L^4(\triangle x_1) \oplus \triangle x_1 \oplus L^2(\triangle x_1) \oplus L^3(\triangle x_1) \oplus L^4(\triangle x_1) = \triangle x_1$$

$x \oplus L(x)$ is invertible, thus we conclude that with the two active words $\triangle x_0$ and $\triangle x_1$ in the input, $\triangle y_1$ and $\triangle y_2$ cannot be zero simultaneously. With the same procedure, we can prove that $\triangle y_1$ and $\triangle y_3$ cannot be zero simultaneously.

Here we only gave the proof for the case $(\triangle x_0, \triangle x_1 \neq 0,\ \triangle x_2 = \triangle x_3 = 0)$. We performed the proof procedure for the other cases and no new condition was added to the previous set of conditions in Case 1.

Case 3: $w(\triangle \mathbf{x}) = 3$

In this case, assuming three active words in the input, we show that the output has at least 2 non-zero words. Here, only the case

$$(\triangle x_0, \triangle x_1, \triangle x_2 \neq 0 \text{ and } \triangle x_3 = 0 \text{ or } \triangle \mathbf{x} = \triangle x_0 | \triangle x_1 | \triangle x_2 | 0)$$

is analyzed. The result holds for the other three cases with $w(\triangle \mathbf{x}) = 3$. Let rewrite the Eq. (3) for $\triangle x_3 = 0$ as follows:

$$D: \begin{cases} \triangle y_0 = \triangle x_0 \oplus L(\triangle x_1) \oplus \triangle x_2 \\ \triangle y_1 = (I \oplus L)(\triangle x_0) \oplus (I \oplus L \oplus L^2)(\triangle x_1) \oplus \triangle x_2 \\ \triangle y_2 = L^2(\triangle x_0) \oplus (I \oplus L \oplus L^3)(\triangle x_1) \oplus (I \oplus L)(\triangle x_2) \\ \triangle y_3 = (I \oplus L^2 \oplus L^3)(\triangle x_0) \oplus (L \oplus L^2 \oplus L^3 \oplus L^4)(\triangle x_1) \oplus (L \oplus L^2)(\triangle x_2) \end{cases} \quad (5)$$

When $\triangle y_0 = \triangle y_1 = 0$, from the first 2 lines of Eq. (5), $\triangle x_0$ and $\triangle x_1$ are obtained as the function of $\triangle x_2$.

$$\begin{cases} \triangle y_0 = \triangle x_0 \oplus L(\triangle x_1) \oplus \triangle x_2 = 0 \\ \triangle y_1 = \triangle x_0 \oplus L(\triangle x_0) \oplus \triangle x_1 \oplus L(\triangle x_1) \oplus L^2(\triangle x_1) \oplus \triangle x_2 = 0 \end{cases} \Rightarrow \begin{cases} \triangle x_1 = L(\triangle x_2) \\ \triangle x_0 = \triangle x_2 \oplus L^2(\triangle x_2) \end{cases}$$

Now, replacing $\triangle x_0 = \triangle x_2 \oplus L^2(\triangle x_2)$ and $\triangle x_1 = L(\triangle x_2)$ into $\triangle y_2$ and $\triangle y_3$ yields:

$$\begin{cases} \triangle y_2 = L^2(\triangle x_0) \oplus (I \oplus L \oplus L^3)(\triangle x_1) \oplus (I \oplus L)(\triangle x_2) = \triangle x_2 \\ \triangle y_3 = (I \oplus L^2 \oplus L^3)(\triangle x_0) \oplus (L \oplus L^2 \oplus L^3 \oplus L^4)(\triangle x_1) \oplus (L \oplus L^2)(\triangle x_2) = (I \oplus L)(\triangle x_2) \end{cases}$$

From Case 1, we know that the functions $x \oplus L(x)$ and $x \oplus L(x) \oplus L^2(x)$ are invertible. Therefore, $\triangle y_2$ and $\triangle y_3$ are non-zero. If the other sub-cases with three active words in the input are investigated, it is easy to see that no new condition is added to the present conditions on L.

Finally, we conclude that the diffusion layer D presented in Fig. 1 is perfect if the linear functions

$$\begin{cases} L_1(x) = L(x) \\ L_2(x) = x \oplus L(x) \\ L_3(x) = x \oplus L(x) \oplus L^2(x) \\ L_4(x) = x \oplus L(x) \oplus L^3(x) \\ L_5(x) = x \oplus L^2(x) \oplus L^3(x) \end{cases}$$

are invertible. We know that $L_3(L_2(x)) = x \oplus L^3(x)$ and $L_5(L_4(L_2(x))) = x \oplus L^7(x)$. Thus, by Lemma 6, we can summarize the necessary conditions on the linear function L as the invertibility of $L(x)$, $(I \oplus L)(x)$, $(I \oplus L^3)(x)$ and $(I \oplus L^7)(x)$.

□

Next, we need a simple method to check whether a linear function L satisfies the conditions of Theorem 7 or not. For this purpose, we use the binary matrix representation of L. Assume that x_i is an n-bit word. Hence, we can represent a linear function L with an $n \times n$ matrix $\mathbf{A}$ with elements in $\mathsf{GF}(2)$. By using Lemma 5, if L is invertible, $\mathbf{A}$ is not singular over $\mathsf{GF}(2)$ ($|\mathbf{A}| \neq 0$). To investigate whether a linear function L satisfies the conditions of Theorem 7, we construct the corresponding matrix $\mathbf{A}_{n \times n}$ from L and check the non-singularity of the matrices $\mathbf{A}$, $\mathbf{I} \oplus \mathbf{A}$, $\mathbf{I} \oplus \mathbf{A}^3$ and $\mathbf{I} \oplus \mathbf{A}^7$ in $\mathsf{GF}(2)$. We introduce some lightweight linear functions with n-bit

Table 1. Some instances of the linear function L satisfying Theorem 7

Length of the input	Some linear functions L
4	$L(x) = (x \oplus x \ll 3) \lll 1$
8	$L(x) = (x \oplus (x \;\&\; \mathsf{0x2}) \ll 1) \lll 1$
16	$L(x) = (x \oplus x \ll 15) \lll 1$
32	$L(x) = (x \oplus x \ll 31) \lll 15$ or $L(x) = (x \lll 24) \oplus (x \;\&\; \mathsf{0xFF})$
64	$L(x) = (x \oplus x \ll 63) \lll 1$ or $L(x) = (x \lll 8) \oplus (x \;\&\; \mathsf{0xFFFF})$

inputs/outputs in Table 1 that satisfy the above conditions. Note that there exist many linear functions which satisfy the conditions of Theorem 7.

Unlike the shift and XOR operations, rotation cannot be implemented as a single instruction on many processors. So, to have more efficient diffusion layers, we introduce new L functions for 32-bit and 64-bit inputs in Table 2 that only use shift and XOR operations.

Table 2. Some examples for the linear function L satisfying Theorem 7 without a circular shift

Length of the input	Sample linear functions L
32	$L(x) = (x \ll 3) \oplus (x \gg 1)$
64	$L(x) = (x \ll 15) \oplus (x \gg 1)$

We can use this diffusion layer with $L(x) = (x \ll 3) \oplus (x \gg 1)$ instead of the diffusion layers used in the block ciphers MMB [3] or Hierocrypt [12]. In MMB, the diffusion layer is a 4 × 4 binary matrix with branch number 4. If we use the proposed diffusion layer in this cipher, it becomes stronger against LC and DC attacks. This change also prevents the attacks presented on this block cipher in [14]. By computer simulations, we observed that this modification reduces the performance of MMB by about 10%. Also, if we use our proposed diffusion layer with the same $L(x)$, instead of the binary matrix of the block cipher Hierocrypt (called $\mathrm{MDS_H}$ [12]), we can achieve a 2 times faster implementation with the same level of security.

Moreover, in the nested SPN structure of Hierocrypt, we replaced the MDS matrix of AES in $\mathsf{GF}(2^{32})$ (because inputs of $\mathrm{MDS_H}$ are 4 32-bit words) with irreducible polynomial $x^{32}+x^7+x^5+x^3+x^2+x+1$ [13] instead of the binary matrix $\mathrm{MDS_H}$. We observed that the replacement of our proposed diffusion layer instead of $\mathrm{MDS_H}$ yields 5% better performance than the replacement of the AES matrix in $\mathsf{GF}(2^{32})$.

In Eq. (1), if $F_i(x_1, x_2, x_3) = F_0(x_1, x_2, x_3) = L(x_1) \oplus x_2 \oplus L^2(x_3)$, where $L(x) = 2x$ and $x \in \mathsf{GF}(2^8)$, PHOTON MDS matrix is obtained [5]. If we change $\mathbf{B}$ to Eq. (2) and define $L(x) = 2x$, we have:

$$\mathbf{B} = \begin{pmatrix} 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \\ 1&2&1&3 \end{pmatrix} \Rightarrow \mathbf{B}^4 = \begin{pmatrix} 1&2&1&3 \\ 3&7&1&4 \\ 4&11&3&13 \\ 13&30&6&20 \end{pmatrix}$$

3 Other Desirable Structures for the Proposed Diffusion Layer

In Section 2, the general form of the proposed diffusion layer was introduced in Fig. 1. Then by assuming a special case of α_i's and β_i's, an instance of this diffusion layer was given in Eq. (2).

In this section, we obtain all sets of α_i's and β_i's such that the diffusion layer of Fig. 1 becomes perfect. We know some properties of α_i's and β_i's; for instance if all the words of the output are directly represented as the function of input words, a function of each x_i $(0 \leq i \leq s-1$) must appear in each equation. Another necessary condition is obtained for two active words of the input. Assume there exist only two indices i, j such that $x_i, x_j \neq 0$. If we write each two output words y_p, y_q in a direct form as a function of x_i and x_j, we obtain:

$$\begin{cases} y_p = L_{p_i}(x_i) \oplus L_{p_j}(x_j) \\ y_q = L_{q_i}(x_i) \oplus L_{q_j}(x_j) \end{cases}$$

If $\frac{\ell_{p_i}}{\ell_{q_i}} = \frac{\ell_{p_j}}{\ell_{q_j}}$ (or $\begin{vmatrix} \ell_{p_i} & \ell_{p_j} \\ \ell_{q_i} & \ell_{q_j} \end{vmatrix}$=0), then $y_p = 0$ is equivalent to $y_q = 0$. Thus, the minimum number of active words in the input and output is less than or equal to s, and the branch number will not reach the maximal value $s+1$. This procedure must be repeated for 3 and more active words in the input. As an extension, we can use Lemma 3 of [13].

Lemma 8. *Assume the diffusion layer has m inputs/outputs bits and ℓ is the linear operator of $L(x)$ and I is the linear operator of $I(x)$. Moreover, $\boldsymbol{ML}_D$ is an $m \times m$ matrix representation of the operator of the diffusion layer. If D is perfect, then all the sub-matrices of $\boldsymbol{ML}_D$ is non-singular.*

If we construct the $\mathbf{ML}_D$ of Eq. (2), we have:

$$\mathbf{ML}_D = \begin{pmatrix} I & \ell & I & I \oplus \ell \\ I \oplus \ell & I \oplus \ell \oplus \ell^2 & I & \ell^2 \\ \ell^2 & I \oplus \ell \oplus \ell^3 & I \oplus \ell & I \oplus \ell^2 \oplus \ell^3 \\ I \oplus \ell^2 \oplus \ell^3 & \ell \oplus \ell^2 \oplus \ell^3 \oplus \ell^4 & \ell \oplus \ell^2 & \ell^2 \oplus \ell^4 \end{pmatrix}$$

If we calculate 69 sub-matrix determinant of $\mathbf{ML}_D$, we find the result of Theorem 7. However, by following this procedure, it is complicated to obtain all sets of α_i's and β_i's analytically. So, by systematizing the method based on Lemma 8, we performed a computer simulation to obtain all sets of α_i's and β_i's in the diffusion layer in Fig. 1 that yield a perfect diffusion. We searched for all α_i's and β_i's that make the diffusion layer of Fig. 1 a perfect diffusion layer. This procedure was repeated for $s = 2, 3, \ldots, 8$. We found one set of $(\alpha_i,\ \beta_i)$ for $s = 2$, four sets for $s = 3$, and four sets for $s = 4$. The obtained diffusion layers along with the conditions on the underlying linear function L are reported in Table 3. We observed that for $s = 5, 6, 7$ the diffusion layer introduced in Fig. 1 cannot be perfect.

Note that some linear functions in Table 1 and Table 2 such as $L(x) = (x \ll 15) \oplus (x \gg 1)$ cannot be used in the diffusion layers for which $x \oplus L^{15}(x)$ must be invertible.

As we can see in Fig. 1 and its instances presented in Table 3, there exists some kind of regularity in the equations defining y_i's, in the sense that the form of y_{i+1} is determined by the form of y_i and vice versa (F_i's are the same in Eq. (1)). However, we can present some non-regular recursive diffusion layers with the following more general form (F_i's are different):

where $A_{i,j}, B_{i,j} \in \{0, 1\}$. If $A_{i,j} = \alpha_{(j-i) \mod s}$ and $B_{i,j} = \beta_{(j-i) \mod s}$, then Fig. 3 is equivalent to Fig. 1. The main property of this new structure is that it still has one linear function L and a simple structure for the inverse. For example, if $s = 4$, then:

$$D : \begin{cases} y_0 = x_0 \oplus A_{0,1} \cdot x_1 \oplus A_{0,2} \cdot x_2 \oplus A_{0,3} \cdot x_3 \oplus L(B_{0,1} \cdot x_1 \oplus B_{0,2} \cdot x_2 \oplus B_{0,3} \cdot x_3) \\ y_1 = x_1 \oplus A_{1,0} \cdot y_0 \oplus A_{1,2} \cdot x_2 \oplus A_{1,3} \cdot x_3 \oplus L(B_{1,0} \cdot y_0 \oplus B_{1,2} \cdot x_2 \oplus B_{1,3} \cdot x_3) \\ y_2 = x_2 \oplus A_{2,0} \cdot y_0 \oplus A_{2,1} \cdot y_1 \oplus A_{2,3} \cdot x_3 \oplus L(B_{2,0} \cdot y_0 \oplus B_{2,1} \cdot y_1 \oplus B_{2,3} \cdot x_3) \\ y_3 = x_3 \oplus A_{3,0} \cdot y_0 \oplus A_{3,1} \cdot y_1 \oplus A_{3,2} \cdot y_2 \oplus L(B_{3,0} \cdot y_0 \oplus B_{3,1} \cdot y_1 \oplus B_{3,2} \cdot y_2) \end{cases}$$

Table 3. Perfect regular recursive diffusion layers for $s < 8$ with only one linear function L

s	Diffusion Layer	Function that must be invertible
2	$D: \begin{cases} y_0 = x_0 \oplus L(x_1) \\ y_1 = x_1 \oplus L(y_0) \end{cases}$	$L(x)$ and $x \oplus L(x)$
3	$D: \begin{cases} y_0 = x_0 \oplus L(x_1 \oplus x_2) \\ y_1 = x_1 \oplus L(x_2 \oplus y_0) \\ y_2 = x_2 \oplus L(y_0 \oplus y_1) \end{cases}$	$L(x)$, $x \oplus L(x)$ and $x \oplus L^3(x)$
3	$D: \begin{cases} y_0 = x_0 \oplus x_1 \oplus L(x_1 \oplus x_2) \\ y_1 = x_1 \oplus x_2 \oplus L(x_2 \oplus y_0) \\ y_2 = x_2 \oplus y_0 \oplus L(y_0 \oplus y_1) \end{cases}$	$L(x)$, $x \oplus L(x)$, $x \oplus L^3(x)$ and $x \oplus L^7(x)$
3	$D: \begin{cases} y_0 = x_0 \oplus x_2 \oplus L(x_1 \oplus x_2) \\ y_1 = x_1 \oplus y_0 \oplus L(x_2 \oplus y_0) \\ y_2 = x_2 \oplus y_1 \oplus L(y_0 \oplus y_1) \end{cases}$	$L(x)$, $x \oplus L(x)$, $x \oplus L^3(x)$ and $x \oplus L^7(x)$
3	$D: \begin{cases} y_0 = x_0 \oplus x_1 \oplus x_2 \oplus L(x_1 \oplus x_2) \\ y_1 = x_1 \oplus x_2 \oplus y_0 \oplus L(x_2 \oplus y_0) \\ y_2 = x_2 \oplus y_0 \oplus y_1 \oplus L(y_0 \oplus y_1) \end{cases}$	$L(x)$, $x \oplus L(x)$, and $x \oplus L^3(x)$
4	$D: \begin{cases} y_0 = x_0 \oplus x_2 \oplus x_3 \oplus L(x_1 \oplus x_3) \\ y_1 = x_1 \oplus x_3 \oplus y_0 \oplus L(x_2 \oplus y_0) \\ y_2 = x_2 \oplus y_0 \oplus y_1 \oplus L(x_3 \oplus y_1) \\ y_3 = x_3 \oplus y_1 \oplus y_2 \oplus L(y_0 \oplus y_2) \end{cases}$	$L(x)$, $x \oplus L(x)$, $x \oplus L^3(x)$ and $x \oplus L^7(x)$
4	$D: \begin{cases} y_0 = x_0 \oplus x_1 \oplus x_2 \oplus L(x_1 \oplus x_3) \\ y_1 = x_1 \oplus x_2 \oplus x_3 \oplus L(x_2 \oplus y_0) \\ y_2 = x_2 \oplus x_3 \oplus y_0 \oplus L(x_3 \oplus y_1) \\ y_3 = x_3 \oplus y_0 \oplus y_1 \oplus L(y_0 \oplus y_2) \end{cases}$	$L(x)$, $x \oplus L(x)$, $x \oplus L^3(x)$ and $x \oplus L^7(x)$
4	$D: \begin{cases} y_0 = x_0 \oplus x_2 \oplus L(x_1 \oplus x_2 \oplus x_3) \\ y_1 = x_1 \oplus x_3 \oplus L(x_2 \oplus x_3 \oplus y_0) \\ y_2 = x_2 \oplus y_0 \oplus L(x_3 \oplus y_0 \oplus y_1) \\ y_3 = x_3 \oplus y_1 \oplus L(y_0 \oplus y_1 \oplus y_2) \end{cases}$	$L(x)$, $x \oplus L(x)$, $x \oplus L^3(x)$, $x \oplus L^7(x)$ and $x \oplus L^{15}(x)$
4	$D: \begin{cases} y_0 = x_0 \oplus x_1 \oplus x_3 \oplus L(x_1 \oplus x_2 \oplus x_3) \\ y_1 = x_1 \oplus x_2 \oplus y_0 \oplus L(x_2 \oplus x_3 \oplus y_0) \\ y_2 = x_2 \oplus x_3 \oplus \oplus y_1 \oplus L(x_3 \oplus y_0 \oplus y_1) \\ y_3 = x_3 \oplus y_0 \oplus y_2 \oplus L(y_0 \oplus y_1 \oplus y_2) \end{cases}$	$L(x)$, $x \oplus L(x)$, $x \oplus L^3(x)$, $x \oplus L^7(x)$ and $x \oplus L^{15}(x)$

```
Input : $s$ $n$-bit words $x_0, \dots, x_{s-1}$
Output : $s$ $n$-bit words $y_0, \dots, x_{s-1}$
for $i = 0$ to $s - 1$ do
    $y_i = x_i$
end for
for $i = 0$ to $s - 1$ do
    $y_i = y_i \oplus \left( \bigoplus_{j=0, j \neq i}^{s-1} A_{i,j} y_j \right) \oplus L \left( \bigoplus_{j=0, j \neq i}^{s-1} B_{i,j} y_j \right)$
end for
```

Fig. 3. Non-regular recursive diffusion layers

We searched the whole space for $s = 3$ and $s = 4$ (the order of search spaces are 2^{12} and 2^{24} respectively). For $s = 3$, we found 196 structures with branch number 4 and for $s = 4$, 1634 structures with branch number 5. The linear functions that must be invertible for each case are different. Among the 196 structures for $s = 3$, the structure with the minimum number of operations (only 7 XORs and one L evaluation) is the following:

$$D : \begin{cases} y_0 = x_0 \oplus x_1 \oplus x_2 \\ y_1 = x_1 \oplus x_2 \oplus L(y_0 \oplus x_2) \\ y_2 = x_2 \oplus y_0 \oplus y_1 \end{cases}$$

where $L(x)$, $x \oplus L(x)$ and $x \oplus L^3(x)$ must be invertible.

This relation is useful to enlarge the first linear function of the new hash function JH for 3 inputs [15]. For $s = 4$, we did not find any D with the number of L evaluations less than four. However, the one with the minimum number of XORs is given as below:

$$D : \begin{cases} y_0 = x_0 \oplus x_1 \oplus x_2 \oplus L(x_3) \\ y_1 = x_1 \oplus x_3 \oplus y_0 \oplus L(x_2 \oplus y_0) \\ y_2 = x_2 \oplus x_3 \oplus y_0 \oplus L(x_3 \oplus y_0) \\ y_3 = x_3 \oplus y_1 \oplus y_2 \oplus L(y_0) \end{cases}$$

Searching the whole space for $s = 5, 6, ...$ is too time consuming (note that for $s = 5$, the order of search has complexity 2^{40}) and we could not search all the space for $s \geq 5$.

4 Increasing the Number of Linear Functions

In Section 3, we observed that for $s > 4$ we cannot design a regular recursive diffusion layer in the form of Fig. 1 with only one linear function L. In this section, we increase the number of linear functions to overcome the regular structure of the diffusion layer of Eq. (2). A new structure is represented in Fig. 4, where $\alpha_k, \beta_k, \gamma_k \in \{0, 1\}$, $k \in \{0, 1, ..., s-1\}$, $\alpha_0 = 1, \beta_0 = 0$ and $\gamma_0 = 0$.

```
Input : $s$ $n$-bit words $x_0, \ldots, x_{s-1}$
Output : $s$ $n$-bit words $y_0, \ldots, x_{s-1}$
for $i = 0$ to $s - 1$ do
    $y_i = x_i$
end for
for $i = 0$ to $s - 1$ do
    $y_i = \bigoplus_{j=0}^{s-1} \alpha_{[(j-i) \mod s]} y_j \oplus L_1 \left( \bigoplus_{j=0}^{s-1} \beta_{[(j-i) \mod s]} y_j \right) \oplus L_2 \left( \bigoplus_{j=0}^{s-1} \gamma_{[(j-i) \mod s]} y_j \right)$
end for
```

Fig. 4. Non-regular recursive diffusion layers with two linear functions L

If L_1 and L_2 are two distinct linear functions, Fig. 4 is too complicated to easily obtain conditions on L_1 and L_2 that make it a perfect diffusion layer. To obtain simplified conditions for a maximal branch number, let L_1 and L_2 have a simple relation like $L_2(x) = L_1^2(x)$ or $L_2(x) = L_1^{-1}(x)$. For the linear functions in Table 2 and Table 3, $L^2(x)$ is more complex in comparison with $L(x)$. However, there exist some linear functions $L(x)$ such that $L^{-1}(x)$ is simpler than $L^2(x)$. As an example, for $L(x_{(n)}) = (x_{(n)} \oplus x_{(n)} \ll b) \lll a$, where $b < \frac{n}{2}$ we have:

$$L^{-1}(x_{(n)}) = ((x_{(n)} \ggg a) \oplus (x_{(n)} \ggg a) \ll b)$$

In Table 4, we introduce some recursive diffusion layers with ($L_1 = L$ and $L_2 = L^{-1}$) or ($L_1 = L$ and $L_2 = L^2$) that have maximal branch numbers. These diffusion layers are obtained similar to that of Table 3. In this table, for each case only y_0 is presented. Other y_i's can be easily obtained from Fig. 4, since F_i's are all the same.

Table 4. Some perfect regular diffusion layers for $s = 5, 6, 7, 8$ with two linear functions

s	y_0 in a perfect diffusion Layer
5	$y_0 = x_0 \oplus x_2 \oplus x_3 \oplus L(x_4) \oplus L^2(x_1)$
5	$y_0 = L^{-1}(x_4) \oplus x_0 \oplus x_2 \oplus L(x_1 \oplus x_3 \oplus x_4)$
6	$y_0 = x_0 \oplus x_5 \oplus L(x_3 \oplus x_5) \oplus L^2(x_1 \oplus x_2 \oplus x_4)$
6	$y_0 = L^{-1}(x_2 \oplus x_5) \oplus x_0 \oplus x_3 \oplus L(x_1 \oplus x_3 \oplus x_4 \oplus x_5)$
7	$y_0 = x_0 \oplus x_2 \oplus x_4 \oplus L(x_3) \oplus L^2(x_1 \oplus x_2 \oplus x_4 \oplus x_5 \oplus x_6)$
7	$y_0 = L^{-1}(x_1 \oplus x_3 \oplus x_6) \oplus x_0 \oplus x_6 \oplus L(x_1 \oplus x_2 \oplus x_4 \oplus x_5)$
8	$y_0 = x_0 \oplus x_1 \oplus x_3 \oplus x_4 \oplus L(x_2 \oplus x_3 \oplus x_5) \oplus L^2(x_1 \oplus x_5 \oplus x_6 \oplus x_7)$
8	$y_0 = L^{-1}(x_3 \oplus x_4 \oplus x_7) \oplus x_0 \oplus x_1 \oplus x_2 \oplus x_4 \oplus L(x_1 \oplus x_5 \oplus x_6 \oplus x_7)$

If the 12 linear functions:

$$\begin{array}{lll} L(x) & I \oplus L(x) & I \oplus L^3(x) \\ I \oplus L^7(x) & I \oplus L^{15}(x) & I \oplus L^{31}(x) \\ I \oplus L^{63}(x) & I \oplus L^{127}(x) & I \oplus L^{255}(x) \\ I \oplus L^{511}(x) & I \oplus L^{1023}(x) & I \oplus L^{2047} \end{array}$$

are invertible (all irreducible polynomials up to degree 11), then all the diffusion layers introduced in Table 4 are perfect. One example for a 32-bit linear function satisfying these conditions is:

$$L(x_{(32)}) = (x_{(32)} \oplus (x_{(32)} \gg 27)) \lll 15$$

5 Conclusion

In this paper, we proposed a family of diffusion layers which are constructed using some rounds of Feistel-like structures whose round functions are linear. These diffusion layers are called recursive diffusion layers. First, for a fixed structure, we determined the required conditions for its underlying linear function to make it a perfect diffusion layer. Then, for the number of words in input (output) less than 8, we extended our approach and found all the instances of the perfect recursive diffusion layers with the general form of Fig. 1. Also, we proposed some other diffusion layers with non-regular forms which can be used for the design of lightweight block ciphers. Finally, diffusion layers with 2 linear functions were proposed. By using two linear functions, we designed perfect recursive diffusion layers for $s = 5, 6, 7, 8$ which cannot be designed based on Fig. 1, i.e, using only one linear function.

The proposed diffusion layers have simple inverses, thus they can be deployed in SPN structures. These proposed diffusion layers can be used to improve the security or performance of some of the current block ciphers and hash functions or in the design of the future block ciphers and hash functions (especially the block ciphers with provable security against DC and LC).

References

1. D.J. Bernstein. The Salsa20 Stream Cipher, 2005. http://www.ecrypt.eu.org/stream/salsa20p2.html.
2. E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In *CRYPTO'90*, volume 537, pages 2–21. Springer-Verlag, 1990.

3. J. Daemen. *Cipher and Hash Function Design Strategies Based on Linear and Differential Cryptanalysis.* PhD thesis, Elektrotechniek Katholieke Universiteit Leuven, Belgium, 1995.
4. J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Springer-Verlag, 2002.
5. J. Guo, T. Peyrin, and A. Poschmann. The PHOTON Family of Lightweight Hash Functions. In *CRYPTO'11*, volume 6841, pages 222–239. Springer-Verlag, 2011.
6. J. Guo, T. Peyrin, and M. Robshaw. Provable Security for an RC6-like Structure and a MISTY-FO-like Structure Against Differential Cryptanalysis. In *ICCSA'06*, volume 3982, pages 446–455. Springer-Verlag, 2006.
7. J. Guo, T. Peyrin, and M. Robshaw. The LED Block Cipher. In *CHES'11*, volume 6917, pages 326–341. Springer-Verlag, 2011.
8. Nyberg K. and L. Knudsen. Provable Security Against a Differential Attack. *Journal of Cryptology*, 8(1):27–37, 1995.
9. J. Kang, S. Hong, S. Lee, O. Yi, C. Park, and J. Lim. Practical and Provable Security Against Differential and Linear Cryptanalysis for Substitution-Permutation Networks. *ETRI Journal*, 23(4):158–167, 2001.
10. M. Matsui. Linear Cryptanalysis Method for DES Cipher. In *EUROCRYPT'93*, volume 765, pages 386–397. Springer-Verlag, 1993.
11. M. Matsui. New Structure of Block Ciphers with Provable Security Against Differential and Linear Cryptanalysis. In *FSE'96*, volume 1039, pages 205–218. Springer-Verlag, 1996.
12. K. Ohkuma, H. Muratani, F. Sano, and S. Kawamura. The Block Cipher Hierocrypt. In *SAC'01*, volume 2012, pages 72–88. Springer-Verlag, 2001.
13. M. Sajadieh, M. Dakhilalian, and H. Mala. Perfect Involutory Diffusion Layers Based on Invertibility of Some Linear Functions. *IET Information Security Journal*, 5(1):228–236, 2011.
14. M. Wang, J. Nakahara, and Y. Sun. Cryptanalysis of the Full MMB Block Cipher. In *SAC'09*, volume 5867, pages 231–248. Springer-Verlag, 2009.
15. H. Wu. The Hash Function JH, 2008. http://icsd.i2r.astar.edu.sg/staff/hongjun/jh/jh.pdf.

Unaligned Rebound Attack:
Application to Keccak

Alexandre Duc[1,*], Jian Guo[2,†], Thomas Peyrin[3,‡], and Lei Wei[3,§]

[1] Ecole Polytechnique Fédérale de Lausanne, Switzerland
[2] Institute for Infocomm Research, Singapore
[3] Nanyang Technological University, Singapore
alexandre.duc@epfl.ch
{ntu.guo,thomas.peyrin}@gmail.com
wl@pmail.ntu.edu.sg

Abstract. We analyze the internal permutations of Keccak, one of the NIST SHA-3 competition finalists, in regard to differential properties. By carefully studying the elements composing those permutations, we are able to derive most of the best known differential paths for up to 5 rounds. We use these differential paths in a rebound attack setting and adapt this powerful freedom degrees utilization in order to derive distinguishers for up to 8 rounds of the internal permutations of the submitted version of Keccak. The complexity of the 8 round distinguisher is $2^{491.47}$. Our results have been implemented and verified experimentally on a small version of Keccak. This is currently the best known differential attack against the internal permutations of Keccak.

Key words: Keccak, SHA-3, hash function, differential cryptanalysis, rebound attack.

1 Introduction

Cryptographic hash functions are used in many applications such as digital signatures, authentication schemes or message integrity and they are among the most important primitives in cryptography. Informally, a hash function H is a function that takes an arbitrarily long message as input and outputs a fixed-length hash value of size n bits. Even if hash functions are traditionally used to simulate the behavior of a random oracle [3], classical security requirements are collision resistance and (second)-preimage resistance. Namely, it should be impossible for an adversary to find a collision (two distinct messages that lead to the same hash value) in less than $2^{n/2}$ hash computations, or a (second)-preimage (a message hashing to a given challenge) in less than 2^n hash computations. Of course, in the ideal case an attacker should also not be able to distinguish the hash function from a random oracle.

Recently, most of the standardized hash functions [25, 20] have suffered from serious collision attacks [29, 28]. As a response the NIST launched in 2007 the SHA-3 competition [21] that will lead to the future hash function standard. 5 candidates made it to the final round, and Keccak [9] is among them. Compared to its opponents, this hash function presents the particularity to be a sponge function [5]. The submitted versions of Keccak to the SHA-3 competition use as main component an internal permutation P of 1600 bits. In the original submission [6] the internal permutation used 18 rounds and the tweaked versions [7] went up to 24 rounds.

[*] Part of the work was done while the author was visiting Nanyang Technological University, supported by the NTU NAP Startup Grant M58110000.

[†] Part of the work was done while the author was visiting Tsinghua University, supported by the National Natural Science Foundation of China under grant No. 61133013 and No. 60931160442.

[‡] The author is supported by the Lee Kuan Yew Postdoctoral Fellowship 2011 and the Singapore National Research Foundation Fellowship 2012.

[§] The author is supported by the Singapore National Research Foundation under Research Grant NRF-CRP2-2007-03, the Singapore Ministry of Education under Research Grant T206B2204 and by the NTU NAP Startup Grant M58110000.

Like any construction that builds a hash function from a subcomponent, the cryptographic quality of this internal permutation is very important for a sponge construction. Therefore, this permutation P should not present any structural flaw, or should not be distinguishable from a randomly chosen permutation. Previous cryptanalysis have not endangered the Keccak security so far. Zero-sum distinguishers [2] can reach an important number of rounds, but generally with a very high complexity. For example, the latest results [11] provide zero-sum partitions distinguishers for the full 24-round 1600-bit internal permutation with a complexity of 2^{1575}. When looking at smaller number of rounds the complexity would decrease, but it is unclear how one can describe the partition of a 1600-bit internal state without using the Keccak round inside the definition of the partition. Moreover, such zero-sum properties seem very hard to exploit when the attacker aims at the whole hash function. On the other side, more classical preimage attack on 3 rounds using SAT-solvers have been demonstrated [19]. Finally, Bernstein published [4] a $2^{511.5}$ computations (second)-preimage attack on 8 rounds that allows a workload reduction of only half a bit over the generic complexity with an important memory cost of 2^{508}.

Our contributions. In this article, we analyze the differential cryptanalysis resistance of the Keccak internal permutation. More precisely, we first introduce a new and generic method that looks for good differential paths for all the Keccak internal permutations, and we obtain the currently best known differential paths. We then describe a simple method to utilize the available freedom degrees which allows us to derive distinguishers for reduced variants of the Keccak internal permutations with low complexity. Finally, we apply the idea of rebound attack [18] to Keccak. This application is far from being trivial and requires a careful analysis of many technical details in order to model the behavior of the attack. This technique is in particular much more complicated to apply to Keccak than to AES or to other 4-bit Sbox hash functions [24, 14]. One reason for that is that Keccak has *weak alignment* [8]. This is why we call our attack "unaligned rebound attack". The model introduced has been verified experimentally on a small version of Keccak and we eventually obtained differential distinguishers for up to 8 rounds of the submitted version of Keccak to the SHA-3 competition. In order to demonstrate why differential analysis is in general more relevant than zero-sum ones in regards to the full hash function, we applied our techniques to the recent Keccak challenges [27] and managed to obtain the currently best known practical collision attack for up to two rounds.

Outline. In Section 2, we first briefly describe the Keccak family of hash functions. We describe our differential path search algorithm in Section 3 and we derive simple differential distinguishers from it in Section 4. We present our theoretical model and we apply the rebound attack on Keccak in Section 5. Finally, we present our results and draw conclusions in Section 6.

2 The Keccak Hash Function Family

Keccak [9, 10] is a family of variable length output hash functions based on the sponge construction [5]. In Keccak family, the underlying function is a permutation chosen from a set of seven Keccak-f permutations, denoted as Keccak-$f[b]$ where $b \in \{1600, 800, 400, 200, 100, 50, 25\}$ is the permutation width as well as the internal state size of the hash function. The Keccak family is parametrized by an r-bit bitrate and c-bit capacity with $b = r + c$.

2.1 The Keccak-f permutations

The internal state of the Keccak family can be viewed as a bit array of 5×5 lanes, each of length $w = 2^\ell$ where $\ell \in \{0, 1, 2, 3, 4, 5, 6\}$ and $b = 25w$. The state can also be described as a three dimensional array of bits defined by $a[5][5][w]$. A bit position (x, y, z) in the state is given by $a[x][y][z]$ where x and y coordinates are taken over modulo 5 and the z coordinate is taken over modulo w. A *lane* of the internal state at *column* x and *row* y is represented by $a[x][y][\cdot]$, while a *slice* of the internal state at *width* z is represented by $a[\cdot][\cdot][z]$.

Keccak-$f[b]$ is an iterated permutation consisting of a sequence of n_r rounds indexed from 0 to $n_r - 1$ and the number of rounds are given by $n_r = 12 + 2\ell$. A round $\mathbf{R}$ consists of a transformation of five step mappings and is defined by: $\mathbf{R} = \iota \circ \chi \circ \pi \circ \rho \circ \theta$. These step mappings are discussed below.

$\boldsymbol{\theta}$ mapping. This linear mapping intends to provide diffusion for the state and is defined for every x, y and z by: $\theta : a[x][y][z] \leftarrow a[x][y][z] + \bigoplus_{y'=0}^{4} a[x-1][y'][z] + \bigoplus_{y'=0}^{4} a[x+1][y'][z-1]$.
That is, the bitwise sum of the two columns $a[x-1][\cdot][z]$ and $a[x+1][\cdot][z-1]$ is added to each bit $a[x][y][z]$ of the state.

$\boldsymbol{\rho}$ mapping. This linear mapping intends to provide diffusion between the slices of the state through intra-lane bit translations. For every x, y and z: $\rho : a[x][y][z] \leftarrow a[x][y][z + T(x, y)]$, where $T(x, y)$ is a translation constant. That is, all bit positions in each lane are translated by a constant amount that depends on the column x and row y considered.

$\boldsymbol{\pi}$ mapping. This linear mapping intends to provide diffusion in the state through transposition of the lanes. More precisely, it is defined for every x, y and z as: $\pi : a[x'][y'][z] \leftarrow a[x][y][z]$, with $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$.
Since this results in transposition of bits into a same slice, this mapping is an intra-slice transposition.

$\boldsymbol{\chi}$ mapping. This is the only non-linear mapping of Keccak-f and is defined for every x, y and z by: $\chi : a[x][y][z] \leftarrow a[x][y][z] + ((\neg a[x+1][y][z]) \wedge a[x+2][y][z])$.
This mapping is similar to an Sbox applied independently to each 5-bit row of the state and can be computed in parallel to all rows. We represent by $s = 5w$ the number of Sboxes/rows in Keccak internal state. Here $\neg$ denotes bit-wise complement, and $\wedge$ the bit-wise AND.

$\boldsymbol{\iota}$ mapping. For every round $\mathbf{R}$ of the Keccak-f permutation, this mapping adds constants derived from an LFSR (see [9] for details) to the lane $a[0][0][\cdot]$. These constants are different in every round i: $\iota : a[0][0][\cdot] \leftarrow a[0][0][\cdot] + \mathrm{RC[i]}$.
This mapping aims at destroying the symmetry introduced by the identical nature of the remaining mappings in every round of the Keccak-f permutation.

We refer to the Keccak specifications document [9] for all the translation and round constants.

3 Finding Differential Paths for Keccak-f Internal Permutations

Before describing how we use the freedom degrees in a rebound attack setting, we first study how to find "good" differential paths for all Keccak variants. In this section, we describe our differential finding algorithms. We start by recalling several special properties of the mappings θ and χ in the round function, followed by our algorithm which provides most of the best known differential paths for the Keccak internal permutations. In particular, we provide the currently best known 3, 4 and 5-round differential paths for Keccak-$f[1600]$, the internal permutation from the submitted version of Keccak.

3.1 Special properties of θ and χ

It is noted by the KECCAK designers [9, Section 2.4.3] that when every column of the state sums ot 0, θ acts as identity. The set of such states is called *column parity kernel* (*CP-kernel*). Since θ is linear, this property applies not only to the state values, but also to differentials. While θ expands a single bit difference into at most 11 bits (2 columns and the bit itself), it acts as identity on differences in the CP-kernel. This property will be intensively used in finding low Hamming weight bitwise differentials. Another interesting property is that θ^{-1} diffuses much faster than θ, i.e., a single bit difference can be propagated to about half state bits through θ^{-1} [9, Section 2.3.2]. However, the output of θ^{-1} is extremely regular when the Hamming weight of the input is low.

The χ layer updates is a row-wise operation and can also be viewed as a 5-bit Sbox. Similar to the analysis of other Sboxes, we build its differential distribution table (DDT). We remark that when a single difference is present, χ acts as identity with best probability 2^{-2}, while input differences with more active bits tend to lead to more possible output differences, but with lower probability. It is also interesting to note that given an input difference to χ, all possible output differences occur with same probability (however this is not the case for χ^{-1}).

3.2 First tools

Our goal is to derive "good" bitwise differential paths by maintaining the bit difference Hamming weight as low as possible. The ι permutation adds predefined constants to the first lane, and hence has no essential influences when such differentials are considered. For the rest of the paper, we will ignore this layer. We note that θ, ρ and π are all linear mappings, while χ acts as a non-linear Sbox. Furthermore, ρ and π do not change the number of active bits in a differential path, but only bit positions. Hence, θ and χ are critical when looking for a "good" differential path. Since χ is followed by θ in the next round (ignoring ι), we consider these two mappings together by treating a slice of the state as a unit, and try to find the potential best mapping of the slice through χ with the following two rules.

1. Given an input difference of the slice, i.e., 5 row differences, find all possible output differences by looking into the DDT table. Then among all combinations of solutions of the 5 rows, choose the output combinations with minimum number of columns with odd parity.
2. In case of a draw, we select the state with the minimum number of active bits.

Rule 1 aims at reducing the amount of active bits after applying θ by choosing each slice of the output of the χ closest to the CP-kernel (i.e., with even parity for most columns), and rule 2 further reduces the amount of active bits within the columns. Although this strategy may not lead to the minimum number of active bits after θ in the entire state (the full KECCAK-f[1600] state is too large to precompute the best mappings for the whole state), it finds the best slice-wise mappings with help of a table of size 2^{25} (tricks like removing the ordering of the rows reduce the table size to about 2^{18}).

3.3 Algorithm for differential path search

Denote $\lambda = \pi \circ \rho \circ \theta$ (all linear mappings), and the state at round i before (resp. after) applying the linear layer λ as a_i (resp. b_i). We start our search from a_1, i.e., the input state to the second round, and compute backwards for one round, and few rounds forwards, as shown below.

$$a_0 \xleftarrow{\lambda^{-1}} b_0 \xleftarrow{\chi^{-1}} \mathbf{a_1} \xrightarrow{\lambda} b_1 \xrightarrow{\chi} a_2 \xrightarrow{\lambda} b_2 \xrightarrow{\chi} a_3 \xrightarrow{\lambda} b_3 \cdots$$

The forward part is longer than the backward part because the diffusion of θ^{-1} is much better than for θ, thus, it will be easier for us to control the bit differences Hamming weight for several forward rounds (instead of backward rounds).

We choose a_1 from the CP-Kernel. Since it is impossible to enumerate all combinations, we further restrict to a subset of the CP-Kernel with at most 8 active bits and each column having exactly 0 or 2 active bits. Note also that any bitwise differential path is invariant through position rotation along the z axis, so we have to run through a set of size about 2^{36}. A brute-force search on this set using our two rules stated previously finds 3-round differential paths with probability 2^{-32}, 4-round differential paths with probability 2^{-142} and 5-round paths with probability 2^{-709} for KECCAK-f[1600]. An example of 4 round path is given in the full version of the paper. We provide also in Table 1 all the best differential path probabilities found for all KECCAK internal permutation sizes.

Table 1. Best differential path results for each version of KECCAK internal permutations, for 1 up to 5 rounds. The detailed differential paths for KECCAK-f[1600] are shown in the full version of the paper. Paths in bold are new results we found with the method presented in this paper.

b	best differential path probability (successive differential complexity of the rounds)				
	1 rd	2 rds	3 rds	4 rds	5 rds
100	2^{-2} (2)	2^{-8} (4 - 4)	2^{-19} (4 - 8 - 7)	2^{-30} (4 - 8 - 10 - 8)	2^{-54} (4 - 8 - 10 - 8 - 24)
200	2^{-2} (2)	2^{-8} (4 - 4)	2^{-20} (4 - 8 - 8)	2^{-46} (11 - 9 - 8 - 8)	2^{-108} (8 - 16 - 20 - 16 - 48)
400	2^{-2} (2)	2^{-8} (4 - 4)	2^{-24} (8 - 8 - 8)	$\mathbf{2^{-84}}$ (16 - 14 - 12 - 42)	2^{-216} (16 - 32 - 40 - 32 - 96)
800	2^{-2} (2)	2^{-8} (4 - 4)	$\mathbf{2^{-32}}$ (4 - 4 - 24)	$\mathbf{2^{-109}}$ (12 - 12 - 12 - 73)	2^{-432} (32 - 64 - 80 - 64 - 198)
1600	2^{-2} (2)	2^{-8} (4 - 4)	$\mathbf{2^{-32}}$ (4 - 4 - 24)	$\mathbf{2^{-142}}$ (12 - 12 - 12 - 106)	$\mathbf{2^{-709}}$ (16 - 16 - 16 - 114 - 547) A better path (2^{-510}) was found independently [23]

4 Simple Distinguishers for the Reduced KECCAK Internal Permutations

Once the differential paths obtained, we can concentrate our efforts on how to use at best the freedom degrees available in order to reduce the complexity required to find a valid pair for the differential trails or to increase the amount of rounds attacked. We present in this section a very simple method that allows to obtain low complexity distinguishers on a few rounds of the KECCAK internal permutations.

4.1 A very simple freedom degrees fixing method

We first describe an extremely simple way of using the available freedom degrees, which are exactly the b-bit value of the internal state (since we already fixed the differential path). For all the best differential paths found from Table 1, we can extend them by one round to the left or to the right, by simply picking some valid Sboxes differential transitions. Obviously, this is going to add a lot of new constraints because the number of active Sboxes will explode in this newly added round and it will force the differential probability to be very low overall. However, we can use our available freedom degrees specifically for this round so that its cost is null. One simply handles each of the active Sboxes differential transitions for this round one by one, independently, by fixing a valid value for the active Sboxes. In terms of freedom degrees consumption for this extra round, in the worst case we have all s Sboxes active and a differential transition probability of 2^{-4} for each of them. Thus, we are ensured to have at least $2^{5s-4s} = 2^s$ freedom degrees remaining after handling this extra round.

Note that some more involved freedom degree methods (such as message modification [28]) might even allow to also control some of the conditions of the original differential path, thus further reducing the complexity.

4.2 The generic case

At the present time, we are able to find valid pairs of internal state values for some differential paths on a few rounds with a rather low complexity. Said in other words, we are able to compute internal state value pairs with a predetermined input/output difference. A direct application from this is to derive distinguishers. For a randomly chosen permutation of b bits, finding a pair of inputs with a predetermined difference that maps to a predetermined output difference costs 2^b computations. Indeed, since the input and the output differences are fixed, the attacker can not apply any birthday-paradox technique. Those distinguishers are called "limited-birthday distinguishers" and can be generalized in the following way (we refer to [12] for more details): for a randomly chosen b-bit permutation, the problem of mapping an input difference from a subset of size I to an output difference from a subset of size J requires $\max\{\sqrt{2^b/J}, 2^b/(I \cdot J)\}$ calls to permutation (while assuming without loss of generality since we are dealing with a permutation that $I \leq J$).

Using the freedom degrees technique from the previous section and reading Table 1, we are for example able to obtain a distinguisher for 5 rounds of the Keccak-f[1600] internal permutation with complexity 2^{142} (while the generic case is 2^{1600}).

4.3 Extending the differential path

Since for many of our distinguishers, the gap between our attack and the generic case complexity is very big, we can try to reach a few more rounds without increasing the complexity. Indeed, by analyzing how the differences will propagate forward from the output and backward from the input of our differential path, we will be able to determine the size of the possible input differences set and the possible output differences set.

For the forward case (i.e. when adding a round to the right), we start from the fully determined difference on the output of the differential path. We first apply the linear layers θ, ρ and π on this output difference and we obtain the difference mask at the input of χ. Now, for each active Sbox, knowing exactly its input difference, we can check with the DDT from χ that only a subset of the 2^5 possibles output differences can be reached. Therefore, the size Γ^{out} of the set of reachable output differences after applying this extra round is bounded and this bound can be computed exactly using the DDT from χ.

For the backward case (i.e. when adding a round to the left), we start from the fully determined difference on the input of the differential path. Then, reading at the DDT from χ^{-1}, one can check that one active Sbox can produce at most a certain small subset of the 2^5 possible input differences. Therefore, the size Γ^{in} of the set of reachable input differences after inverting this χ layer is bounded and this bound can be computed exactly using the DDT from χ^{-1}. Note that continuing to invert the extra round by computing θ^{-1}, ρ^{-1} and π^{-1} will not modify the size of this set.

To conclude, using a r-round path from Table 1 with differential probability p, we extend it by one more round in order to find valid internal state pairs for this new $(r+1)$-round differential path with p^{-1} computations (see Section 4.1). Then, using the limited-birthday distinguishers, one can derive a $(r+3)$-round distinguisher for the Keccak internal permutation with complexity p^{-1}, if $p^{-1} < \max\{\sqrt{2^b/J}, 2^b/(I \cdot J)\}$, where $I = \Gamma^{\mathsf{out}}$ and $J = \Gamma^{\mathsf{in}}$ if $\Gamma^{\mathsf{out}} \leq \Gamma^{\mathsf{in}}$; $I = \Gamma^{\mathsf{in}}$ and $J = \Gamma^{\mathsf{out}}$ otherwise. All the distinguishers we obtain with this method are summarized in Table 2.

Note that the reader might be concerned by the fact that the sizes Γ^{in} and Γ^{out} of the reachable differences sets can be very big and might be not easy to describe in a compact way in our distinguisher. However, we emphasize that all the reachable differences on the output (resp. input) are actually built from the independent combinations of all the possible output differences (resp. input differences) of all active Sboxes in the last round (resp. first round). Therefore, the description of this set is easily done by identifying the reachable output differences (resp. input differences) for all the Sboxes independently.

5 The Rebound Attack on Keccak

The rebound attack is a freedom degrees utilization technique that was first proposed by Mendel et al. in [18] as an analysis of round-reduced Grøstl and Whirlpool. It was then improved in [17, 16, 12, 26] to analyze AES and AES-like permutations and also ARX ciphers [15].

With the help of rebound techniques, we show in this section how to extend the number of attacked rounds significantly, but for a higher complexity. We will see that the application of the rebound attack for Keccak seems quite difficult. Indeed, the situation for Keccak is not as pleasant as the AES-like permutations case where the utilization of truncated differential paths (i.e. path for which one only checks if one cell is active or inactive, without caring about the actual difference value) makes the application of rebound attacks very easy to handle.

5.1 The original rebound attack

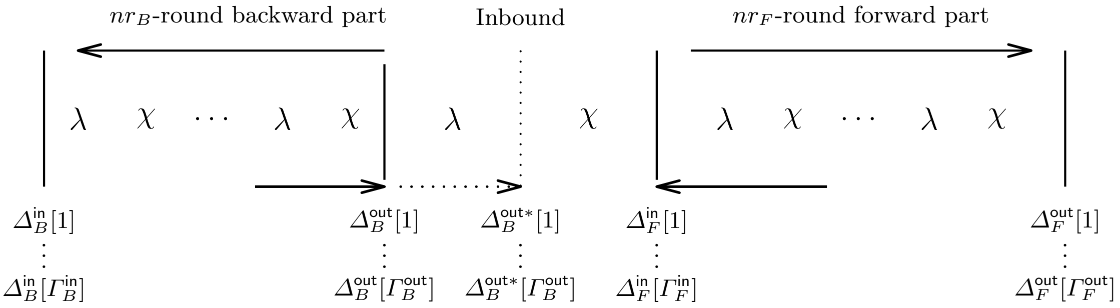
Let P denote a permutation, which can be divided into 3 sub-permutations, i.e., $P = E_F \circ E_I \circ E_B$. The rebound attack works in two phases.

- **Inbound phase or controlled rounds**: this phase usually starts with several chosen input/output differences of E_I that are propagated through linear layers forward and backward. Then, one can carry out meet-in-the-middle (MITM) match for differences through a single Sbox layer in E_I and generate all possible value pairs validating the matches.
- **Output phase or uncontrolled rounds**: With all solutions provided in the inbound phase, check if any pair validates as well the differential paths for both the backward part p_B and the forward part p_F.

The SuperSbox technique [16, 12] extends the E_I from one Sbox layer to two Sbox layers for an AES-like permutation, by considering two consecutive AES-like rounds as one with column-wise SuperSboxes. This technique is possible due to the fact that one can swap few linear operations with the Sbox in AES, so that the two layers of Sboxes in two rounds become close enough to form one SuperSbox layer. However, in the case of Keccak, it seems very hard to form any partition into independent SuperSboxes. For the same reason, using truncated differential paths seems very difficult for Keccak, as it has recently been shown in [8].

5.2 Applying the rebound attack for Keccak internal permutations

Assume that we know a set of n_B differential trails (called *backward trails*) on nr_B Keccak rounds and whose DP is higher or equal to p_B. For the moment, we want all these backward paths to share the same input difference mask Δ_B^{in} and we denote by $\Delta_B^{\mathsf{out}}[i]$ the output difference mask of the i-th trail of the set. Similarly, we consider that we also know a set of n_F differential trails (called *forward trails*) on nr_F Keccak rounds and whose DP is higher or equal to p_F. We want all those forward paths to share the same output difference mask Δ_F^{out} and we denote by $\Delta_F^{\mathsf{in}}[i]$ the input difference mask of the i-th trail of the set.



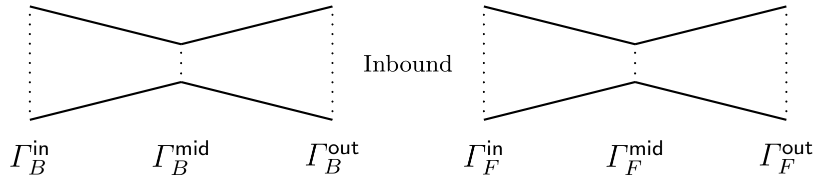
nr_B-round backward part
Inbound
nr_F-round forward part
λ
χ
$\cdots$
λ
χ
λ
χ
λ
χ
$\cdots$
λ
χ
$\Delta_B^{\mathsf{in}}[1]$
$\Delta_B^{\mathsf{in}}[\Gamma_B^{\mathsf{in}}]$
$\Delta_B^{\mathsf{out}}[1]$
$\Delta_B^{\mathsf{out}}[\Gamma_B^{\mathsf{out}}]$
$\Delta_B^{\mathsf{out}*}[1]$
$\Delta_B^{\mathsf{out}*}[\Gamma_B^{\mathsf{out}}]$
$\Delta_F^{\mathsf{in}}[1]$
$\Delta_F^{\mathsf{in}}[\Gamma_F^{\mathsf{in}}]$
$\Delta_F^{\mathsf{out}}[1]$
$\Delta_F^{\mathsf{out}}[\Gamma_F^{\mathsf{out}}]$

Fig. 1. Rebound attack on Keccak

Our goal here is to build a differential path on $nr_B + nr_F + 1$ Keccak rounds (thus one Sbox layer of inbound), by connecting a forward and a backward trail with the rebound technique, and eventually to find the corresponding solution values for the controlled round. We represent by p_{match} the probability that a match is possible from a given element of the backward set and a given element of the forward set, and we denote by N_{match} the number of solution values that can be generated once a match has been obtained.

For this connection to be possible, we need the inbound phase to be a valid differential path, that is we need to find a valid differential path from a $\Delta_B^{\mathsf{out}*}$ to a Δ_F^{in}. By using random $\Delta_B^{\mathsf{out}*}$ and Δ_F^{in} this will happen in general with very small probability, because we need the very same set of Sboxes to be active/inactive in both forward and backward difference masks to have a chance to get a match. Even if the set of active Sboxes matches, we still require the differential transitions through all the active Sboxes to be possible.

We can generalize a bit this approach by allowing a fixed set of differences Δ_B^{in} (resp. Δ_F^{out}) instead of just one. We call Γ_B^{in} (resp. Γ_B^{out}) the *size* of the set of possible Δ_B^{in} (resp. Δ_B^{out}) values for the backward paths. Similarly, we call Γ_F^{in} (resp. Γ_F^{out}) the size of the set of possible Δ_F^{in} (resp. Δ_F^{out}) values for the forward paths. In fact, the number of possible differences in the backward or forward parts will form a butterfly shape (see Figure 2). We call Γ_B^{mid} (resp. Γ_F^{mid}) the minimum number of differences in the backward (resp. forward) part.



Inbound
Γ_B^{in}
Γ_B^{mid}
Γ_B^{out}
Γ_F^{in}
Γ_F^{mid}
Γ_F^{out}

Fig. 2. Number of differences for the rebound attack on Keccak.

The total complexity C to find one valid internal state pair for the $(nr_B + nr_F + 1)$-round path is

$$C = n_F + n_B + \frac{1}{p_{\mathsf{match}}} \cdot \left\lceil \frac{1}{p_F \cdot p_B \cdot N_{\mathsf{match}}} \right\rceil + \frac{1}{p_B \cdot p_F} \ , \tag{1}$$

with

$$\Gamma_B^{\mathsf{out}} \cdot \Gamma_F^{\mathsf{in}} = \frac{1}{p_{\mathsf{match}}} \cdot \left\lceil \frac{1}{p_F \cdot p_B \cdot N_{\mathsf{match}}} \right\rceil \ . \tag{2}$$

The first two terms are the costs to generate the backward and forward paths. The term $\lceil \frac{1}{p_F \cdot p_B \cdot N_{\mathsf{match}}} \rceil$ denotes the number of time we will need to perform the inbound and each inbound costs $1/p_{\mathsf{match}}$. The last term is the cost for actually performing the outbound phase. The condition (2) is needed since we need enough differences to perform the inbound phase.

Roadmap. For a better understanding of the behavior of the Sboxes in the rebound attack, we will introduce some useful lemmas in Section 5.3. We explain how to prepare the forward and backward differential paths in Section 5.4 and describe the inbound and outbound phases in Section 5.5 and 5.6 respectively. We explain how to relate Sections 5.4, 5.5 and 5.6 in Section 5.7, we show also how we can reduce the complexity of the attack and we give a numerical application of our model. Finally we construct distinguishers from the differential paths in Section 5.8.

5.3 An Ordered Buckets and Balls Problem

We model the active/inactive Sboxes match as a **limited capacity ordered buckets and balls problem**:the $s = 5w$ ordered buckets ($s = 320$ for KECCAK-$f[1600]$) limited to capacity 5 will represent the s 5-bit Sboxes and the x_B (resp. x_F) balls will stand for the Hamming weight of the difference in $\Delta_B^{\mathsf{out}*}$ (resp. in Δ_F^{in}). Given a set B of s buckets in which we randomly throw x_B balls and a set F of s buckets in which we randomly throw x_F balls, we call the result a **pattern-match** when the set of empty buckets in B and F after the experiment are the same.[4] Before computing the probability of having a pattern-match, we need the following lemma.

Lemma 1. *The number of possible combinations* $b_{\mathsf{bucket}}(n, s)$ *to place* n *balls into* s *buckets of capacity* 5 *such that no bucket is empty is*

$$b_{\mathsf{bucket}}(n,s) := \begin{cases} \displaystyle\sum_{i=\lceil n/5\rceil}^{s} (-1)^{s-i}\binom{s}{i}\binom{5i}{n} & \textit{if } s \le n \le 5s \\ 0 & \textit{else.} \end{cases} \tag{3}$$

The proof of this lemma is given in the full version of the paper.

Using (3), we can derive the probability $p_{\mathsf{bucket}}(n, s)$ that every bucket contains at least one ball when n balls are thrown into s buckets with capacity 5 and the expected number of active buckets when n balls are thrown into s buckets. We can now relate this lemma to the more general pattern-match problem. This model tells us that when the number of balls (i.e., active bits) is not too small on both sides, most of the matches happen when (almost) all the Sboxes are active. We analyze this behavior in more details in the full version of the paper.

A More General Problem. We can also look into a more general problem, i.e., we characterize more precisely how the bits are distributed into the Sboxes.

Lemma 2. *The probability* p_{dist} *of distributing randomly* n *active bits into* s 5*-bit Sboxes such that exactly* A_i *Sboxes contain* i *bits, for* $i \in [1, 5]$ *is*

$$p_{\mathsf{dist}}(A_1, A_2, A_3, A_4, A_5) := \frac{s!\binom{5}{1}^{A_1}\binom{5}{2}^{A_2}\binom{5}{3}^{A_3}\binom{5}{4}^{A_4}\binom{5}{5}^{A_5}}{(s - A_1 - A_2 - A_3 - A_4 - A_5)!A_1!A_2!A_3!A_4!A_5!\binom{5s}{n}}\,, \tag{4}$$

with $n = A_1 + 2A_2 + 3A_3 + 4A_4 + 5A_5$.

Important Remark. Since most matches happen when all the Sboxes are active, in order to simplify the analysis, we will use from now on only forward and backward paths such that *all Sboxes are active in the* χ *layer of the inbound phase.*

[4] Note that the *position* of the balls in the buckets is significant. This is why we refer to an ordered buckets and balls problem.

5.4 The differential paths sets

In this section, we explain how we generate the forward and backward paths, since this will have an impact on the derivation of p_{match} and N_{match} (this will be handled in the next two sections).

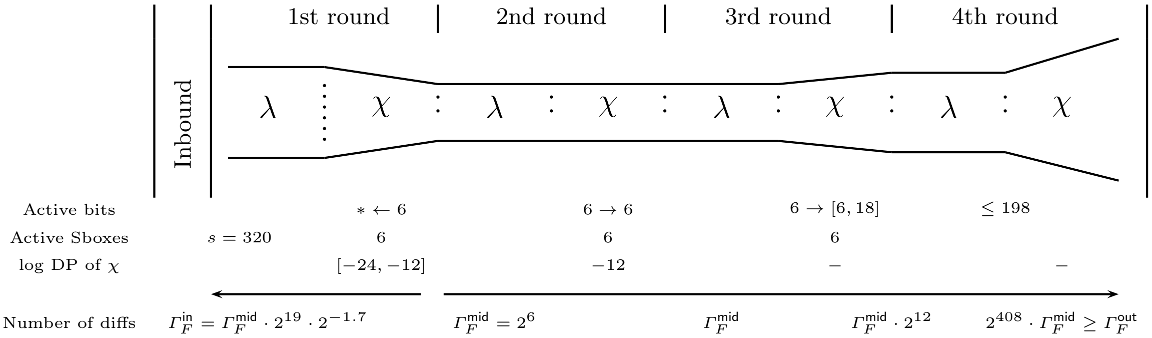
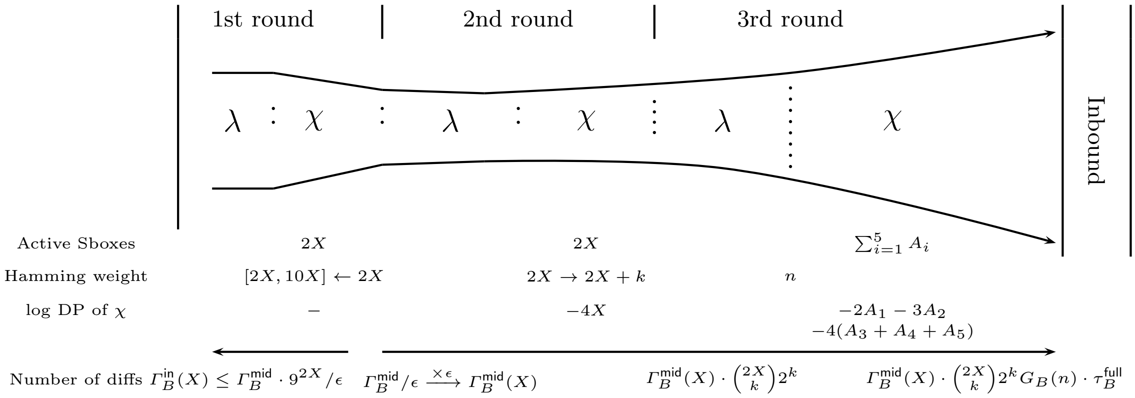


Fig. 3. The forward trails we are using. The distance between the two lines reflects the number of differences.

The forward paths. For the forward paths set (see Fig. 3), we start by choosing a differential trail computed from the previous section and we derive a set from it by exhausting all the possible Sbox differential transitions for the inverse of the χ layer in its first round (all the paths will be the same except the differences on their input and on the input of the χ layer in the first round). For example, we can use the 2 first rounds of the 4-round differential path we found (see full version) which have a total success probability 2^{-24} and present 6 active Sboxes during the χ layer of the first round. We randomize the χ^{-1} layer differential transitions for the 6 active Sboxes of the first round, and we obtain about 2^{19} distinct trails in total. We analyzed that all the trails of this set have a success probability of at least $2^{-24} \cdot 2^{-2 \cdot 6} = 2^{-36}$ (this is easily obtained with the χ^{-1} DDT). Moreover, note that they will all have the same output difference mask (at the third round), but distinct input masks (at the first round). Since we previously forced the requirement that all Sboxes must be active for the inbound match, we check experimentally that $2^{17.3}$ of the 2^{19} members of the set fulfill this condition.[5] We call τ_F the ratio of paths that verify this condition over the total number of paths, i.e., $\tau_F = 2^{-1.7}$. Overall, we built a set of $2^{17.3}$ forward differential paths on $nr_F = 2$ Keccak-f[1600] rounds, all with DP higher or equal to $p_F = 2^{-36}$. We can actually generate 64 times more paths by observing that they are equivalent by translation along the Keccak lane (the z axis). However, these paths will have distinct output difference masks (the same difference mask rotated along the z axis), and we have $\Gamma_F^{\mathsf{mid}} = 2^6$. The *total amount of input differences* is $\Gamma_F^{\mathsf{in}} := \Gamma_F^{\mathsf{mid}} \cdot 2^{17.3} = 2^{23.3}$ and we have to generate in total $n_F = \tau_F \cdot \Gamma_F^{\mathsf{in}} = 2^{25}$ forward differential paths. We discuss the amount of output differences in Section 5.8, since we extend there the path with two free additional rounds.

The backward paths. Applying the same technique to the backward case does not generate a sufficient amount of output differences Γ_B^{out}, crucial for a rebound-like attack. Thus, concerning the backward paths set, we build another type of 2-round trails. We need first to ensure that we have *enough differential paths* to be able to find a match in the inbound phase, i.e., we want $\Gamma_B^{\mathsf{out}} \cdot \Gamma_F^{\mathsf{in}} = 1/p_{\mathsf{match}} \cdot \lceil \frac{1}{p_F \cdot p_B \cdot N_{\mathsf{match}}} \rceil$ following (2). Moreover, we will require these paths to verify two conditions:

[5] The small amount of filtered forward paths (a factor $2^{1.7}$) is due to the regularity of the output of θ inverse. Thus, most of the paths have all Sboxes active when the Hamming weight of the input is low.



1st round
2nd round
3rd round
λ
χ
λ
χ
λ
χ
Inbound
Active Sboxes
$2X$
$2X$
$\sum_{i=1}^{5} A_i$
Hamming weight
$[2X, 10X] \leftarrow 2X$
$2X \rightarrow 2X + k$
n
log DP of χ
–
$-4X$
$-2A_1 - 3A_2$
$-4(A_3 + A_4 + A_5)$
Number of diffs $\Gamma_B^{\mathsf{in}}(X) \leq \Gamma_B^{\mathsf{mid}} \cdot 9^{2X}/\epsilon$
$\Gamma_B^{\mathsf{mid}}/\epsilon \xrightarrow{\times\epsilon} \Gamma_B^{\mathsf{mid}}(X)$
$\Gamma_B^{\mathsf{mid}}(X) \cdot \binom{2X}{k} 2^k$
$\Gamma_B^{\mathsf{mid}}(X) \cdot \binom{2X}{k} 2^k G_B(n) \cdot \tau_B^{\mathsf{full}}$

Fig. 4. The backward trails we are using. The distance between the two arrows reflects the number of differences.

1. First, we need to filter paths that have not all Sboxes active in the χ layer of the inbound phase. This happens with a probability about $\tau_B^{\mathsf{full}} := b_{\mathsf{bucket}}(800, 320)/\binom{5\cdot 320}{n} = 2^{-15.9}$ if we assume that about half of the bits are active. This assumption will be verified in our case (and was verified in practice) since our control on the diffusion of the active bits will be reduced greatly.
2. Moreover, *all* the paths we collect should have a *DP of at least p_B such that the number of solutions N_{match} generated in the inbound phase is sufficient.* Indeed, we must have $N_{\mathsf{match}} \geq 1/(p_F \cdot p_B)$ in order to have a good success probability to find one solution for the entire path. We call τ_B^{DP} the probability that a path verifies this property. Hence, we need $p_B \geq p_B^{\min} = 1/(p_F \cdot N_{\mathsf{match}})$. We will show in Section 5.7 that $N_{\mathsf{match}} = 2^{486}$ and we previously showed that $p_F = 2^{-36}$. Hence, $p_B^{\min} = 2^{36-486} = 2^{-450}$.

These two filters induce a ratio $\tau_B := \tau_B^{\mathsf{full}} \cdot \tau_B^{\mathsf{DP}}$ of "good" paths. We have $n_B \cdot \tau_B = \Gamma_B^{\mathsf{out}}$, where n_B is the number of paths we need to generate. Thus, we need to generate $n_B^{\min} := 1/(p_{\mathsf{match}} \cdot \lceil \frac{1}{p_F \cdot p_B \cdot N_{\mathsf{match}}} \rceil \cdot \Gamma_F^{\mathsf{in}} \cdot \tau_B)$ trails to perform the rebound. We will show in Section 5.7 that $p_{\mathsf{match}} = 2^{-491.47}$, that $\lceil \frac{1}{p_F \cdot p_B \cdot N_{\mathsf{match}}} \rceil = 1$ and that $\tau_B = 2^{-15.9}$. We also know that $\Gamma_F^{\mathsf{in}} = 2^{23.3}$. Hence, $n_B^{\min} = 2^{491.47+15.9-23.3} = 2^{484.07}$.

We show now how we generated these paths. Fig. 4 can help the reading. We start at the beginning of the second round by forcing X columns of the internal state to be active and each active column will contain only 2 active bits (thus a total of $2X$ active bits). Therefore, we can generate $\binom{5}{2}^X \cdot \binom{s}{X}$ distinct starting differences and each of them will lead to a distinct input difference of the backward path. Note also that all active columns are in the CP-Kernel and thus applying the θ function on this internal state will leave all bit-differences at the same place. Then, applying the ρ and π layers will move the $2X$ active bits to random locations before the Sbox layer of the second round. If X is not too large, we can assume that for a good fraction of the paths, all active bits are mapped to distinct Sboxes and thus we obtain $2X$ active Sboxes, each with one active bit on its input. We call ϵ this fraction of paths which is given by

$$\epsilon := p_{\mathsf{dist}}(2X, 0, 0, 0, 0)\ , \tag{5}$$

where p_{dist} is given by Lemma 2.[6] We will need to take ϵ into account when we count the total number of paths we can generate. This position in the paths, i.e., after the linear layer of the second round, is the part with the lowest amount of distinct differences. Hence, we call the number of differences at this point $\Gamma_B^{\mathsf{mid}}(X) := \binom{5}{2}^X \cdot \binom{s}{X} \cdot \epsilon$.

[6] Simulations verified this behavior in practice for the parameters we use in our attack.

Looking at the DDT from χ, one can check that with one active input bit in an Sbox, there always exists:

- two distinct transitions with probability 2^{-2} for the Keccak Sbox such that we observe 2 active bits on its output (we call it a $1 \mapsto 2$ transition)
- one single transition with probability 2^{-2} and one single active bit on its output (a $1 \mapsto 1$ transition). This transition is in fact the identity.

We need to define how many $1 \mapsto 1$ and $1 \mapsto 2$ transitions we have to use, since there is a tradeoff between the number of paths obtained and the DP of these paths. Whatever choices we make, we always have that the success probability of this χ transition (in the second round) is 2^{-4X}. Let k be the number of $1 \mapsto 2$ transitions among the $2X$ possible ones. We will observe $2X + k$ active bits after χ. Before the χ transition, we have $\Gamma_B^{\mathsf{mid}}(X)$ different paths from the initial choice. For each of these paths, we can now select $\binom{2X}{k}$ distinct sets of $1 \mapsto 2$ transitions each of which can generate 2^k different paths. These $2X + k$ bits are expanded through θ to *at most* $11 \cdot (2X + k) = 22X + 11k$ bits. However, this expansion factor (every active bit produces 11 one) is smaller when the number of bits increases. Let n be the number of obtained active bits at the input of the Sboxes in the third round. At the beginning of the third round, we have $2X + k$ active bits. For Keccak-$f[1600]$, given $2X + k$ active bits at the input of θ, we get $n \approx u - (u \cdot (u - 1))/1600$ bits at the output, with $u \coloneqq 11(2X + k)$ for X small enough. Indeed, the $2X + k$ bits are first multiplied by 11 due to the property of θ. We suppose now that these u active bits are thrown randomly and we check for collisions. Given u bits, we can form $u \cdot (u - 1)/2$ different pairs of bits. The probability that a pair collides is 2^{-1600}, hence, we have about $u \cdot (u - 1)/(2 \cdot 1600)$ collisions of two bits. In a 2-collision, two active bits are wasted (they become inactive). Hence, we can remove $u \cdot (u - 1)/1600$ from the overall number of active bits. For small X, we can neglect collisions between three, four and five active bits, since the bits before θ are most likely separated and will not collide. Hence, we verify the equation for n. This model has been verified in simulations for the values we are using.

We need now to evaluate the number of active Sboxes in the χ layer of the third round. However, in order to precisely evaluate the DP of this layer (that we want to be higher than $p_B^{\min}$) and the expansion factor we get on the amount of distinct differential paths, we also need to look at how the bits are distributed into the input of the Sboxes. The probability p_{dist} of distributing randomly n active bits into s 5-bit Sboxes such that exactly A_i Sboxes contain i bits, for $i \in [1, 5]$ is given by Lemma 2.

Lemma 3. *Suppose that we have n active bits before χ in the third round. Then, if $n \leq s$, the expected number of* useful *(i.e., which have $\mathsf{DP} \geq p_B^{\min}$) paths $G_B(n)$ we can generate verifies*

$$G_B(n) \geq \sum_{A_5=0}^{\left\lfloor \frac{n}{5} \right\rfloor} \sum_{A_4=0}^{\left\lfloor \frac{(n-5A_5)}{4} \right\rfloor} \sum_{A_3=0}^{\left\lfloor \frac{(n-5A_5-4A_4)}{3} \right\rfloor} \sum_{A_2=0}^{\left\lfloor \frac{(n-5A_5-4A_4-3A_3)}{2} \right\rfloor} F(X, A_1, A_2, A_3, A_4, A_5) \cdot 2^{2A_1+3A_2+3.58A_3+4(A_4+A_5)} , \tag{6}$$

where $A_1 \coloneqq n - 5A_5 - 4A_4 - 3A_3 - 2A_2$ and

$$F(X, A_1, A_2, A_3, A_4, A_5) \coloneqq \begin{cases} p_{\mathsf{dist}}(A_1, A_2, A_3, A_4, A_5) & \textit{if } 2^{-4X-2A_1-3A_2-4(A_3+A_4+A_5)} \geq p_B^{\min} \\ 0 & \textit{else.} \end{cases} \tag{7}$$

Note that we use $F(\dots)$ to filter the paths that have a too low DP.

Proof. Given the number of active input bits in every Sbox, it is easy to compute the number of paths we can generate by looking into the DDT.[7] We find that for an input Hamming weight

[7] We considered the average case here since we already have a lot of paths to start with at the input of the third round.

of 1 (resp. 2), there are always 2^2 (resp. 2^3) possible output differences. For an Hamming weight of 3, half of the input differences can produce 2^3 differences and half 2^4 differences. Hence, the expected value is $2^{3.58}$. For input Hamming weights of 4 and 5, we can always produce 2^4 differences. Thus, the total expected number of paths we can generate when we have A_i Sboxes with an input Hamming weight of i is $2^{2A_1+3A_2+3.58A_3+4(A_4+A_5)}$.

Moreover, we count only the paths that verify $p_B \geq p_B^{\min}$ by discarding all the paths that have a DP smaller than $p_B^{\min}$ using the filter $F(\dots)$. The DP of the complete path is given by

$$2^{-4X-2A_1-3A_2-4(A_3+A_4+A_5)} \, . \tag{8}$$

Indeed, for the second round, we have one active bit per Sbox and, hence, a DP of 2^{-4X}. For the third round, an analysis of the DDT shows that, when we have 1 (resp. 2) active bit in the input, the DP of the SBox is always 2^{-2} (resp. 2^{-3}). For a Hamming weight of 3, there are two different DPs depending on the input. We considered the worst case, which is 2^{-4}. For a Hamming weight of 4 and 5, the DP is always 2^{-4}. Hence, the DP of the complete path verifies (8).

Now, using Lemma 2, we find that the paths occur with probability $p_{\mathsf{dist}}(A_1, A_2, A_3, A_4, A_5)$. Hence, the expected number of paths we will get is the sum of all the probabilities of the path that are not discarded by the filter. □

In practice, we compute $G_B(n)$ by summing over all possible values of $A_1, \dots, A_5$, such that $n = A_1 + 2A_2 + 3A_3 + 4A_4 + 5A_5$.

We have now reached the inbound round and we discard all the paths that do not have all Sboxes active. Hence, we keep only a fraction of $\tau_B^{\mathsf{full}} = 2^{-15.9}$ paths.

It is now easy to see that

$$\tau_B^{\mathsf{DP}} := \sum_{A_5=0}^{\lfloor n/5 \rfloor} \sum_{A_4=0}^{\lfloor (n-5A_5)/4 \rfloor} \sum_{A_3=0}^{\lfloor (n-5A_5-4A_4)/3 \rfloor} \sum_{A_2=0}^{\lfloor (n-5A_5-4A_4-3A_3)/2 \rfloor} F(X, A_1, A_2, A_3, A_4, A_5) \tag{9}$$

with $F(\dots)$ defined in (7) since this is exactly the fraction of path we keep.

To summarize, we have now reached the inbound round and we are able to generate

$$\Gamma_B^{\mathsf{out}} = \epsilon \cdot \binom{5}{2}^X \cdot \binom{s}{X} \cdot \binom{2X}{k} \cdot 2^k \cdot G_B(n) \cdot \tau_B^{\mathsf{full}} \tag{10}$$

differences that have a good DP and all Sboxes active and the total number of paths we have to generate is $n_B = \Gamma_B^{\mathsf{out}} / \tau_B = \Gamma_B^{\mathsf{out}} / (\tau_B^{\mathsf{full}} \cdot \tau_B^{\mathsf{DP}})$.

By playing with the filter bound, we noticed the following behavior. The stronger the filter is (i.e., the higher we set the bound on the DP), the higher the expected value of the Hamming weight at the input of the Sboxes of the inbound phase will be. This behavior will allow us to reduce the complexity of our attack in Section 5.7, where we discuss the numerical application. Hence, instead of filtering at $p_B^{\min}$, we will filter at a higher value to get better results.

Summary. At this point, we started with n_F (resp. n_B) forward (resp. backward) paths from which we kept only Γ_F^{in} (resp. Γ_B^{out}) candidates that have a DP greater than p_F (resp. p_B) and all Sboxes actives during the inbound.

5.5 The inbound phase

Now that we have our forward and backward sets of differential paths, we need to estimate the average probability p_{match} that two trails can match during the inbound phase of the rebound attack. We recall that we already enforced all Sboxes to be active during this match, so p_{match} only takes into account the probability that the differential transitions through all the s Sboxes of the internal state are possible.

A trivial method to estimate p_{match} would be to simply consider an average case on the KECCAK Sbox. More precisely, the average probability that a differential transition is possible through the KECCAK Sbox, given two random non-zero 5-bit input/output differences is equal to $2^{-1.605}$. Thus, one is tempted to derive $p_{\mathsf{match}} = 2^{-1.605 \cdot s}$. However, we observed experimentally that the event of a match greatly depends on the **Hamming weight of the input of the Sboxes** and this can be easily observed from the DDT of the χ layer (for example with an input Hamming weight of one the match probability is $2^{-2.95}$, while for an input Hamming weight of four the match probability is $2^{-0.95}$). Note that *this effect is only strong regarding the input of the Sbox* (i.e. the backward paths), but there is no strong bias on the differential matching probability concerning the output Hamming weight.

Therefore, in order to model more accurately the input Hamming weight effect on the matching event, we first divide the backward paths *depending on their Hamming weight* and treat each class separately. More precisely, we look at each possible input Hamming weight division among the s Sboxes. To represent this division, we only need to look at the number of Sboxes having a specific input Hamming weight (their relative position do not matter). We denote by c_i the number of Sboxes having an input Hamming weight i and we need the following equations to hold $\sum_{i=1}^{5} c_i = s$ since we forced that all Sboxes are active during a match. Moreover, for a Hamming weight value w, we have $\sum_{i=1}^{5} i \cdot c_i = w$. The set of divisions c_i verifying the above mentioned equations is denoted by C_w. The number of possible $5s$-bit vectors satisfying $(c_1, \ldots, c_5)$ (i.e., c_1 Sboxes with 1 active bit, c_2 with 2 etc.) is denoted $b_c(c_1, \ldots, c_5)$ and

$$b_c(c_1, \ldots, c_5) = \frac{s!}{c_1! c_2! \ldots c_5!} \cdot 5^{c_1 + c_4} \cdot 10^{c_2 + c_3} \ . \tag{11}$$

We can now compute the probability of having a match p_{match} depending on the input Hamming weight divisions:

Theorem 1. *The probability p_{match} of having a match is*

$$p_{\mathsf{match}} = \sum_{w=s}^{5s} \Pr[Hw_{\mathsf{total}} = w | \mathsf{full}] \sum_{(c_1,\ldots,c_5) \in C_w} \frac{b_c(c_1, \ldots, c_5)}{b_{\mathsf{bucket}}(w, s)} \prod_{i=1}^{5} \left(\sum_{y \in \{0,1\}^5} \sum_{\substack{v \in \{0,1\}^5: \\ \mathsf{Hw}(v) = i}} \frac{P_{\mathsf{out}}(y) \cdot \mathbb{1}_{\mathsf{DDT}[v][y]}}{\binom{5}{i}} \right)^{c_i} , \tag{12}$$

where $P_{\mathsf{out}}(y)$ is the measured probability distribution of having y at the output of an Sbox when we enforce all Sboxes to be active, $\Pr[\mathsf{Hw}_{\mathsf{total}} = w | \mathsf{full}]$ *is the measured probability distribution of the Hamming weight of the input of the Sboxes when all Sboxes are active, $b_c(\ldots)$ is given by* (11)*, $b_{\mathsf{bucket}}(w, s)$ by Lemma 1 and $\mathbb{1}_{\mathsf{DDT}[v][y]}$ is set to one if the entry $[v][y]$ is non-zero in the DDT of the χ layer and zero otherwise.*[8]

Proof. Let full be the event denoting that all Sboxes are active at the inbound phase. We have

$$p_{\mathsf{match}} := \Pr[\mathsf{match} | \mathsf{full}] = \sum_{w} \Pr[\mathsf{match} | \mathsf{Hw}_{\mathsf{total}} = w, \mathsf{full}] \cdot \Pr[\mathsf{Hw}_{\mathsf{total}} = w | \mathsf{full}] \ .$$

We define $p_{\mathsf{match}}(w) := \Pr[\mathsf{match} | \mathsf{Hw}_{\mathsf{total}} = w, \mathsf{full}]$. We have

$$p_{\mathsf{match}}(w) = \sum_{(c_1,\ldots,c_5) \in C_w} \Pr[\mathsf{match} | (c_1, \ldots, c_5), \mathsf{Hw}_{\mathsf{total}} = w, \mathsf{full}] \cdot \Pr[(c_1, \ldots, c_5) | \mathsf{Hw}_{\mathsf{total}} = w, \mathsf{full}] \ . \tag{13}$$

[8] Note that $\Pr[\mathsf{Hw}_{\mathsf{total}} = w | \mathsf{full}]$ greatly depends on the backward paths we choose and that these paths depends on p_{match}. We explain how to solve this cyclic dependency in Section 5.7.

We easily find that

$$\Pr[(c_1,\ldots,c_5)|\mathsf{Hw}_{\mathsf{total}} = w, \mathsf{full}] = \frac{b_c(c_1,\ldots,c_5)}{b_{\mathsf{bucket}}(w,s)}\ , \tag{14}$$

since $b_c(c_1,\ldots,c_5)$ is the number of possible combinations of vectors verifying $c_1,\ldots,c_5$ and $b_{\mathsf{bucket}}(w,s)$ the number of possible combinations of vectors for which all Sbox are active. It remains to compute

$$\Pr[\mathsf{match}|(c_1,\ldots,c_5), \mathsf{Hw}_{\mathsf{total}} = w, \mathsf{full}] = \Pr[\mathsf{match}|(c_1,\ldots,c_5), \mathsf{full}]\ ,$$

since $(c_1,\ldots,c_5)$ have all a total Hamming weight of w. We can now consider every Sbox independently. Hence,

$$\Pr[\mathsf{match}|(c_1,\ldots,c_5), \mathsf{full}] = \prod_{i=1}^{5} \left(\Pr[\mathsf{match}|\mathsf{Hw}_{\mathsf{SBox}} = i, \mathsf{full}]\right)^{c_i} \tag{15}$$

and

$$\Pr[\mathsf{match}|\mathsf{Hw}_{\mathsf{SBox}} = i, \mathsf{full}] = \sum_{y\in\{0,1\}^5} \sum_{\substack{v\in\{0,1\}^5:\\ \mathsf{Hw}(v)=i}} \frac{P_{\mathsf{out}}(y)\cdot \mathbb{1}_{\mathsf{DDT}[v][y]}}{\binom{5}{i}}\ .$$

□

We continue now with our example of the Keccak-f[1600] internal permutation. The measured distributions along with some intermediate values will be given in the extended version of the paper.

We require to test $1/p_{\mathsf{match}}$ backward/forward paths combinations in order to have a good chance for a match. Note that in the next section, we will actually put an extra condition on the match in order to be able to generate enough values in the worst case during the outbound phase.

5.6 The outbound phase

Now that we managed to obtain a match with complexity $1/p_{\mathsf{match}}$, we need to estimate how many solutions can be generated from this match. Again, one is tempted to consider an average case on the Keccak Sbox: the average number of Sbox values verifying a non-zero random input/output difference such that the transition is possible is equal to $2^{1.65}$. The overall number of solutions would then be $2^{1.65\cdot s}$. However, as for p_{match}, this number highly depends on the Hamming weight of the input of the Sboxes and this can be easily observed from the DDT of the χ layer (for example with an input Hamming weight of one the average number of solutions is 2^3, while for an input Hamming weight of four the average number of solutions is 2^1).

In order to obtain the expected number of values N_{match} we can get from a match, we proceed like in the previous section and divide according to the input Hamming weight.

Theorem 2. *Let N be a random variable denoting the number of values we can generate. Let also* full *be the event denoting that all the Sboxes are active for the inbound phase. Given a Hamming weight of w at the input of the Sboxes, we can get $N_w := \mathbb{E}[N|\mathsf{match}, \mathsf{Hw}_{\mathsf{total}} = w, \mathsf{full}]$ values from a match, with*

$$N_w = \frac{1}{p_{\mathsf{match}}(w)} \sum_{(c_1,\ldots,c_5)\in C_w} \prod_{i=1}^{5} Z^{c_i} \cdot \frac{b_c(c_1,\ldots,c_5)}{b_{\mathsf{bucket}}(w,s)}\ , \tag{16}$$

with

$$Z \coloneqq \frac{1}{\binom{5}{i}^2} \left(\sum_{\substack{v\in\{0,1\}^5:\\ \mathsf{Hw}(v)=i}} \mathsf{DDT}[v] \right) \sum_{y\in\{0,1\}^5} \sum_{\substack{v\in\{0,1\}^5:\\ \mathsf{Hw}(v)=i}} P_{\mathsf{out}}(y) \cdot \mathbb{1}_{\mathsf{DDT}[v][y]} \ ,$$

where $\mathsf{DDT}[v]$ *is the value of the non-zero entries in line* v *of the DDT,* $P_{\mathsf{out}}(y)$ *is the measured probability distribution of having* y *at the output of an Sbox when we enforce all Sboxes to be active,* $p_{\mathsf{match}}(w)$ *is given by (13),* $b_c(\dots)$ *is given by (11),* $b_{\mathsf{bucket}}(w, s)$ *is given by Lemma 1 and* $\mathbb{1}_{\mathsf{DDT}[v][y]}$ *is set to one if the entry* $[v][y]$ *is non-zero in the DDT of the* χ *layer and zero otherwise.*

Proof. We have

$$\begin{aligned} N_w &= \sum_{(c_1,\dots,c_5)\in C_w} \mathbb{E}[N|\mathsf{match}, (c_1, \dots, c_5), \mathsf{Hw}_{\mathsf{total}} = w, \mathsf{full}] \cdot \Pr[(c_1, \dots, c_5)|\mathsf{match}, \mathsf{Hw}_{\mathsf{total}} = w, \mathsf{full}] \\ &= \sum_{(c_1,\dots,c_5)\in C_w} N_{\mathsf{match}}(c_1, \dots, c_5) \cdot \frac{\Pr[\mathsf{match}|(c_1, \dots, c_5), \mathsf{full}] \cdot \Pr[(c_1, \dots, c_5)|\mathsf{Hw}_{\mathsf{total}} = w, \mathsf{full}]}{p_{\mathsf{match}}(w)} \ , \end{aligned}$$

where $N_{\mathsf{match}}(c_1, \dots, c_5) \coloneqq \mathbb{E}[N|\mathsf{match}, (c_1, \dots, c_5), \mathsf{full}]$. Note that the remaining terms can be computed from (14) and (15). Like before, we can now consider each Sbox independently. Thus

$$N_{\mathsf{match}}(c_1, \dots, c_5) = \prod_{i=1}^{5} \left(\mathbb{E}[N_{\mathsf{SBox}}|\mathsf{match}, \mathsf{Hw}_{\mathsf{SBox}} = i, \mathsf{full}]\right)^{c_i} \ ,$$

where N_{SBox} is a random variable denoting the number of values we can obtain for a single Sbox. Note that no output distribution needs to be considered, since for a fixed input the non-zero values of the DDT are always the same. We call this non-zero value $\mathsf{DDT}[v]$. Then,

$$\mathbb{E}[N_{\mathsf{SBox}}|\mathsf{match}, \mathsf{Hw}_{\mathsf{SBox}} = i, \mathsf{full}] = \frac{1}{\binom{5}{i}} \sum_{\substack{v\in\{0,1\}^5:\\ \mathsf{Hw}(v)=i}} DDT[v] \ .$$

□

One would be tempted to take the expected value of all the N_w and compute N_{match} as

$$\sum_{w} \mathbb{E}[N|\mathsf{match}, \mathsf{Hw}_{\mathsf{total}} = w, \mathsf{full}] \cdot \Pr[\mathsf{Hw}_{\mathsf{total}} = w|\mathsf{match}, \mathsf{full}] \ .$$

This expectancy would be fine if we were expecting a high number of matches. This is however not necessarily our case. Hence, we need to ensure that the number of values we can generate from the inbound is sufficient. To do this, first note that N_w decreases exponentially while w increases. Similarly, $p_{\mathsf{match}}(w)$ increases exponentially while w increases. Thus, we are more likely to obtain a match at a high Hamming weight which will lead to an insufficient N_{match}.

To solve this issue, we proceed as follows. First, we compute N_w for every w. We look then for the maximum Hamming weight $w_{\max}$ we can afford, i.e., such that $N_{w_{\max}} \geq 1/(p_B \cdot p_F)$. This way, we are ensured to obtain enough solutions from the match. However, we need to update our definition of a match: a match occurs only when the Hamming weight of the input is lower than $w_{\max}$. Hence, instead of summing over all possible values of w, we sum only up to $w_{\max}$ and (12) becomes

$$\begin{aligned} p_{\mathsf{match}} = \sum_{w=s}^{w_{\max}} \Pr[Hw_{\mathsf{total}} = w|\mathsf{full}] \cdot \sum_{(c_1,\ldots,c_5)\in C_w} \frac{b_c(c_1,\ldots,c_5)}{b_{\mathsf{bucket}}(w,s)} \times \\ \times \left(\prod_{i=1}^{5} \left(\sum_{y\in\{0,1\}^5} \sum_{\substack{v\in\{0,1\}^5: \\ \mathsf{Hw}(v)=i}} \frac{P_{\mathsf{out}}(y)\cdot \mathbb{1}_{\mathsf{DDT}[v][y]}}{\binom{5}{i}} \right)^{c_i} \right) . \end{aligned} \tag{17}$$

The number of values we can then obtain from the inbound is $N_{\mathsf{match}} \geq N_{w_{\max}}$.

We can now apply this model to the KECCAK-f[1600] internal permutation. Some useful intermediate results and relevant $N_{w_{\max}}$ (with their associated p_{match}) will be given in the extended version of the paper.

5.7 Finalizing the Attack and Improvements

In Section 5.4, we showed how to choose the backward paths given the probability of having a match in the inbound phase (p_{match}) and the number of solution we can generate from this match (N_{match}). In Sections 5.5 and 5.6, we showed how to compute p_{match} and N_{match}. However, in these computations, we needed the probability distribution of the Hamming weight of the input of the Sbox, $\Pr[\mathsf{Hw}_{\mathsf{total}} = w|\mathsf{full}]$. This probability depends greatly on the paths we select in Section 5.4.

To solve this circular dependency, we performed several iterations of the following algorithm until we found some parameters that verify all equations. First, we estimated roughly $\Pr[\mathsf{Hw}_{\mathsf{total}} = w|\mathsf{full}]$ by taking some random backward paths with limited complexity. Using the worst case cost of these paths, we were able to select $w_{\max}$ such that the number of values generated from the inbound is sufficient. Then, we computed p_{match} and N_{match}. With this first guess, we searched for an X and a k such that the we can find a match with a good probability and such that we can generate enough values from the inbound. Then, we computed $\Pr[\mathsf{Hw}_{\mathsf{total}} = w|\mathsf{full}]$ using these new paths generated by X, k and p_B and started our algorithm again with this new distribution. After some iterations, we found a set of filtered backwards paths that provided a sufficient p_{match} and N_{match}.

As discussed in Section 5.4, we noticed the following interesting behavior. By increasing p_B, the expectation of $\Pr[\mathsf{Hw}_{\mathsf{total}} = w|\mathsf{full}]$ is higher. This leads then to a smaller N_{match} and a greater p_{match}. Furthermore, less values need to be generated from the inbound phase since the worst case cost of the backward paths is lower. By taking advantage of this behavior, we were able to reduce significantly the complexity of our attack.

When $(X,k) = (8,8)$, we have $\epsilon = 0.736$ and $\Gamma_B^{\mathsf{mid}} = \binom{5}{2}^X \cdot \binom{s}{X} \cdot \epsilon = 2^{77.26}$. If we filter all paths that have a DP smaller than 2^{-450}, i.e., we set $p_B = 2^{-450}$, we get for $(X,k) = (8,8)$ at least $\epsilon \cdot \binom{5}{2}^X \cdot \binom{s}{X} \cdot \binom{2X}{k} \cdot 2^k \cdot G_B(n) \cdot \tau_B^{\mathsf{full}} = 2^{493.88-15.9} = 2^{477.98}$ distinct differences using (10) for the inbound (for these parameters, the difference Hamming weight at the input of the χ layer in the third round is $n = 227.9$). With these parameters, since we remove the paths with a DP lower than p_B, we keep $\tau_B^{\mathsf{DP}} \approx 1-10^{-10}$ of the paths, following (9), i.e., we have almost no filtering on the DP. Hence, we filter the backward paths with a ratio $\tau_B = \tau_B^{\mathsf{full}} \cdot \tau_B^{\mathsf{DP}} \approx 2^{-15.9} \cdot (1 - 10^{-10}) = 2^{-15.9}$. We have also $p_B = 2^{-450}$ and $p_F = 2^{-36}$. Therefore, we need $N_{\mathsf{match}} \geq 2^{486}$. Computations show that we have to set $w_{\max} = 1000$. This leads to $p_{\mathsf{match}} = 2^{-491.47}$. This implies that the minimum total number of backward paths we need to generate is $n_B^{\min} = 1/(p_{\mathsf{match}} \cdot \Gamma_F^{\mathsf{in}} \cdot \tau_B) = 1/(p_{\mathsf{match}} \cdot \Gamma_F^{\mathsf{in}} \cdot \tau_B^{\mathsf{full}}) = 2^{484.07}$. All these paths apply on $nr_B = 2$ KECCAK-f[1600] rounds, all with DP higher or equal to $p_B^{\min} = 2^{36-486} = 2^{-450}$.

To summarize, we have that the number of backward output differences is $\Gamma_B^{\mathsf{out}} = n_B^{\min} \cdot \tau_B = 2^{484.07-15.9} = 2^{468.17}$ and that the number of forward input differences is $\Gamma_F^{\mathsf{in}} = 2^{23.3}$. Hence, there is a total of $2^{491.47}$ couples of $(\Delta_B^{\mathsf{out}}, \Delta_F^{\mathsf{in}})$ for the inbound phase, which is enough since it is equal to $1/p_{\mathsf{match}}$. Once a match is found, the worst case complexity of the connected path is $1/(p_B \cdot p_F) \leq 2^{450+48} = 2^{486}$ which is lower or equal to N_{match}. Hence, we can generate enough values from the inbound phase to find with a good probability values verifying the differential path.

The *overall complexity for the rebound attack* given by (1) is $C = 2^{491.47}$.

This model was verified on the KECCAK-$f[100]$ internal permutation. By applying this attack on it, we found a 4-round result together with solution pairs. This gives a 6-round distinguisher with complexity $2^{28.76}$ which is higher than the simple distinguishers for 6 rounds. However, our goal for KECCAK-$f[100]$ was to verify our model in practice, so that we can be confident for applying it to the KECCAK-$f[1600]$ version. Moreover, finding such a solution is hard since all s Sboxes are active in the middle of the path.

5.8 The distinguisher

We will use exactly the same type of limited-birthday distinguishers as in Section 4. Our rebound attack finds pairs of internal state values such that the input and output difference masks are fully predetermined and we already showed that this should require 2^b operations in the generic case. Therefore, we obtain a $(nr_B + nr_F + 1)$-round distinguisher for the b-bit KECCAK internal permutation considered if the total cost of the rebound attack to find one solution is lower than 2^b.

In the case of KECCAK-$f[1600]$, we have $nr_B = nr_F = 2$ but note that the backward paths utilized do not have the same input difference Δ_B^{in} and the forward paths do not have the same output difference Δ_F^{out}. We can attack three more rounds by adding two extra rounds to the right of the forward paths exactly as we did for the distinguishers in Section 4.2 and one more round to the left of the backwards paths (see Fig. 3 and 4).

Relaxing the Forward Paths. We analyze now the impact of this two additional paths on Γ_F^{out}, the set of reachable output differences. At the entrance of the third round, every Sbox has one single active bit. Hence, according to the DDT, there are only 4 different possibilities at the output of the Sboxes. Since we have 6 active Sboxes in the third round, the number of possible differences at the output of the third round is multiplied by $4^6 = 2^{12}$. Thus, the number of differences at the output of the third round is $\Gamma_F^{\mathsf{mid}} \cdot 2^{12} = 2^6 \cdot 2^{12} = 2^{18}$.

We need now to look at the fourth round to obtain Γ_F^{out} and compute the generic complexity of the distinguisher. In the third round, every active Sbox can produce at most 3 active bits at its output, since each active Sbox has only one single active bit at its input. Hence, the maximum Hamming weight at the output is $3 \cdot 6 = 18$. Each of these active bits can be expanded to at most 11 bits through θ and hence, we have at most $11 \cdot 18 = 198$ active bits at the input of the Sboxes of the fourth round. In the worst case, each of these bits will be in a different Sbox and will produce four possible differences. Hence, we have $\Gamma_F^{\mathsf{out}} \leq \Gamma_F^{\mathsf{mid}} \cdot 2^{12} \cdot 4^{198} = 2^{18} \cdot 2^{396} = 2^{414}$.

Relaxing the Backward Paths. Each Sbox with one single active bit at its output can have 9 possible input differences and the maximum possible of input differences that can occur for a given input difference is 12 (see χ^{-1} DDT). Since we have $2X$ active Sboxes, the number of possible input differences is increased by a factor of at most 9^{2X}. Therefore, $\Gamma_B^{\mathsf{in}} \leq \Gamma_B^{\mathsf{mid}} \cdot 9^{2X}/\epsilon$ and we reduced the complexity by a factor 2^{4X}.

We have $\Gamma_B^{\mathsf{in}} \leq \Gamma_B^{\mathsf{mid}}(8) \cdot 9^{2 \cdot 8}/\epsilon = 2^{77.7+50.7} = 2^{128.4}$ and $\Gamma_F^{\mathsf{out}} \leq 2^{414}$. The generic complexity of the distinguisher is, hence, greater than $2^{1057.6}$. This is much greater than the complexity of the rebound attack $C = 2^{491.47}$.

6 Results and Conclusion

Table 2. Best differential distinguishers complexities for each version of Keccak internal permutations, for 1 up to 8 rounds. Note that due to its technical complexity when applied on Keccak, the rebound attack has only been applied to Keccak-$f[100]$ and Keccak-$f[1600]$.

b	best differential distinguishers complexity							
	1 rd	2 rds	3 rds	4 rds	5 rds	6 rds	7 rds	8 rds
100	1	1	1	2^2	2^8	2^{19}	-	-
200	1	1	1	2^2	2^8	2^{20}	2^{46}	-
400	1	1	1	2^2	2^8	2^{24}	2^{84}	-
800	1	1	1	2^2	2^8	2^{32}	2^{109}	-
1600	1	1	1	2^2	2^8	2^{32}	2^{142}	$2^{491.47}$

In this article, we analysed the internal permutations used in the Keccak family of hash functions in regards to differential cryptanalysis. We first proposed a generic method that looks for the best differential paths using CP-kernel considerations and better χ mapping. This new method provides some of the best known differential paths for the Keccak internal permutations and we derived distinguishers with rather low complexity exploiting these trails. In particular we were able to obtain a practical distinguisher for 6 rounds of the Keccak-$f[1600]$ permutation. Then, aiming for attacks reaching more rounds, we adapted the rebound attack to the Keccak case. This adaptation is far from trivial and contains many technical details. Our model was verified by applying the attack on the reduced version Keccak-$f[100]$. The main final result is a 8-round distinguisher for the Keccak-$f[1600]$ internal permutation with a complexity of $2^{491.47}$. All our distinguisher results are summarized in Table 2. Note that our attack does not endanger the security of the full Keccak. We believe that this work will also help to apply the rebound attack on a much larger set of primitives.

This work might be extended in many ways, in particular by further refining the differential path search or by improving the inbound phase of the rebound attack such that the overall cost is reduced. Moreover, another research direction would be to analyse how the differential paths derived in this article can lead to collision attacks against reduced versions of the Keccak hash functions. Also, the bottleneck of our attack is now p_{match}. Using the techniques presented in [22] could help reducing the complexity of it.

Acknowledgements. The authors would like to thank the anonymous referees for their helpful comments. We are extremely grateful to Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van Assche for their remarks on the first drafts of this paper. Finally, we are very grateful to Praveen Gauravaram, Tao Huang, Phuong Ha Nguyen, Wun-She Yap, Przemyslaw Sokolowski and Wenling Wu for useful discussions.

References

1. Masayuki Abe, editor. *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *LNCS*. Springer, 2010.
2. Jean-Philippe Aumasson and Willi Meier. Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. Presented at the rump session of CHES 2009, 2009.

3. Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *CCS*, pages 62–73. ACM, 1993.
4. Daniel J. Bernstein. Second preimages for 6 (7? (8??)) rounds of Keccak?, November 2010. Available at `http://ehash.iaik.tugraz.at/uploads/6/65/NIST-mailing-list_Bernstein-Daemen.txt`.
5. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Sponge functions. ECRYPT Hash Workshop 2007, May 2007.
6. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Keccak specifications. Submission to NIST (Round 1), 2008. Available at `http://keccak.noekeon.org/Keccak-specifications.pdf`.
7. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Keccak specifications. Submission to NIST (Round 2), 2009. Available at `http://keccak.noekeon.org/Keccak-specifications-2.pdf`.
8. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On alignment in Keccak. ECRYPT II Hash Workshop, 2011.
9. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. The KECCAK Reference. Submission to NIST (Round 3), 2011. Available at `http://keccak.noekeon.org/Keccak-reference-3.0.pdf`.
10. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. The KECCAK SHA-3 Submission. Submission to NIST (Round 3), 2011. Available at `http://keccak.noekeon.org/Keccak-submission-3.pdf`.
11. Christina Boura, Anne Canteaut, and Christophe De Cannière. Higher-Order Differential Properties of Keccak and *Luffa*. In Antoine Joux, editor, *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 252–269. Springer, 2011.
12. Henri Gilbert and Thomas Peyrin. Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations. In Hong and Iwata [13], pages 365–383.
13. Seokhie Hong and Tetsu Iwata, editors. *Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers*, volume 6147 of *LNCS*. Springer, 2010.
14. Dmitry Khovratovich, María Naya-Plasencia, Andrea Röck, and Martin Schläffer. Cryptanalysis of *Luffa* v2 Components. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 388–409. Springer, 2010.
15. Dmitry Khovratovich, Ivica Nikolic, and Christian Rechberger. Rotational Rebound Attacks on Reduced Skein. In Abe [1], pages 1–19.
16. Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, and Martin Schläffer. Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *LNCS*, pages 126–143. Springer, 2009.
17. Florian Mendel, Thomas Peyrin, Christian Rechberger, and Martin Schläffer. Improved Cryptanalysis of the Reduced Grøstl Compression Function, ECHO Permutation and AES Block Cipher. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *LNCS*, pages 16–35. Springer, 2009.
18. Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In Orr Dunkelman, editor, *FSE*, volume 5665 of *LNCS*, pages 260–276. Springer, 2009.
19. Pawel Morawiecki and Marian Srebrny. A SAT-based preimage analysis of reduced Keccak hash functions. Presented at Second SHA-3 Candidate Conference, Santa Barbara 2010, 2010.
20. National Institute of Standards and Technology. FIPS 180-1: Secure Hash Standard. `http://csrc.nist.gov`, April 1995.
21. National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. *Federal Register*, 27(212):62212–62220, November 2007. Available: `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`(2008/10/17).
22. María Naya-Plasencia. How to Improve Rebound Attacks. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 188–205. Springer, 2011.
23. María Naya-Plasencia, Andrea Röck, and Willi Meier. Practical analysis of reduced-round keccak. In Daniel J. Bernstein and Sanjit Chatterjee, editors, *INDOCRYPT*, volume 7107 of *Lecture Notes in Computer Science*, pages 236–254. Springer, 2011.
24. Vincent Rijmen, Deniz Toz, and Kerem Varici. Rebound Attack on Reduced-Round Versions of JH. In Hong and Iwata [13], pages 286–303.
25. Ronald L. Rivest. The MD5 message-digest algorithm. Request for Comments (RFC) 1320, Internet Activities Board, Internet Privacy Task Force, April 1992.
26. Yu Sasaki, Yang Li, Lei Wang, Kazuo Sakiyama, and Kazuo Ohta. Non-full-active Super-Sbox Analysis: Applications to ECHO and Grøstl. In Abe [1], pages 38–55.
27. Keccak team. Keccak Crunchy Crypto Collision and Pre-image Contest, 2011. See `http://keccak.noekeon.org/crunchy_contest.html`.
28. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.
29. Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 19–35. Springer, 2005.

Differential propagation analysis of Keccak

Joan Daemen and Gilles Van Assche

STMicroelectronics

Abstract. In this paper we introduce new concepts that help read and understand low-weight differential trails in Keccak. We then propose efficient techniques to exhaustively generate all 3-round trails in its largest permutation below a given weight. This allows us to prove that any 6-round differential trail in Keccak-f[1600] has weight at least 74. In the worst-case diffusion scenario where the mixing layer acts as the identity, we refine the lower bound to 82 by systematically constructing trails using a specific representation of states.

Keywords: cryptographic hash function, Keccak, differential cryptanalysis, computer-aided proof

1 Introduction

The goal of cryptanalysis is to assess the security of cryptographic primitives. Finding attacks or properties not present in ideal instances typically contributes to the cryptanalysis of a given primitive. Building upon previous results, attacks can be improved over time, possibly up to a point where the security of the primitive is severely questioned.

In contrast, cryptanalysis can also benefit from positive results that exclude classes of attacks, thereby allowing research to focus on potentially weaker aspects of the primitive. Interestingly, weaknesses are sometimes revealed by challenging the assumptions underlying positive results. Nevertheless, both attacks and positive results can be improved over time and together contribute to the understanding and estimation of the security of a primitive by narrowing the gap between what is possible to attack and what is not.

Differential cryptanalysis (DC) is a discipline that attempts to find and exploit predictable difference propagation patterns to break iterative cryptographic primitives [6]. For ciphers, this typically means key retrieval, while for hash functions, this is the generation of collisions or of second preimages. The basic version makes use of differential trails (also called characteristics or differential paths) that consist of a sequence of differences through the rounds of the primitive. Given such a trail, one can estimate its differential probability (DP), namely, the fraction of all possible input pairs with the initial trail difference that also exhibit all intermediate and final difference when going through the rounds.

A more natural way to characterize the power of trails in unkeyed primitives is by their weight w. In general the weight of a trail is is the sum of the weight of its round differentials, where the latter is the negative of its binary logarithm. For many round functions, including that of Keccak-f and Rijndael, the weight equals the number of binary equations that a pair must satisfy to follow the specified differences. Assuming that these conditions are independent, the weight of the trail relates to its DP as $\mathrm{DP} = 2^{-w}$ and exploiting such a trail becomes harder as the weight increases. For a primitive with, say, b input and output bits, the number of pairs that satisfy these conditions is then 2^{b-w}. The assumption of independence does not always apply. For instance, a trail with $w > b$ implies redundant or contradictory conditions on pairs, for which satisfying pairs may or may not exist. Another example where this independence assumption breaks down are the plateau trails that occur in Rijndael [9]. These trails, with weight starting from $w = 30$ for 2 rounds, have a DP equal to 2^{z-w} with $z > 0$ for a fraction 2^{-z} of the keys and zero for the remaining part. In general, they occur in primitives with strong alignment [4] and a mixing layer based on maximum-distance separable (MDS) codes.

In the scope of DC, positive results can be established by finding a lower bound on the weight of any trail over a specified number of rounds. For instance, the structure of Rijndael and the properties of its diffusion operations allow to analytically derive such lower bounds [8]. Such results can be transposed to the permutations underlying the hash function Grøstl [12]. Other examples include a lower bound on the number of active S-boxes in JH [17] or computer-aided proofs on the weight of trails in Noekeon [7] and on the minimum number of active AND gates in MD6 [16,13].

Keccak is a sponge function submitted to the SHA-3 contest [15,5,2]. Recently, new results were published on the differential resistance of this function and among those heuristic techniques were proposed to build low-weight differential trails [11,14]. These gave the currently best trails for 3, 4 and 5 rounds of the underlying permutation Keccak-f[1600]. In particular, Duc et al. found a trail of weight 32 for 3 rounds, and this motivated us to systematically investigate whether trails of lower weight exist. Also, there are some similarities between Keccak and MD6, but unlike MD6, the permutation used in the proposed SHA-3 candidate Keccak has no significant lower bounds on the weight of trails. So the philosophy behind [16,13] was another source of inspiration and motivation for our research.

Lower bounds on symmetric trails were already proven in [5]. They provide lower bounds with weight above the permutation width on Keccak-f[25] to Keccak-f[200] but only partial bounds in the case of Keccak-f[1600]. Thanks to the Matryoshka structure [5], a lower bound W on trails in Keccak-f[$25w$] implies a lower bound $W' = W\frac{w'}{w}$ on w-symmetric trail in Keccak-f[$25w'$] for $w' > w$. These are summarized in Table 1.

w	**Lower bound for Keccak-f[$25w$]**	**Lower bound for Keccak-f[1600]**	
1	30 per 5 rounds	1920 per 5 rounds	tight
2	54 per 6 rounds	1728 per 6 rounds	tight
4	146 per 16 rounds	2336 per 16 rounds	non-tight
8	206 per 18 rounds	1648 per 18 rounds	non-tight

Table 1. Lower bounds above the permutation width on 1- to 8-symmetric trails [5].

In this paper, we report on techniques to efficiently generate all the trails in Keccak-f[1600] up to a given weight. We implemented these techniques in a computer program, which allowed us at this point to completely scan the space of 3-round differential trails up to weight 36. This confirmed that the trail found by Duc et al. has minimum weight and allowed us to demonstrate that there are no 6-round trails with weight below 74. These results are summarized in Table 2. The source code of the program will be made available in an updated version of the KeccakTools package [3].

As a by-product of this trail search, this paper proposes new techniques to relate the properties of the θ mapping in Keccak to the weight of differential trails. In the worst-case diffusion scenario where θ acts as the identity, we build upon the results of [5] and [14] to systematically construct so-called in-kernel trails using an efficient representation of states.

Further discussions on how to exploit differential trails in Keccak can be found in [4]. Also, the attacks in [10] combine algebraic techniques with a differential trail.

The paper is organized as follows. In Section 2, we recall the structure of Keccak and mappings inside its round function. Section 3 focuses on how to represent and extend the differential trails of Keccak. Section 4 sets up the overall strategy and Section 5 introduces a basic trail

Rounds	Lower bound	Best known
3	32 (this work)	32 [11]
4	-	134 (Appendix B)
5	-	510 [14]
6	74 (this work)	1360 [5]
24	296 (this work)	-

Table 2. Weight of differential trails in Keccak-f[1600].

generation technique. The advanced techniques are covered in Sections 6 and 7, which address two complementary cases. Finally, Section 8 extends the results from 3 to 6 rounds.

2 Keccak

Keccak combines the sponge construction with a set of seven permutations denoted Keccak-$f[b]$, with b ranging from 25 to 1600 bits [1,5]. In this paper, we concentrate on the permutation used in the SHA-3 submission, namely, Keccak-f[1600].

The state of Keccak-f[1600] is organized as a set of $5 \times 5 \times 64$ bits with (x, y, z) coordinates. The coordinates are always considered modulo 5 for x and y and modulo 64 for z. A *row* is a set of 5 bits with given (y, z) coordinates, a *column* is a set of 5 bits with given (x, z) coordinates and a *slice* is a set of 25 bits with given z coordinate.

The round function of Keccak-f[1600] consists of the following steps, which are only briefly summarized here. For more details, we refer to the specifications [5].

- θ is a linear mixing layer, which adds a pattern that depends solely on the parity of the columns of the state. Its properties with respect to differential propagation will be detailed and exploited in Section 6.
- ρ and π displace bits without altering their value. Jointly, their effect is denoted by $(x, y, z) \xrightarrow{\pi\circ\rho} (x', y', z')$, with (x, y, z) a bit position before ρ and π and (x', y', z') its coordinates afterward.
- χ is a degree-2 non-linear mapping that processes each row independently. It can be seen as the application of a translation-invariant 5-bit S-box. The differential propagation properties will be detailed below.
- ι adds a round constant. As it has no effect on difference propagation, we will ignore it in the sequel.

3 Representing and extending trails

In general, for a function f with domain $\mathbb{Z}_2^b$, we define the *weight* of a differential (u', v') as

$$\mathrm{w}(u' \xrightarrow{f} v') = b - \log_2 \left|\{u \ : \ f(u) \oplus f(u \oplus u') = v'\}\right| .$$

If the argument of the logarithm is non-zero (i.e., the DP is non-zero), we say that u' and v' are *compatible*. Otherwise, the weight is undefined.

The weight of a trail is the sum of the weight of the differentials that compose this trail. In Keccak-f, we specify differential trails with the differences before each round function. For clarity, we adopt a redundant description by also specifying the differences before and after the linear steps $\lambda = \pi \circ \rho \circ \theta$. An n-round trail is of the following form, where each b_i must be equal to $\lambda(a_i)$,

$$Q = a_0 \xrightarrow{\pi\circ\rho\circ\theta} b_0 \xrightarrow{\chi} a_1 \xrightarrow{\pi\circ\rho\circ\theta} b_1 \xrightarrow{\chi} a_2 \xrightarrow{\pi\circ\rho\circ\theta} \quad \dots \quad \xrightarrow{\chi} a_n, \tag{1}$$

and has weight $\mathrm{w}(Q) = \sum_i \mathrm{w}(a_i \xrightarrow{\chi\circ\pi\circ\rho\circ\theta} a_{i+1})$. Since $b_i = \lambda(a_i)$, this expression simplifies to $\mathrm{w}(Q) = \sum_i \mathrm{w}(b_i \xrightarrow{\chi} a_{i+1})$.

3.1 Extending forward and trail prefixes

Given a trail as in (1), it is possible to characterize all states that are compatible with $b_n = \lambda(a_n)$ through χ and thus to find all $n+1$-round trails Q' that have Q as its leading part. This process is called *forward extension.*

The χ mapping has algebraic degree 2 and, for a given input difference b_n, the space of compatible output differences forms a linear affine variety $\mathcal{A}(b_n)$ with $|\mathcal{A}(b_n)|$ elements [5]. For a compatible a_{n+1}, the weight $\mathrm{w}(b_n \overset{\chi}{\to} a_{n+1})$ depends only on b_n and is equal to $\mathrm{w}(b_n) \triangleq \log_2 |\mathcal{A}(b_n)|$, with the symbol $\triangleq$ denoting a definition. As χ operates on each row independently, the weight $\mathrm{w}(b)$ can also be computed on each row independently and summed. To construct $\mathcal{A}(b)$, the bases resulting from each active row are gathered. Table 3 displays offsets and bases for the affine spaces of all single-row differences.

	forward propagation							
Difference	offset	base elements				$\mathrm{w}(\cdot)$	$\mathrm{w}^{\mathrm{rev}}(\cdot)$	$\|\| \cdot \|\|$
00000	00000					0	0	0
00001	00001	00010	00100			2	2	1
00011	00001	00010	00100	01000		3	2	2
00101	00001	00010	01100	10000		3	2	2
10101	00001	00010	01100	10001		3	3	3
00111	00001	00010	00100	01000	10000	4	2	3
01111	00001	00011	00100	01000	10000	4	3	4
11111	00001	00011	00110	01100	11000	4	3	5

Table 3. Space of possible output differences, weight, minimum reverse weight and Hamming weight of all row differences, up to cyclic shifts.

As a consequence, the weight of a n-round trail Q is $\mathrm{w}(Q) = \sum_{i=0}^{n-1} \mathrm{w}(b_i)$ and depends only on the n-tuple $(b_0, \ldots, b_{n-1})$. We call the latter a *trail prefix.* All n-round trails sharing this trail prefix and with a_n compatible with b_{n-1} through χ have the same weight.

3.2 Extending backward and trail cores

Similarly, given a trail as in (1), it is possible to construct all states that are compatible with a_0 through χ^{-1} and thus to find all $n+1$-round trails Q' that have Q as its trailing part. This process is called *backward extension.* In contrast to χ, its inverse has algebraic degree 3 and the space of compatible differences is not an affine variety in general. Yet, compatible values can be identified per active row and combined.

For a difference a after χ, we define the *minimum reverse weight* $\mathrm{w}^{\mathrm{rev}}(a)$ as the minimum weight over all compatible b before χ. Namely,

$$\mathrm{w}^{\mathrm{rev}}(a) \triangleq \min_{b \,:\, a \in \mathcal{A}(b)} \mathrm{w}(b).$$

Like for the restriction weight, the minimum reverse weight $\mathrm{w}^{\mathrm{rev}}(a)$ can be computed on each row independently and summed. Values are also shown in Table 3.

Given a $n-1$-round trail prefix $Q = (b_1, \ldots, b_{n-1})$, it is easy to construct a difference b_0 such that the trail prefix $Q' = b_0 || Q$ has weight given by $\mathrm{w}(Q') = \mathrm{w}(Q) + \mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_1))$. This is the smallest possible weight a n-round trail can have with Q as its trailing part. It follows that a sequence of $n-1$ state values $\tilde{Q} = (b_1, \ldots, b_{n-1})$ defines a set of n-round trails with a weight at least

$$\tilde{\mathrm{w}}(\tilde{Q}) \triangleq \mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_1)) + \sum_{i=1}^{n-1} \mathrm{w}(b_i).$$

We denote the former by the term *trail core* and the latter by its weight. Note that a n-round trail core is determined by only $n-1$ states, although its weight takes n individual weights into account.

4 Towards a bound for trails in Keccak-f[1600]

To find a lower bound on differential trail weights in Keccak-f[1600], our strategy is the following.

- First, we exhaustively generate all 3-round trails up to a given weight T_3. There exists a trail of weight 32 as found by Duc et al. [11]. So by scanning the space of trails up to weight $T_3 \geq 32$, we are sure to hit at least one trail and the trail with minimum weight yields a tight lower bound on 3-round trails.
- Second, we derive a lower bound, not necessarily tight, on the weight of 6-round trails by using the 3-round trails found. Any 6-round trail of weight $2T_3 + 1$ or less satisfies either $\mathrm{w}(b_0)+\mathrm{w}(b_1)+\mathrm{w}(b_2) \leq T_3$ or $\mathrm{w}(b_3)+\mathrm{w}(b_4)+\mathrm{w}(b_5) \leq T_3$. We thus use forward and backward extension from 3-round trails up to weight $2T_3+1$. If such trails are found, the one with the smallest weight defines the lower bound, which is naturally tight. Otherwise, this establishes a lower bound for the weight of 6-round trails to $2T_3 + 2$. In the latter case no trail with weight $2T_3 + 2$ is known so the bound is not necessarily tight.

The reason for targeting 3-round trails in the first phase is the following. The minimum weight of a 1-round trail is 2, with a single active bit in b_0. For the 24 rounds of Keccak-f[1600], this amounts to a lower bound of $24 \times 2 = 48$. Constructing a state a with only two active bits in the same column leads to 2-round trail core with weight 8. Hence, if we base ourselves only on 2-round trail, we reach a lower bound of $12 \times 8 = 96$. If the 3-round trail of weight 32 found by Duc et al. [11] has minimum weight, this would mean that a 24-round trail has weight at least $8 \times 32 = 256$. Also, 3-round trail cores can be constructed by taking into account conditions across one layer of χ. Generating exhaustively trails of 4 rounds or more up to some weight would probably yield better bounds, but at the same time it is more difficult as several layers of χ must be dealt with. Instead, the two-step approach described above can take advantage of the exhaustive set of trails covered (i.e., all up to weight T_3) to derive a bound based on T_3 instead of on the minimum weight over 3 rounds.

4.1 Generating all 3-round trails up to a given weight

In our approach we generate all 3-round differential trails of the form

$$Q = a_0 \xrightarrow{\pi\circ\rho\circ\theta} b_0 \xrightarrow{\chi} a_1 \xrightarrow{\pi\circ\rho\circ\theta} b_1 \xrightarrow{\chi} a_2 \xrightarrow{\pi\circ\rho\circ\theta} b_2 \xrightarrow{\chi} a_3, \tag{2}$$

up to some weight limit $\mathrm{w}(Q) \leq T_3$. We call this the target space. We do this by searching for all trail cores (b_1, b_2) with weight below T_3. Each such trail core (b_1, b_2) thus represents a set 3-round trails of the form of Eq. (2) with weight not below that of its core. In the scope of this paper, we limited ourselves to $T_3 = 36$.

We covered the set of all 3-round trails up to weight T_3 in three sub-phases:

1. In Section 5, we start with all cores such that $\mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_1)) \leq 7$, $\mathrm{w}(b_1) \leq 7$ or $\mathrm{w}(b_2) \leq 7$.
2. In Section 6, we generate all remaining cores, except where both a_1 and a_2 are in the kernel.
3. In Section 7, we finish by generating all cores where both a_1 and a_2 are in the kernel.

4.2 Too many states to generate and extend, even when exploiting symmetry

A way to generate all trails in the target space is to first generate all states up to a given weight and then do backward and forward extensions to obtain trail cores. If we define $T_1 \triangleq \lfloor \frac{T_3}{3} \rfloor$, then for $\tilde{\mathrm{w}}(b_1, b_2) \leq T_3$ either $\mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_1)) \leq T_1$, $\mathrm{w}(b_1) \leq T_1$ or $\mathrm{w}(b_2) \leq T_1$. To cover the target space, we need to consider these cases:

- $\mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_1)) \leq T_1$, so we have to generate all states a_1 with $\mathrm{w}^{\mathrm{rev}}(a_1) \leq T_1$, compute $b_1 = \lambda(a_1)$ and extend forward the 2-round trail cores (b_1) to get 3-round trail cores.
- $\mathrm{w}(b_1) \leq T_1$, so we have to generate all states b_1 with $\mathrm{w}(b_1) \leq T_1$ and extend forward the 2-round trail cores (b_1).
- $\mathrm{w}(b_2) \leq T_1$, so we have generate all states b_2 with $\mathrm{w}(b_2) \leq T_1$ and extend backward the 2-round trail cores (b_2).

Unfortunately, this brute-force strategy requires a high number of states to cover the whole space for an interesting target weight. E.g., if $T_3 = 36$, then $T_1 = 12$ and there are about $1.42 \times 10^{15} \approx 2^{50}$ states with weight up to 12 in Keccak-f[1600].

We can reduce this number by taking the z symmetry into account. Except for ι, which does not influence difference propagation, all the step mappings of Keccak-f are invariant when translated along z. Hence, for each trail $Q = (b_0, b_1, \ldots, b_n)$ there exists a trail $Q' = (\mathrm{z}(b_0), \mathrm{z}(b_1), \ldots, \mathrm{z}(b_n))$ of same weight, with z the translation operator along the z axis. In the sequel, we will always consider trails up to translations in z. This reduces the search space by approximately a factor $w = 64$—not exactly a factor w because of states that are periodic in z. Yet, the number of states to extend forward and backward is still about 2^{44}.

5 Generating trails with a low number of active rows

In this section, we generate and extend states with weight up to $T_1' = 7$. This does not cover the whole target space with $T_3 = 36$ but the remaining portion of the target space is limited to trails with a more flat weight profile, i.e., they satisfy $\mathrm{w}(b_i) \geq T_1' + 1 = 8$ for all $i \in \{0, 1, 2\}$ and $\mathrm{w}(b_i) + \mathrm{w}(b_{i+1}) \leq T_2' = T_3 - (T_1' + 1) = 28$ for all $i \in \{0, 1\}$.

More specifically, in this phase we look at the number of active rows in order to generate all trail cores such that $\mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_1)) \leq T_1'$, $\mathrm{w}(b_1) \leq T_1'$ or $\mathrm{w}(b_2) \leq T_1'$, for $T_1' = 7$. According to Table 3, each active row contributes for at least 2 to the weight. Hence,

$$\mathrm{w}(b) \geq 2\|b\|_{\mathrm{row}} \quad \text{and} \quad \mathrm{w}^{\mathrm{rev}}(b) \geq 2\|b\|_{\mathrm{row}},$$

and we can cover all the states up to weight 7 by generating all states with up to $\lfloor \frac{T_1'}{2} \rfloor = 3$ active rows.

This approach can be refined by looking at the number of active rows not only for one state but for two consecutive states. With χ, the minimum weight a round differential can have is 2. So, $\mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_1)) \geq 2$ implies that $\mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_2)) + \mathrm{w}(b_2) \leq \mathrm{w}(b_1) + \mathrm{w}(b_2) \leq T_3 - 2 = 34$ and similarly $\mathrm{w}(b_2) \geq 2$ implies that $\mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_1)) + \mathrm{w}(b_1) \leq T_3 - 2 = 34$. Hence,

$$\mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_i)) + \mathrm{w}(b_i) \leq T_3 - 2 = 34 \quad \Rightarrow \quad \|\lambda^{-1}(b_i)\|_{\mathrm{row}} + \|b_i\|_{\mathrm{row}} \leq \left\lfloor \frac{T_3 - 2}{2} \right\rfloor = 17.$$

In practice, what we did was the following.

- Generate $\mathcal{B} = \{b \ : \ (\|b\|_{\mathrm{row}} \leq 3 \text{ or } \|\lambda^{-1}(b)\|_{\mathrm{row}} \leq 3) \text{ and } \|\lambda^{-1}(b)\|_{\mathrm{row}} + \|b\|_{\mathrm{row}} \leq 17\}$. This is done by first generating all states b with up to 3 active rows and filter on $\|\lambda^{-1}(b)\|_{\mathrm{row}}$, and then generate all states a with up to 3 active rows, compute $b = \lambda(a)$ and filter on $\|b\|_{\mathrm{row}}$.

- Do forward extension of all $b_1 \in \mathcal{B}$ and keep the cores $\tilde{Q} = (b_1, b_2)$ with $\tilde{\mathrm{w}}(\tilde{Q}) \le T_3$.
- Do backward extension of all $b_2 \in \mathcal{B}$ and keep the cores $\tilde{Q} = (b_1, b_2)$ with $\tilde{\mathrm{w}}(\tilde{Q}) \le T_3$.

We found a trail core (b_1, b_2) with $\mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_1)) + \mathrm{w}(b_1) + \mathrm{w}(b_2) = 4 + 4 + 24 = 32$ (see also Table 4). It contains the 3-round trail found by Duc et al. [11], of which a trail prefix is displayed in Figure 2 in Appendix A.

There are $\binom{320}{n}(31)^n$ states with n active rows. As this function grows very quickly, it was not reasonable to extend this search beyond 3 active rows.

6 Generating trails using the properties of θ

To investigate the remaining part of the target space, we look at the properties of states a with respect to θ, and specifically the parity of its columns, to limit the weight of two-round trails. An important parameter to classify the states a is their column parity, so as to study states in sets of parities. From the column parity, we derive the θ-gap, defined below. With θ-gap g, the effect of θ is to flip $10g$ bits. There are thus at least $10g$ active bits, each either in a or in $\theta(a)$. So, the higher the θ-gap the higher $\mathrm{w}^{\mathrm{rev}}(a) + \mathrm{w}(\lambda(a))$ is likely to be. We can efficiently compute a lower bound for $\mathrm{w}^{\mathrm{rev}}(a) + \mathrm{w}(\lambda(a))$ over all a with a given parity. For the target weights considered in this paper, this allows us to limit the states to consider to those with a parity belonging to a mere handful of values.

We then use the generated states a are to build trail cores by forward and backward extension. As the θ-gap increases, the number of states a to consider decreases since more states a can immediately be excluded. An important case is when all the columns of a have even parity, i.e., a is in the kernel. In this case, the θ-gap is zero and a high number of states must be generated and extended. For this reason, this section focuses only the case where either a_1 or a_2 is not in the kernel. The complementary case is covered in Section 7.

6.1 Properties of θ

As θ is a linear function, its properties are the same whether applied on a state absolute value or on a difference, so we just write "value". The following definitions are from [5].

The *column parity* (or *parity* for short) $P(a)$ of a value a is defined as the parity of the columns of a, namely $P(a)[x][z] = \sum_y a[x][y][z]$. A column is *even* (resp. *odd*) if its parity is 0 (resp. 1). The parity can also be defined on a slice, namely $P(a_z)[x] = \sum_y a[x][y][z]$. When the parity of a state or of a slice is zero (i.e., all its columns are even), we say it is in the *column-parity kernel* (or *kernel* for short).

The mapping θ consists in adding a pattern to the state, which we call the *θ-effect.* The θ-effect of a value a is $E(a)[x][z] = P(a)[x-1][z] + P(a)[x+1][z-1]$. For a fixed θ-effect $e[x][z]$, θ comes down to adding the y-symmetric pattern $e[x][y][z] \triangleq e[x][z](\forall y)$. So θ depends only on column parities and always affects columns symmetrically in y.

A column of coordinates (x, z) is *affected* iff $E(a)[x][z] = 1$; otherwise, it is *unaffected.* Note that the θ-effect always has an even Hamming weight so the number of affected columns is even.

The *θ-gap* is defined as the Hamming weight of the θ-effect divided by two. Hence, if the θ-gap of a value at the input of θ is g, the number of affected columns is $2g$ and applying θ to it results in $10g$ bits being flipped.

We have introduced the θ-gap via the θ-effect, but it can be defined directly using the parity itself. For this we introduce an alternative, single-dimensional, representation of a parity $p[x][z]$. We map the (x, z) coordinates to a single coordinate t as $t \rightarrow (x, z) = (-2t, t)$ and denote the result by $p[t]$. In this representation a *run* is a sequence of ones delimited by zeroes. As illustrated on Figure 1, each run induces two affected columns. First, if it starts in coordinates

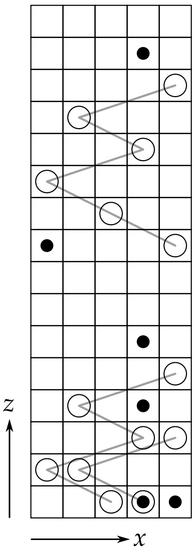


Fig. 1. Example of parity pattern. Each square represents a column. An odd column contains a circle, while an affected column is denoted by a dot. A column can be both odd and affected. The odd columns of a run are connected with a line. The affected columns due to a run are located at the right (resp. top left) of the start (resp. end) column of the run.

(x, z), it implies an affected column in its right neighbor $(x + 1, z)$. And if it ends in (x', z') it implies an affected column in its top-left neighbor $(x' - 1, z' + 1)$. Another example can be found in Figure 2. The following lemma links the number of runs to the θ-gap.

Lemma 1. *The parity p has θ-gap g iff $p[t]$ has g runs.*

6.2 The propagation branch number

The *propagation branch number* of a parity p is the minimum weight of the 2-round trail core (b) among states with this parity. More formally,

$$B(p) \triangleq \min\{\tilde{\mathrm{w}}(b) \ : \ P(\lambda^{-1}(b)) = p\}.$$

Owing to the portion of the target space already covered in Section 5, we can limit the propagation branch number to $T_2' = 28$. The strategy is as follows:

- First, we identify and exclude parity patterns p such that the propagation branch number can be proven to exceed $T_2' = 28$.
- Then, for the remaining parity patterns p we look for all states $b = \lambda(a)$ with $P(a) = p$ and $\tilde{\mathrm{w}}(b) \leq T_2' = 28$.
- Finally, we forward and backward extend the states seen as 2-round trail cores up to weight $T_3 = 36$.

Clearly, the kernel states, i.e., states such that $P(a) = 0$ must be considered. For instance, a state a with just two active bits in the same column will have $\mathrm{w}^{\mathrm{rev}}(a) = 4$. Then, $b = \lambda(a) = \pi(\rho(a))$ since θ has no effect in this case, and b also has two active bits. For Keccak-f[1600], all the rotation constants in ρ are different and these two bits will not be in the same slice, so not in the same row and $\mathrm{w}^{\mathrm{rev}}(a) + \mathrm{w}(b) = 8$. Hence, the propagation branch number of the all-zero parity is at least 8 and thus the all-zero parity pattern must be included.

States that are out of the kernel are likely to have a higher propagation branch number. We now concentrate on how to find a lower bound on the propagation branch number of a given parity pattern.

6.3 Bounding the row branch number

The *row branch number* of a parity p is the minimum number of active rows before and after λ among states with this parity. More formally,

$$B_{\text{rows}}(p) \triangleq \min\{\|\lambda^{-1}(b)\|_{\text{row}} + \|b\|_{\text{row}} \;:\; P(\lambda^{-1}(b)) = p\}.$$

Since an active row has at least propagation weight 2, this means that $B(p) \geq 2B_{\text{rows}}(p)$. We can thus use the row branch number as a way to limit the search to parity patterns for which $\tilde{\text{w}}(b) \leq T_2'$.

For a given parity pattern, we classify the columns as either affected, unaffected odd or unaffected even. We make use of the following properties to find a lower bound on the row branch number.

Lemma 2. *In terms of active rows, θ satisfies the following properties:*

- *An active bit in an affected column before θ will be passive after θ, and vice-versa. So, for each bit $(x, y, z) \xrightarrow{\pi\circ\rho} (x', y', z')$ of an affected column, at least one of row (y, z) in $\lambda^{-1}(b)$ and row (y', z') in b will be active.*
- *An odd unaffected column always contains at least one active bit and this bit stays active after θ. So, for at least one bit $(x, y, z) \xrightarrow{\pi\circ\rho} (x', y', z')$ of an odd unaffected column, both rows (y, z) in $\lambda^{-1}(b)$ and (y', z') in b will be active.*

These properties are translated into Algorithm 1, which returns a lower bound of $B_{\text{rows}}(p)$. The algorithm avoids counting twice an active row by marking (in the sets a and b) the row positions already encountered.

6.4 Looking for candidate parity patterns

To find trails such that any two consecutive rounds have weight up to $T_2' = 28$, we have to consider the parity patterns listed in Lemma 3.

Lemma 3. *A 2-round differential trail $Q = (b_0, b_1, b_2)$ in* Keccak-f[1600] *with* $\text{w}(Q) \leq 28$ *necessarily satisfies one of the following properties on the parity of $a_1 = \lambda^{-1}(b_1)$:*

- *a_1 is in the kernel, i.e., $P(a_1) = 0$;*
- *the θ-gap of a_1 is 1 with a single run of length 1 or 2; or*
- *the θ-gap of a_1 is 2 or 3 with runs of length 1 each, all starting in the same slice.*

If parities are considered up to translation along z, we can restrict ourselves to parity patterns with runs starting in slice $z = 0$.

To prove this result, we conducted a recursive search as follows. Each parity is represented as a set of runs. First, all parity patterns p with a single run (so θ-gap 1) are investigated. All p with $B_{\text{rows}}(p) \leq \frac{T_2'}{2} = 14$ are stored into a set S. Then, we recursively add runs not overlapping the already added ones (so as to cover θ-gaps higher than 1), and all found p with $B_{\text{rows}}(p) \leq \frac{T_2'}{2} = 14$ are stored into a set S.

To limit the search, we use the following monotonicity property on the number of active rows. Using Lemma 2, changing an unaffected even column into either an unaffected odd or an affected column cannot decrease the number of active rows.

Algorithm 1 Computing a lower bound of $B_{\text{rows}}(p)$

```
Let $a$ and $b$ be sets of row positions, which are initially empty
$B \leftarrow 0$
for each affected column $(x, z)$ do
  for $y \in \mathbb{Z}_5$ do
    Let $(x, y, z) \xrightarrow{\pi \circ \rho} (x', y', z')$
    if $(y, z) \notin a$ and $(y', z') \notin b$ then
      $B \leftarrow B + 1$
      $a \leftarrow a \cup \{(y, z)\}$ and $b \leftarrow b \cup \{(y', z')\}$
    end if
  end for
end for
for each unaffected odd column $(x, z)$ do
  Let $(x, i, z) \xrightarrow{\pi \circ \rho} (x'_i, y'_i, z'_i)$ for $i \in \mathbb{Z}_5$
  if $\{(i, z), i \in \mathbb{Z}_5\} \cap a = \emptyset$ then
    $B \leftarrow B + 1$
    $a \leftarrow a \cup \{(i, z), i \in \mathbb{Z}_5\}$
  end if
  if $\{(y'_i, z'_i), i \in \mathbb{Z}_5\} \cap b = \emptyset$ then
    $B \leftarrow B + 1$
    $b \leftarrow b \cup \{(y'_i, z'_i), i \in \mathbb{Z}_5\}$
  end if
end for
return $B$
```

In the recursive search described above, adding a run to a parity pattern p can turn an unaffected odd column into an affected column. Hence, we cannot use the monotonicity property directly on the runs. However, adding a run never turns an affected column back into an unaffected one. So, before recursively adding a run to p, we apply a modified version of Algorithm 1 that does not take unaffected odd columns into account; this modified algorithm is monotonic in the runs. If the value returned by this modified algorithm is already above $\frac{T'_2}{2} = 14$, then there is no need to further add runs. This efficiently cuts the search.

Before being added to the candidate set S, the parity pattern p is tested with the unmodified Algorithm 1. For the remaining parity patterns, we explicitly generated all states a with these parities up to $\tilde{\mathrm{w}}(\lambda(a)) \leq T'_2 = 28$. This allowed us to prove Lemma 3.

6.5 Starting from out-of-kernel states

For a given parity pattern p, we can construct all states $b = \lambda(a)$ with $P(a) = p$ and $\tilde{\mathrm{w}}(b) \leq T'_2 = 28$. We proceed in two phases.

- In a first phase, we generate all states a such that $P(a) = p$ by assigning all possible 16 values to affected (odd or even) columns and by assigning a single active bit in each unaffected odd column. These states are such that $||a|| + ||\lambda(a)||$ is exactly $10g + 2c$, with g the θ-gap and c the number of unaffected odd columns.
- In a second phase, we take the states generated in the first phase and add pairs of bits to all unaffected columns. By adding a pair of bits, we do not alter $P(a)$.

In both phases, we keep only the states $b = \lambda(a)$ for which $\tilde{\mathrm{w}}(b) \leq T'_2 = 28$. As can be seen in Table 3, both the weight and the reverse minimum weight are *monotonic*, i.e., adding an active bit to the state cannot decrease them. We can therefore limit the search by stopping adding pairs of bits when $\tilde{\mathrm{w}}(b)$ is above $T'_2 = 28$.

In practice, what we did was the following.

- Let $\mathcal{P}$ be the set of parity patterns satisfying one of the conditions of Lemma 3 except $p = 0$.

- By the method described above, we construct all states in the set $\mathcal{B} = \{b : P(\lambda^{-1}(b)) \in \mathcal{P}$ and $\tilde{\mathrm{w}}(b) \leq T_2' = 28\}$.
- Finally, we forward and backward extend the states in $\mathcal{B}$ to 3-round trail cores up to weight $T_3 = 36$.

We again found the same trail core as in Section 5. The trail prefix of weight 32 has $P(a_1) = 0$ (so a_1 is in the kernel) and $P(a_2)$ has one run of length 2 (so a_2 has θ-gap 1). No other trail cores were found.

When extending the states in $\mathcal{B}$, we exhaustively scan all compatible states, thereby including cases where $P(a_1) = 0$ or $P(a_2) = 0$. Hence, we covered the whole target space, except for trails such that both $P(a_1) = 0$ and $P(a_2) = 0$.

7 Generating in-kernel trails

To close the target space, we must look at in-kernel trails of the form in Eq. (2) with both $P(a_1) = 0$ and $P(a_2) = 0$. In the case of in-kernel trails, we were able to be completely cover the space up to weight $T_3 = 40$, and we expect the techniques presented here can cover trails of higher weight. As $P(a_1) = P(a_2) = 0$, the θ operation has no effect and therefore $b_i = \pi(\rho(a_i))$. So this comes down to looking for states $a = a_1$, $b = b_1$, $c = a_2$ and $d = b_2$ connected as:

$$a \xrightarrow{\pi\circ\rho} b \xrightarrow{\chi} c \xrightarrow{\pi\circ\rho} d, \text{ with } P(a) = P(c) = 0. \tag{3}$$

We now summarize how we can efficiently generate all in-kernel three-round trail cores up to some weight and provide more details in following subsections. The key element in our method is the observation that any state b with $P(a) = 0$ and for which there exists a state c with $P(c) = 0$ can be represented in a specific way. The states a and b are iteratively constructed by adding active bits in the form of bit sequences called chains and vortices, defined in Section 7.2 below. Chains and vortices have an even number of active bits per column in a by construction and hence ensure $P(a) = 0$.

In b, there can be zero, one or more slices called knots, which contain three or more active bits. Each of these active bits is the end point of a chain that leads to another knot or that connects back to the same knot. The intermediate active bits of a chain appear pairwise in slices holding exactly two active bits in one column (called orbital slices, see Section 7.1). On top of chains connecting knots, a state b can exhibit a vortex, i.e., a cyclic sequence of active bits that appear pairwise both in the columns of a and in the columns of b.

By starting with an empty state and progressively adding chains, knots and vortices, one can quickly build states a and b that satisfy $P(a) = 0$ and for which there exist c with $P(c) = 0$, leading to 3-round in-kernel trail cores. Any state leading to a in-kernel trail can be represented in this way, and care is taken so that all possible states are generated, up to a given target weight. At each step, a lower bound on the weight of 3-round trail cores containing a and b is computed so as to efficiently limit the search.

As a final step, the generated states a and b are forward-extended to states c and d, limiting to c values in the kernel. Thanks to the properties of χ (see Section 3.1), the compatible states c can be expressed as a linear affine space. It is thereby easy to take the intersection of this affine space with the set of states such that $P(c) = 0$.

7.1 Characterizing the slices in b

Definition 1. *A state b is* tame *if $P(\lambda^{-1}(b)) = 0$ and such that there exists at least one state c compatible with b through χ such that $P(c) = 0$.*

To characterize states b such that $P(c) = 0$, we can reason on the slices b_z of b since χ and P can be jointly described in terms of slices. In particular, each slice c_z of c must be in the kernel, namely, $P(c_z) = 0$, and we have to characterize the slices b_z under that constraint. First, if $b_z = 0$ then $c_z = 0$ and $P(c_z) = 0$. Then, a slice b_z with a single active bit cannot be in the kernel after χ, as at least one column of c_z will have a single active bit. Finally, a slice b_z with two active bits must have its two active bits in the same column for c_z to be in the kernel. By inspection of Table 3, a row with a single active bit at coordinate x, e.g., 00100 transforms into an active row of the form $uv100$ with $u, v \in \{0, 1\}$, so the active bit stays active at x and zero, one or two active bits can appear at $x - 2$ and $x - 1$ of the same row. So, if the two bits are not in the same column, one of the active bits that stays after χ will not find another active bit in the same column. We summarize this in the next lemma.

Lemma 4. *If b is tame, then each of its slices has either*

- *no active bit,*
- *two active bits in the same column, or*
- *three or more active bits.*

We call an *empty slice* a slice with no active bit, and an *orbital slice* is a slice with two active bits in the same column. A slice that is neither empty not an orbital slice is called a *knot*. We say that a knot is *tame* if it can transform after χ into a slice in the kernel. According to Lemma 4, a tame knot has at least three active bits.

7.2 Characterizing the set of active bits

Since in the kernel θ acts as the identity, the active bits of a are just moved to other positions in b and their number remains the same, i.e., $||a|| = ||b||$. We can therefore represent a and b by a list of active bit positions $(p_i)_{i=1\ldots||a||}$ in either the coordinates (x_i, y_i, z_i) in a or the coordinates (x'_i, y'_i, z'_i) in b, with $(x_i, y_i, z_i) \xrightarrow{\pi\circ\rho} (x'_i, y'_i, z'_i)$.

First, we start with the active bits in a. We say that active bits p_i and p_j are *peer* if they are in the same column in a, i.e., $x_i = x_j$ and $z_i = z_j$. Since each column has an even number of active bits when $P(a) = 0$, an active bit thus always has a peer.[1]

Then, we move to the active bits in b. We say that the two active bits p_i and p_j are *chained* if they both lie in the same orbital slice in b. So $x'_i = x'_j$ and $z'_i = z'_j$ and no other active bit is in slice z'_i.

A *chain* is a sequence of bit positions of even length $(p_0, p_1, p_2, \ldots, p_{2n-1})$ such that p_{2k} and p_{2k+1} are peer ($\forall k \in \{0, \ldots, n-1\}$) and that p_{2k+1} and p_{2k+2} are chained ($\forall k \in \{0, \ldots, n-2\}$). In addition, the first and last active bits p_0 and p_{2n-1} must be in knots (either the same one or different ones). The simplest possible chain has length 2 and consists only in two peer active bits.

The definition of a *vortex* is the same as that of a chain $(p_0, p_1, p_2, \ldots, p_{2n-1})$, except that the first and last active bits p_0 and p_{2n-1} must be chained. In other words, a vortex forms a cycle of bit positions linked alternatively by peer and chained relationships, all in orbital slices.

In a tame state, each active bit position has exactly one peer position. The active bit positions in knots are the end points of chains, while the active bits in orbital slices are chained and belong to chains or vortices. Therefore, any tame state can be represented as a set of vortices and chains connecting knots.

[1] While for columns with two active bits, the peer relationship is unambiguous, in the case of columns with four active bits, we choose which pairs of active bits are peer. Thus we can see the representation of the states as being augmented with additional attributes specifying the peer relationship and there may be several ways to represent the same state. By generating states via this representation, the only risk is to generate more states than necessary.

7.3 Generating all tame states

To generate all tame states up to a target weight T_3, we generate states a and b by representing them using the concepts of Sections 7.1 and 7.2. The generation builds (initially empty) states a and b by iterating the following nested loops:

- In the outer loop, we add chains to the existing state. When adding a chain $(p_0, p_1, p_2, \ldots, p_{2n-1})$, the slices that receive the end points p_0 and p_{2n-1} must become knots if they are not already. If $n > 1$, the pairs of (chained) active bits (i_{2k+1}, i_{2k+2}) are added to empty slices, which become orbital slices. Active bits cannot be added to already constructed orbital slices, as it would contradict the definition of an orbital slice. Enough chains must be added such that each knot contains at least 3 active bits (see Lemma 4).
- For a fixed set of chains produced in the previous step, the inner loop iterates on the number and position of vortices. In a vortex, all active bits are chained, so they must be added to empty slices, which become orbital slices.

With the monotonic lower bound function defined in the next section, we add chains and vortices until this lower bound exceeds T_3.

7.4 Lower-bounding the weight of in-kernel trails

We wish to determine a lower bound on the weight of 3-round in-kernel trail cores (b, d), namely, on $\mathrm{w}^{\mathrm{rev}}(a) + \mathrm{w}(b) + \mathrm{w}(d)$ with $a = \lambda^{-1}(b)$, from a and b only, for use in our trail generation. Since only d is unknown, this implies finding a lower bound on $\mathrm{w}(d)$. This can be done by first determining a lower bound on the Hamming weight $||d||$ and then bounding the weight of any state with given Hamming weight.

To determine a lower-bound on $||d||$, we work on each slice of b. If slice b_z has $u = \|b_z\|_{\mathrm{row}}$ active rows, then the slice c_z has at least u active bits. In addition, $P(c_z) = 0$ implies that the number of active bits must be even, so $||c_z|| \geq 2\lceil \frac{u}{2} \rceil$. Finally, we have $||d|| = ||c||$ so

$$||d|| \geq 2 \sum_z \left\lceil \frac{\|b_z\|_{\mathrm{row}}}{2} \right\rceil .$$

From Table 3, it is easy to verify the following lower bound:

$$\mathrm{w}(d) \geq \hat{\mathrm{w}}(||d||) \triangleq \left\lceil \frac{4||d||}{5} \right\rceil + [1 \text{ if } ||d|| = 1 \text{ or } 2 \pmod 5].$$

Hence, we define the *lower weight* of b as

$$\mathrm{L}(b) \triangleq \mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b)) + \mathrm{w}(b) + \hat{\mathrm{w}}\left(2 \sum_z \left\lceil \frac{\|b_z\|_{\mathrm{row}}}{2} \right\rceil \right) .$$

The lower weight yields a lower bound on the weight of 3-round in-kernel trail cores (b, d) regardless of d.

7.5 Limiting the search by lower-bounding the weight

At each level of the loop described in Section 7.3, the corresponding iteration is aborted, and elements are not further added, if we can be sure that the lower weight $\mathrm{L}(b)$ will become larger than the target weight T_3. Adding a chain to the state can potentially bring new knots and/or new orbital slices. Adding a vortex necessarily brings new orbital slices. Therefore, there is a

Number	$\tilde{\mathrm{w}}(\cdot)$	$\mathrm{w}^{\mathrm{rev}}(b_1)$	$\mathrm{w}(b_1)$	$\mathrm{w}(b_2)$	$P(a_1)$	$P(a_2)$	Structure of a_1, b_1
1	32	4	4	24	kernel	θ-gap 1	
1	35	12	12	11	kernel	kernel	vortex of length 6
7	36	12	12	12	kernel	kernel	vortex of length 6
7	39	12	12	15	kernel	kernel	vortex of length 6
2	39	12	11	16	kernel	kernel	2 knots connected by 3 chains
41	40	12	12	16	kernel	kernel	vortex of length 6
4	40	12	12	16	kernel	kernel	2 knots connected by 3 chains

Table 4. Summary of all 3-round differential trail cores found in Keccak-f[1600] up to weight 36, and up to weight 40 for in-kernel trails. The number indicates the number of cores with the same properties indicated in the other columns.

limit in the number of knots and orbital slices that must be considered for the generation to be complete up to the target weight.

As a preliminary step, the minimum reverse weight satisfies the following inequality (see Table 3):

$$\mathrm{w}^{\mathrm{rev}}(a) \geq \hat{\mathrm{w}}^{\mathrm{rev}}(||a||) \triangleq \left\lceil \frac{3||a||}{5} \right\rceil .$$

We see from Lemma 4 that each tame knot contributes to at least 3 active bits in a and in b. Furthermore, the number of bits in each slice of a must be even ($P(a) = 0$), so $||a|| \geq 2 \left\lceil \frac{3k}{2} \right\rceil$ and $\mathrm{w}^{\mathrm{rev}}(a) \geq \hat{\mathrm{w}}^{\mathrm{rev}}(||a||)$, with k the number of knots. In b, each tame knot has at least 3 active bits on at least 2 different active rows, hence contributing at least 5 to the weight, and so $\mathrm{w}(b) \geq 5k$. Each active row in b contributes to at least one active bit in d so $||d|| \geq 2k$ and $\mathrm{w}(d) \geq \hat{\mathrm{w}}(||d||)$.

For instance, $k = 5$ knots implies that $||a|| \geq 16$ and $\mathrm{w}^{\mathrm{rev}}(a) \geq \hat{\mathrm{w}}^{\mathrm{rev}}(16) = 10$, that $\mathrm{w}(b) \geq 25$ and that $||d|| \geq 10$ and $\mathrm{w}(d) \geq \hat{\mathrm{w}}(10) = 8$, so a lower weight of at least 43. If $T_3 \leq 42$, looking for configurations with from 0 to 4 knots is therefore sufficient, not even counting the orbital slices that also compose chains.

We found cores of weight 35, 36, 39 and 40, as detailed in Table 4. For illustration purposes, examples of trail prefixes are shown in Figures 3, 4 and 5 in Appendix A.

8 Extension to six-round trails

Table 4 summarizes all the 3-round cores found. These trail cores completely represent all the 3-round trails up to weight 36 (or 40 for in-kernel trails).

The second phase introduced in Section 4 consists in exhaustively extending forward and backward all the 3-round trail cores into 6-round trails cores. As no 6-round trail of weight up to 73 were found, we conclude that a 6-round differential trail in Keccak-f[1600] has at least weight 74. In the specific case of in-kernel trails, no 6-round trail of weight up to 81 were found and we conclude that a 6-round in-kernel differential trail in Keccak-f[1600] has at least weight 82.

For the 24 rounds of Keccak-f[1600], a differential trail has at least weight 296, and an in-kernel trail has at least weight 328.

9 Conclusions

We studied and implemented the exhaustive generation of 3-round differential trails in the Keccak-f[1600] permutation, which allowed us to prove a lower bound on the weight of differential trails. The techniques developed in this paper exploit the properties of the mixing layer in its round function to provide better bounds than what a brute-force method could provide. Table 2 shows that there remains a gap between the best known trails and the lower bound beyond three rounds that calls for future work. Finally, the concepts introduced in this paper, such as chains, vortices, knots and parity runs, help read trails and understand them.

References

1. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *On the indifferentiability of the sponge construction*, Advances in Cryptology – Eurocrypt 2008 (N. P. Smart, ed.), Lecture Notes in Computer Science, vol. 4965, Springer, 2008, `http://sponge.noekeon.org/`, pp. 181–197.
2. ______, *Cryptographic sponge functions*, January 2011, `http://sponge.noekeon.org/`.
3. ______, KeccakTools *software*, September 2011, `http://keccak.noekeon.org/`.
4. ______, *On alignment in* keccak, ECRYPT II Hash Workshop 2011, 2011.
5. ______, *The* Keccak *reference*, January 2011, `http://keccak.noekeon.org/`.
6. E. Biham and A. Shamir, *Differential cryptanalysis of DES-like cryptosystems*, CRYPTO (A. Menezes and S. Vanstone, eds.), Lecture Notes in Computer Science, vol. 537, Springer, 1990, pp. 2–21.
7. J. Daemen, M. Peeters, G. Van Assche, and V. Rijmen, *Nessie proposal: the block cipher* Noekeon, Nessie submission, 2000, `http://gro.noekeon.org/`.
8. J. Daemen and V. Rijmen, *The design of Rijndael — AES, the advanced encryption standard*, Springer-Verlag, 2002.
9. ______, *Plateau characteristics and AES*, IET Information Security **1** (2007), no. 1, 11–17.
10. I. Dinur, O. Dunkelman, and A. Shamir, *New attacks on Keccak-224 and Keccak-256*, Fast Software Encryption 2012, 2012, to appear, draft available from Cryptology ePrint Archive, Report 2011/624.
11. A. Duc, J. Guo, T. Peyrin, and L. Wei, *Unaligned rebound attack: Application to Keccak*, Fast Software Encryption 2012, 2012, to appear, draft available from Cryptology ePrint Archive, Report 2011/420.
12. P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schläffer, and S. S. Thomsen, *Grøstl – a SHA-3 candidate*, Submission to NIST (round 3), 2011.
13. E. Heilman, *Restoring the differential security of MD6*, ECRYPT II Hash Workshop 2011, 2011.
14. M. Naya-Plasencia, A. Röck, and W. Meier, *Practical analysis of reduced-round Keccak*, Indocrypt 2011, 2011.
15. NIST, *Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family*, Federal Register Notices **72** (2007), no. 212, 62212–62220, `http://csrc.nist.gov/groups/ST/hash/index.html`.
16. R. Rivest, B. Agre, D. V. Bailey, S. Cheng, C. Crutchfield, Y. Dodis, K. E. Fleming, A. Khan, J. Krishnamurthy, Y. Lin, L. Reyzin, E. Shen, J. Sukha, D. Sutherland, E. Tromer, and Y. L. Yin, *The MD6 hash function – a proposal to NIST for SHA-3*, Submission to NIST, 2008, `http://groups.csail.mit.edu/cis/md6/`.
17. H. Wu, *The hash function JH*, Submission to NIST (round 3), 2011.

A Some three-round differential trails

In this section, we give some examples of trails for illustration purposes. In the figures, trail prefixes are depicted with b_0, a_1, b_1, a_2, b_2 from top to bottom as in Eq. (2) and the weight of each round is given before χ. The difference b_0 was taken such that $\mathrm{w}(b_0) = \mathrm{w}^{\mathrm{rev}}(a_1)$. At each step, only the slices with non-zero difference are shown with their z coordinate. The x coordinate goes from left to right with $x = 0$ at the center, while the y coordinate goes from bottom to top with $y = 0$ at the center. Active bits are depicted in black.

When $P(a_1) = 0$, the peer and chained relationships are shown with straight and dashed lines, respectively. Examples of structures include a vortex of length 6, two knots connected by three chains, and one knot connected to itself by two chains. In Figure 2, $P(a_2) \neq 0$ and the effect of θ is illustrated in details.

B A four-round differential trail

Figure 6 shows a 4-round differential trail of weight 134. This is the differential 4-round trail on Keccak-f[1600] with the lowest known weight at this time of writing. The uneven weight profile (16, 13, 12, 93) suggests that trails with lower weight exist.

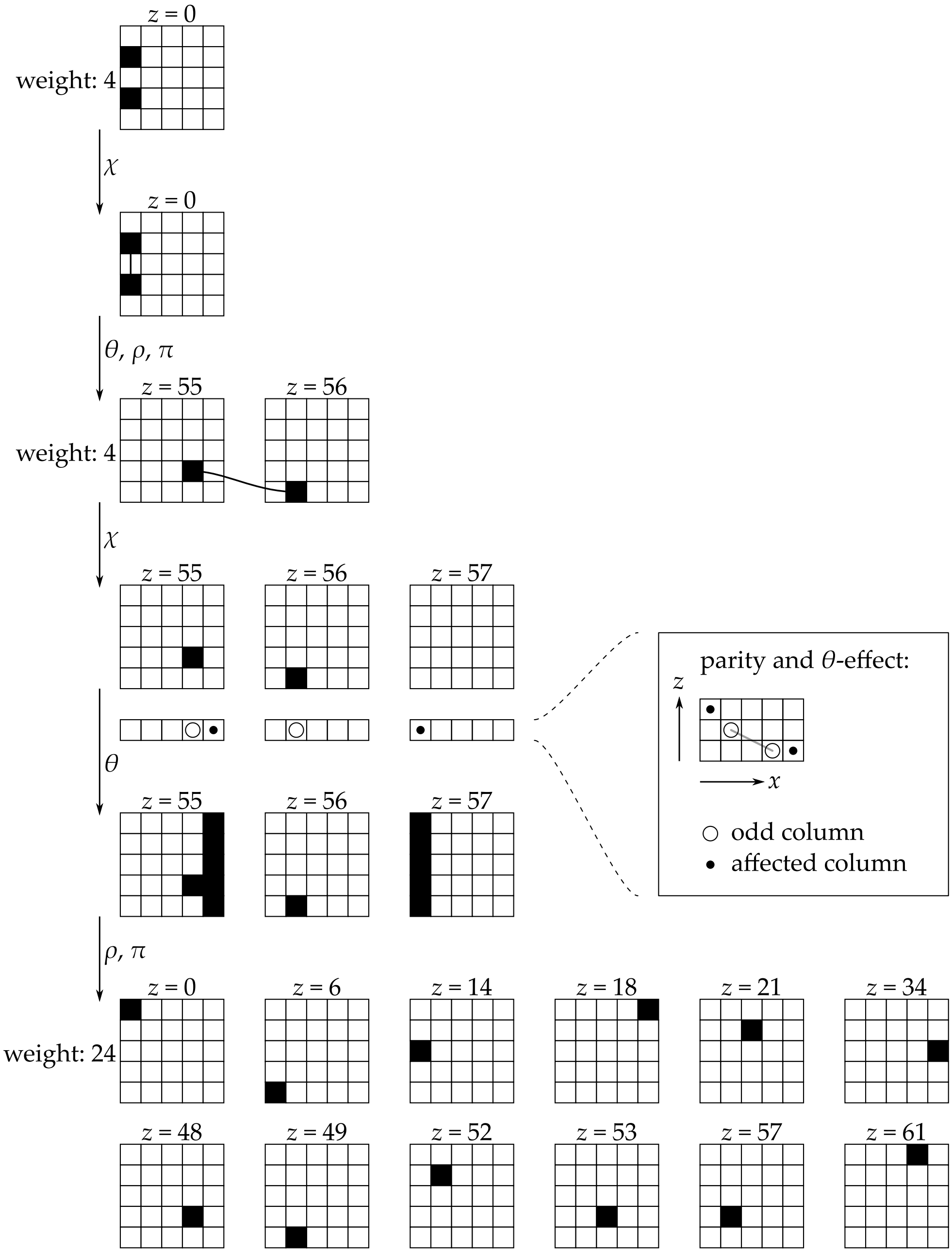


Fig. 2. Trail prefix of weight 32. It contains the 3-round trail core with smallest weight. The state a_1 is in the kernel. A chain of length 2 connects the knots in $z' = 55$ and $z' = 56$. However, these knots are not tame and a_2 cannot be in the kernel. Instead, a_2 has θ-gap 1 and contains a run of length 2 with odd columns in $(x, z) \in \{(1, 55), (4, 56)\}$. The columns $(x, z) \in \{(2, 55), (3, 57)\}$ are affected and hence are flipped by θ.

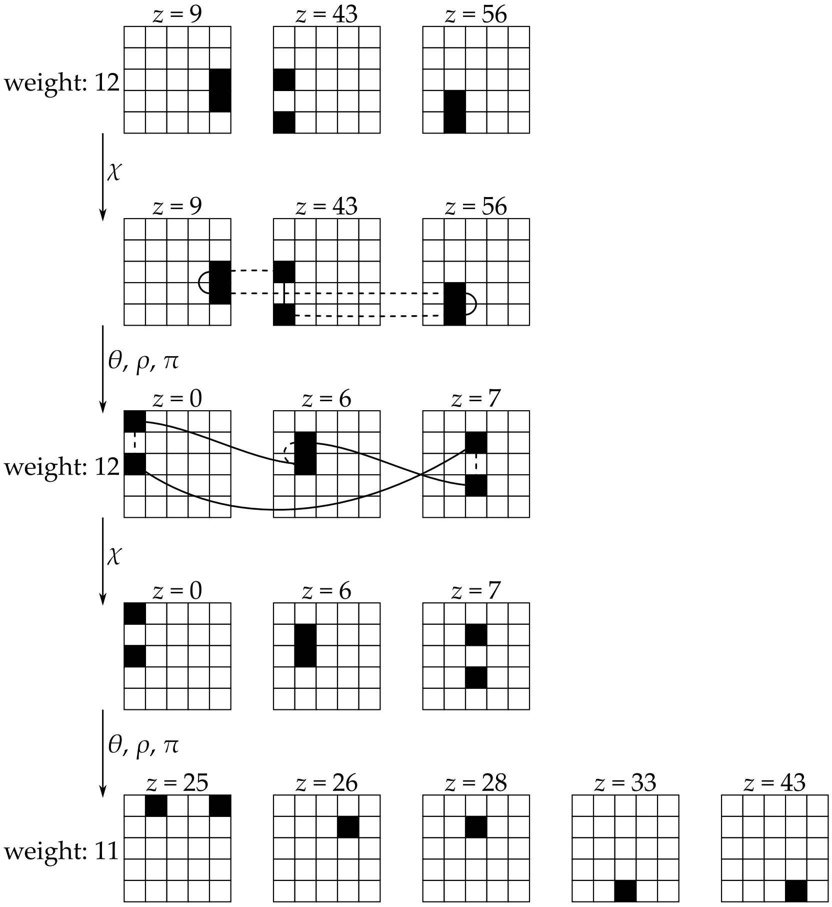


Fig. 3. Trail prefix of weight 35. It contains a vortex of length 6 in orbital slices $z' \in \{0, 6, 7\}$.

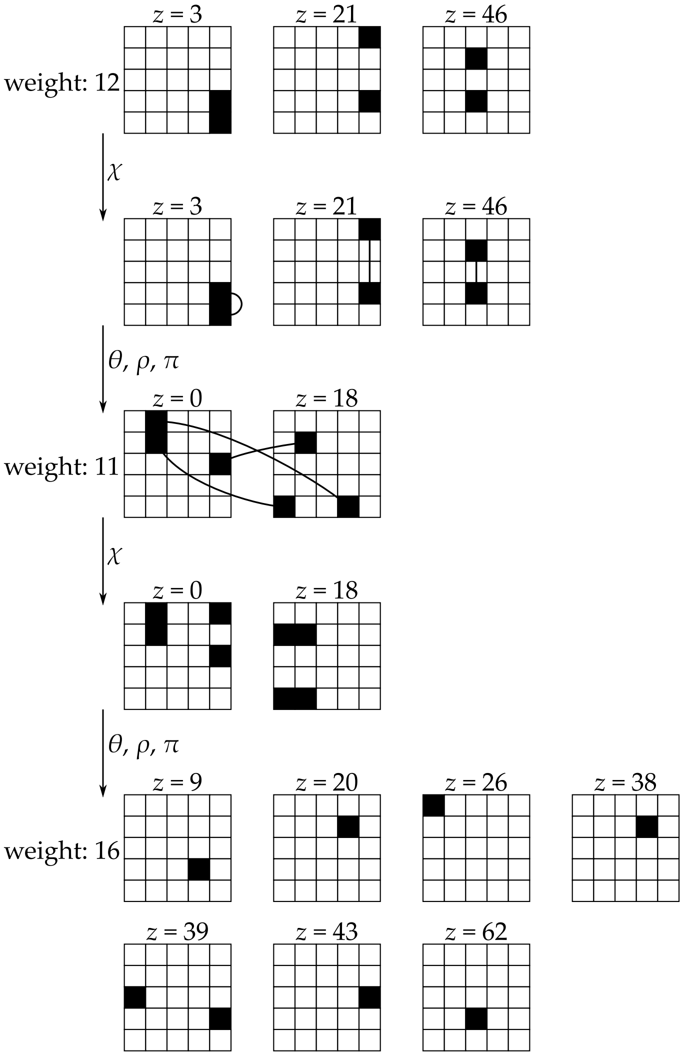


Fig. 4. Trail prefix of weight 39. It contains two knots, one in $z' = 0$ and the other in $z' = 18$. The knots are connected with three chains of length 2, ensuring that each knot has three active bits. There are no orbital slices.

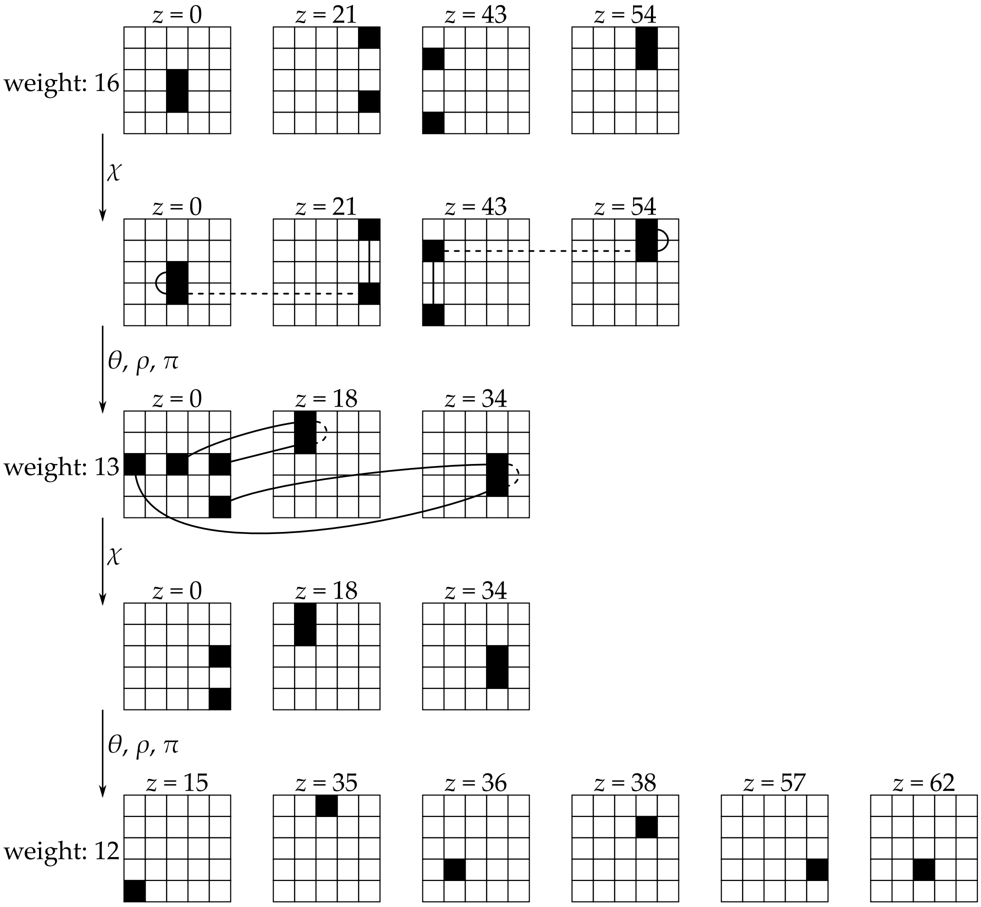


Fig. 5. Trail prefix of weight 41. It contains a single knot in $z' = 0$. Two chains of length 4 connect this knot to itself, which has four active bits. The chains go through orbital slices $z' = 18$ and $z' = 34$.

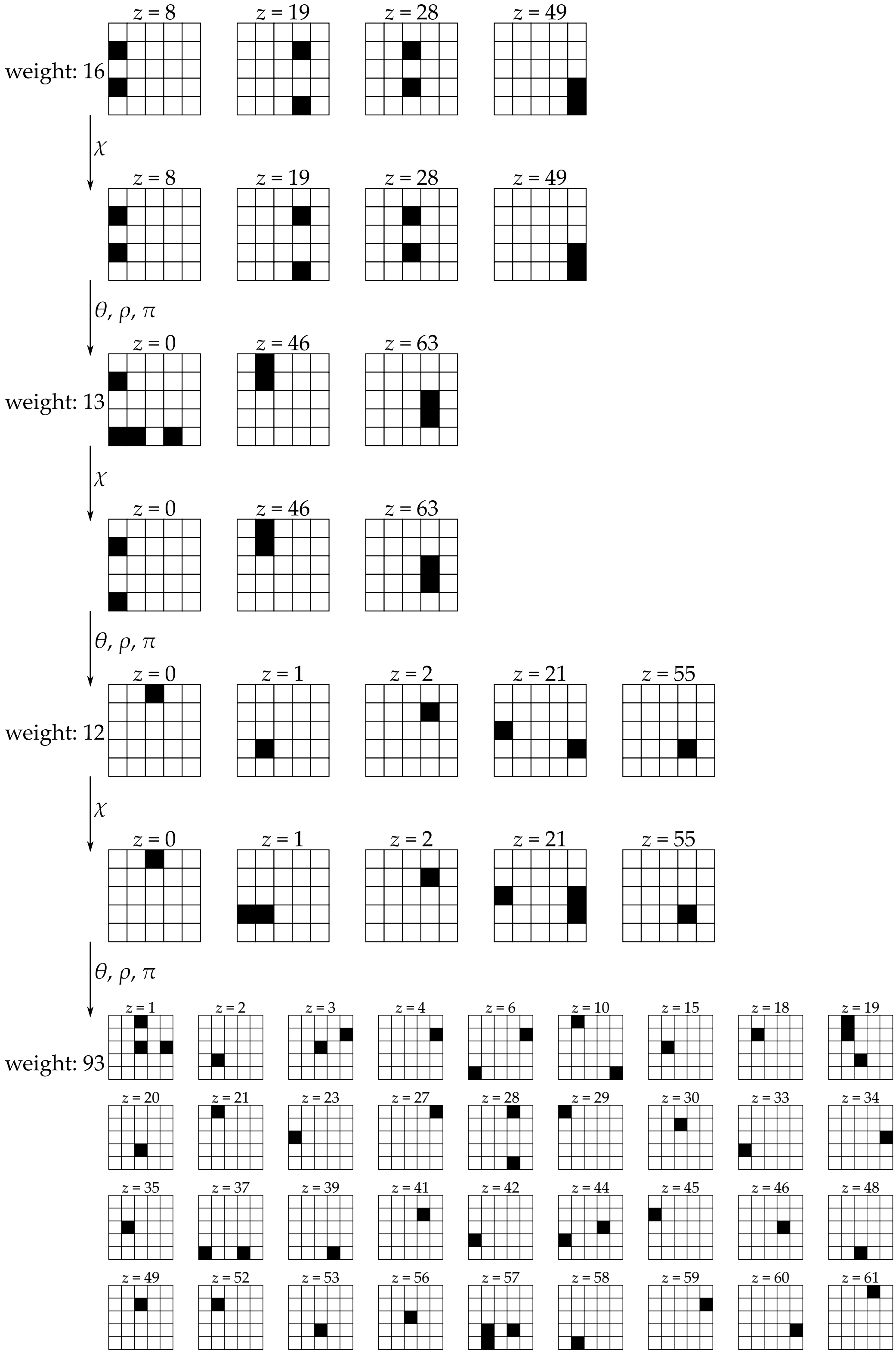


Fig. 6. A 4-round trail prefix of weight 134.

New attacks on Keccak-224 and Keccak-256

Itai Dinur[1], Orr Dunkelman[1,2] and Adi Shamir[1]

[1] Computer Science department, The Weizmann Institute, Rehovot, Israel
[2] Computer Science Department, University of Haifa, Israel

Abstract. The Keccak hash function is one of the five finalists in NIST's SHA-3 competition, and so far it showed remarkable resistance against practical collision finding attacks: After several years of cryptanalysis and a lot of effort, the largest number of Keccak rounds for which actual collisions were found was only 2. In this paper we develop improved collision finding techniques which enable us to double this number. More precisely, we can now find within a few minutes on a single PC actual collisions in standard Keccak-224 and Keccak-256, where the only modification is to reduce their number of rounds to 4. When we apply our techniques to 5-round Keccak, we can get in a few days excellent near collisions, where the Hamming distance is 5 in the case of Keccak-224 and 10 in the case of Keccak-256. Our new attack combines differential and algebraic techniques, and uses the fact that each round of Keccak is only a quadratic mapping in order to efficiently find pairs of messages which follow a high probability differential characteristic.

Keywords: Cryptanalysis, SHA-3, Keccak, collision, near-collision, practical attack.

1 Introduction

The Keccak hash function [4] uses the sponge construction [3] to map arbitrary long inputs into fixed length outputs, and is one of the five finalists of NIST's SHA-3 competition. The Keccak versions submitted to the SHA-3 competition have an internal state size of $b = 1600$ bits, and an output size n of either 224, 256, 384 or 512 bits. The internal permutation of Keccak consists of 24 application of a non-linear round function, applied to the 1600-bit state. Previous papers on Keccak, such as [13], include analysis of Keccak versions with a reduced internal state size, or with different output sizes. However, in this paper, we concentrate on the standard Keccak versions submitted to the SHA-3 competition, and the only way in which we modify them is by reducing their number of rounds.

Previous results on Keccak's internal permutation include zero-sum distinguishers presented in [1], and later improved in [5, 6, 8]. Although zero-sum distinguishers reach a significant number of rounds of Keccak's internal permutation, they have very high complexities, and they seem unlikely to threaten the core security properties of Keccak (namely, collision resistance, preimage resistance and second-preimage resistance). Other results on Keccak's internal permutation include a differential analysis given in [9]. Using techniques adapted from the rebound attack [12], the authors construct differential characteristics which give distinguishers on up to 8 rounds of the permutation, with complexity of about 2^{491}. However, in their method it is not clear how to reach the starting state differences of these characteristics from valid initial states of Keccak's internal permutation, since in sponge constructions a large portion of the initial state of the permutation is fixed and cannot be chosen by the cryptanalyst. Thus, although the results of [9] seem to be more closely related to the core security properties of Keccak than zero-sum distinguishers, they still do not lead to any attacks on the Keccak hash function itself.

Currently, there are very few results that analyze reduced-round variants of the full Keccak (rather than its building blocks): in [2], Bernstein described preimage attacks which extend up to 8 rounds of Keccak, but are only marginally faster than exhaustive search, and use a huge amount of memory. More recently, Naya-Plasencia, Röck and Meier presented practical attacks on Keccak-224 and Keccak-256 with a very small number of rounds [14]. These attacks include a preimage attack on 2 rounds, as well as collisions on 2 rounds and near-collisions on 3 rounds.

In this paper, we extend these collision attacks on Keccak-224 and Keccak-256 by 2 additional rounds: we find actual collisions in 4 rounds and actual near-collisions in 5 rounds of Keccak-224 and Keccak-256, with Hamming distance 5 and 10, respectively.

The collisions and near-collisions of [14] were obtained using low Hamming weight differential characteristics, starting from the initial state of Keccak's permutation. Such low Hamming weight characteristics are also the starting point of our new attacks, but we do not require the characteristics to start from the initial state of the permutation. Given a low Hamming weight starting state difference of a characteristic, we can easily extend it backwards by one round, and maintain its high probability (as done in [9]). However, due to the very fast diffusion of the inverse linear mapping used by Keccak's permutation, the new starting state difference of the extended characteristic has a very high Hamming weight. We call this starting state difference a *target difference*, since our goal is to find message pairs which have this difference after one round of the Keccak permutation (after the fixed round, this difference will evolve according to the characteristic with high probability).[1] One of the main tools we develop in this paper is an algorithm that aims to achieve this goal, namely, to find message pairs which satisfy a given target difference after one Keccak permutation round. We call this algorithm a *target difference algorithm*, and it allows us to extend our initial characteristic by two additional rounds (as shown in Figure 1): we first extend the characteristic backwards by one round to obtain the target difference (while maintaining the characteristic's high probability). Then, we use the target difference algorithm to link the characteristic to the initial state of Keccak's permutation, through an additional round. We note that the final link, which efficiently bypasses Keccak's first Sbox layer, uses algebraic techniques rather than standard probabilistic techniques.

The target difference algorithm is related to several hash function cryptanalytic techniques that were developed in recent years. In particular, it is related to the work of Khovratovich, Biryukov and Nikolic [11], where, similarly to our algorithm, the authors use linear algebra to quickly satisfy many conditions of a differential characteristic. However, these techniques seem to work best on byte-oriented hash functions, whose internal structure can be described using a few sparse equations, which is not the case for Keccak. Our algorithm is also closely related to the work of Khovratovich [10] that exploits structures (which aggregate internal states of the hash function) in order to reduce the amortized complexity of collision attacks: the attacker first finds a truncated differential characteristic and searches for a few pairs of initial states that satisfy it. Then, using the structures and the initially found pairs, the attacker efficiently obtains many additional pairs that satisfy the truncated characteristic. However, in the case of Keccak, there are very few characteristics that can lead to a collision with high probability, and it seems unlikely that they can be joined in order to form the truncated differential characteristic required in order to organize the state differences into such structures. Moreover, it seems difficult to find even one pair of initial states that satisfy the target difference for Keccak. Another attack related to the target difference algorithm is the rebound attack [12]. In this attack, the cryptanalyst uses the available degrees of freedom to efficiently link and extend two truncated differential characteristics, both forwards and backwards, from an intermediate state of the hash function. However, once again, such high probability truncated characteristics are unlikely to exist for Keccak. Moreover, it is not clear how to use the rebound attack to link the backward characteristic to the initial state of the permutation. Thus, our target difference algorithm can be viewed as an asymmetric rebound attack, where one side of the characteristic is fixed.

Our full attacks have two parts, where in the first part we execute the target difference algorithm in order to obtain a sufficiently large set of message pairs that satisfy the target

[1] We note that the target difference is not a valid initial difference of the permutation, which fixes many of the state bits to pre-defined values. As a result, the high probability characteristic cannot be used to extend the results of [14] by an additional round.

difference after the first round. In the second part of the attack, we try different message pairs in this set in order to find a pair whose difference evolves according to a characteristic whose starting state is the target difference. Since the target difference algorithm does not control the differences beyond the first round, the second part of the attack is a standard probabilistic differential attack (which only searches for collisions or near-collisions obtained from message pairs within a specific set). The high probability differential characteristic beyond the first round ensures that the time complexity of the second part of the attack is relatively low.

Although the target difference algorithm is heuristic, and there is no provable bound on its running time, it was successfully applied with its expected complexity to many target differences defined by the high probability differential characteristics. Consequently, we were able to find actual collisions for 4 rounds of Keccak-224 and Keccak-256 within minutes on a standard PC. By using good differential characteristics for an additional round, we found near-collisions for 5 rounds of Keccak-224 and Keccak-256. However, this required more computational effort (namely, a few days on a single PC), since the extended characteristics have lower probabilities.

The paper is organized as follows. In Section 2, we briefly describe Keccak, and in Section 3 we introduce our notations. In Section 4, we give a comprehensive overview of the target difference algorithm and describe the properties of Keccak that it exploits. In Section 5, we present our results on round-reduced Keccak. In Appendix A, we describe the full details of the target difference algorithm, and in Appendix B, we propose an alternative algorithm, which has a better understood time complexity. Since the original algorithm gave us very good results in practice, we did not use this alternative version. However, it may be more efficient in some cases, especially if someone finds longer high probability characteristics for Keccak's permutation.

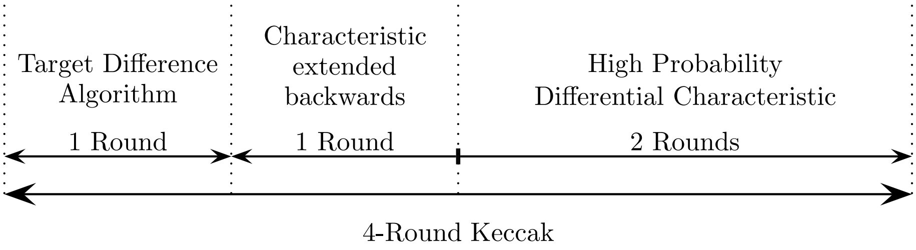

Target Difference Algorithm
Characteristic extended backwards
High Probability Differential Characteristic
1 Round
1 Round
2 Rounds
4-Round Keccak

Fig. 1. Extending a 2-Round Differential Characteristic by Two Additional Rounds

2 Description of Keccak

In this section we give short descriptions of the sponge construction and the Keccak hash function. More details can be found in the Keccak specification [4].

The sponge construction [3] works on a state of b bits, which is split into two parts: the first part contains the first r bits of the state (called the outer part of the state) and the second part contains the last $c = b - r$ bits of the state (called the inner part of the state).

Given a message, it is first padded and cut into r-bit blocks, and the b state bits are initialized to zero. The sponge construction then processes the message in two phases: In the absorbing phase, the message blocks are processed iteratively by XORing each block into the first r bits of the current state, and then applying a fixed permutation on the value of the b-bit state. After processing all the blocks, the sponge construction switches to the squeezing phase. In this phase,

n output bits are produced iteratively, where in each iteration the first r bits of the state are returned as output and the permutation is applied.

The Keccak hash function uses multi-rate padding: given a message, it first appends a single 1 bit. Then, it appends the minimum number of 0 bits followed by a single 1 bit, such that the length of the result is a multiple of r. Thus, multi-rate padding appends at least 2 bits and at most $r + 1$ bits.

The Keccak versions submitted to the SHA-3 competition have $b = 1600$ and $c = 2n$, where $n \in \{224, 256, 384, 512\}$. The 1600-bit state can be viewed as a 3-dimensional array of bits, a[5][5][64], and each state bit is associated with 3 integer coordinates, a[x][y][z], where x and y are taken modulo 5, and z is taken modulo 64.

The Keccak permutation consists of 24 rounds, which operate on the 1600 state bits. Each round of the permutation consists of five mappings $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$. Keccak uses the following naming conventions, which are helpful in describing these mappings:

- A row is a set of 5 bits with constant y and z coordinates, i.e. $a[*][y][z]$.
- A column is a set of 5 bits with constant x and z coordinates, i.e. $a[x][*][Z]$.
- A lane is a set of 64 bits with constant x and y coordinates, i.e. $a[x][y][*]$.
- A slice is a set of 25 bits with a constant z coordinate, i.e. $a[*][*][z]$.

The five mappings are given below, for each x,y, and z (where the state addition operations are over $GF(2)$):

1. θ is a linear map, which adds to each bit in a column, the parity of two other columns.
   θ: $a[x][y][z] \leftarrow a[x][y][z] + \sum_{y'=0}^{4} a[x-1][y'][z] + \sum_{y'=0}^{4} a[x+1][y'][z-1]$
   In this paper, we also use the inverse mapping, θ^{-1}, which is more complicated and provides much faster diffusion: for θ^{-1}, flipping the value of any input bit, flips the value of more than half of the output bits.
2. ρ rotates the bits within each lane by T(x,y), which is a predefined constant for each lane.
   ρ: $a[x][y][z] \leftarrow a[x][y][z + T(x, y)]$
3. π reorders the lanes.
   π: $a[x][y][z] \leftarrow a[x'][y'][z]$, where $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} x' \\ y' \end{pmatrix}$
4. χ is the only non-linear mapping of Keccak, working on each of the 320 rows independently.
   χ: $a[x][y][z] \leftarrow a[x][y][z] + ((\neg a[x+1][y][z]) \wedge a[x+2][y][z])$
   Since χ works on each row independently, in can be viewed as an Sbox layer which simultaneously applies the same 5 bits to 5 bits Sbox to the 320 rows of the state. We note that the Sbox function is an invertible mapping, and we will use the extremely important observation that the algebraic degree of each output bit of χ as a polynomial in the five input bits is only 2. We also note that the algebraic degree the inverse mapping χ^{-1} is 3 (as noted in [4]).
5. ι adds a round constant to the state.
   ι: $a \leftarrow a + RC[i_r]$
   We omit the values of $RC[i_r]$, as they are not needed for our analysis.

3 Notations

Given a message M, we denote its length in bits by $|M|$. Unless specified otherwise, in this paper we assume that $|M| = r - 8$, namely we consider only single-block messages of maximal length such that $|M|(modulo\ 8) \equiv 0$ (which give us the maximal number of degrees of freedom, for single-block messages containing an integral number of bytes). Given M, we denote the initial state of the Keccak permutation as the 1600-bit word $\overline{M} \triangleq M||p||0^{2n}$, where $||$ denotes concatenation, and p denotes the 8-bit pad 10000001.

The first three operations of Keccak's round function are linear mappings, and we denote their composition by $L \triangleq \rho \circ \pi \circ \theta$. We denote the Keccak nonlinear function on 5-bit words defined by varying the first index by $\chi_{|5}$. The difference distribution table (DDT) of this function is a two-dimensional 32×32 integer table, where all the differences are assumed to be over $GF(2)$. The entry $DDT(\delta^{in}, \delta^{out})$ specifies the number of input pairs to this Sbox with difference δ^{in} that give the output difference δ^{out} (i.e., the size of the set $\{x \in \{0,1\}^5 \mid \chi_{|5}(x) + \chi_{|5}(x + \delta^{in}) = \delta^{out}\}$).

We denote the 1600-bit target difference, which is the input of the target difference algorithm, by Δ_T. The output of the algorithm is a subset of ordered pairs of single block messages $\{(M_1^1, M_1^2), (M_2^1, M_2^2), ..., (M_k^1, M_k^2)\}$ that satisfy this difference after one round R, namely $R(\overline{M}_i^1) + R(\overline{M}_i^2) = \Delta_T\ \forall i \in \{1, 2, ..., k\}$.

4 Overview of the Target Difference Algorithm

When designing the target difference algorithm, we face two problems: first, the target difference extends backwards, beyond the first Keccak Sbox layer, with very low probability (due to its high Hamming weight). The second problem is that the initial state of the permutation fixes many of the state bits to pre-defined values, and the initial states that we use must satisfy these constraints. On the other hand, Keccak has several useful properties that we can exploit in our target difference algorithm. In this section, we describe these properties in detail and give an overview of the algorithm.

4.1 The Properties of Keccak Exploited by the Target Difference Algorithm

The First Property Keccak-224 and Keccak-256 allow the user to control many of the 1600 state bits of the initial state of the permutation. Thus, given a target difference, we expect many solutions to exist (namely, one-block message pairs which have the 1600-bit target difference after one permutation round): since we consider message pairs, where each message is of length $r - 8 = 1600 - 8 - 2n$ bits (1144 for Keccak-224, and 1080 for Keccak-256), given an arbitrary 1600-bit target difference, there is an expected number of $2^{2(1600-8-2n)-1600} = 2^{1584-4n}$ message pairs of this length that satisfy this difference (regardless of the value of the inner part of the state). Thus, the algorithm has 704 and 560 degrees of freedom for Keccak-224 and Keccak-256, respectively.

Despite the large number of available degrees of freedom, the number of possible solutions varies significantly according to the target difference. To demonstrate this, we use the fact that L^{-1} has very fast diffusion (i.e., even an input with one non-zero bit is mapped by L^{-1} into a roughly balanced output). We consider the case where $t > 0$ out of the 320 Sboxes of the target difference are active (i.e., they have a non-zero output difference). Each one of the $320 - t$ non-active Sbox zero output differences is uniquely mapped backwards to a zero input difference into the first Sbox layer. Using the Keccak Sbox DDT, it is easy to see that each one of the t active Sbox output differences is mapped to more than 8 possible input differences. Thus, the number of possible state differences after the first linear layer (or before the first Sbox layer) is more than $8^t = 2^{3t}$. Since L is invertible and acts deterministically on the differences, the number of possible input differences to the Keccak compression function remains the same. We now recall from the first difference constraint in Section 4, that we require that the $2n + 8$ MSBs of Δ_I are zero. However, for t large enough, we still expect more than $2^{3t-2n-8}$ valid solutions. When the target difference is chosen at random, we have $t \approx 310$ (since the probability that an Sbox output difference is zero is $\frac{1}{32}$). This gives more than $2^{930-448-8} = 2^{474}$ expected solutions for Keccak-224, and more than $2^{930-512-8} = 2^{410}$ expected solutions for Keccak-256. On the other hand, consider the extreme case of $t = 1$ (i.e., the target difference has only one active Sbox). Clearly, this Sbox cannot contribute more than 31 possible differences after the first linear layer. Since L^{-1} has very fast diffusion, these possible differences are mapped to at most 31 roughly balanced non-zero possible input differences, and we do not expect the $2n + 8$ MSBs of any of them to be zero. To conclude, target differences with a small number of active Sboxes are likely to have no solutions at all. On the other hand, a majority of the target differences have a very large number of expected solutions for Keccak-224 and Keccak-256. Note that having a large number of solutions does not imply that it is easy to find any one of them, since their density is still minuscule.

The Second Property The algebraic degree of the Keccak Sboxes is only 2. This implies that given a 5-bit input difference δ^{in} and a 5-bit output difference δ^{out}, the set of values $\{v_1, v_2, ..., v_l\}$ such that $\chi_{|5}(v_i) + \chi_{|5}(v_i + \delta^{in}) = \delta^{out}$ is an affine subset. Since $(v_i + \delta^{in}) + \delta^{in} = v_i$, then $v_i + \delta^{in} \in \{v_1, v_2, ..., v_l\}$, implying $\{v_1, v_2, ..., v_l\} = \{v_1 + \delta^{in}, v_2 + \delta^{in}, ..., v_l + \delta^{in}\}$. Thus, both coordinates of the ordered pairs give the same subset, and we denote it by $A(\delta^{in}, \delta^{out})$ (note that $|A(\delta^{in}, \delta^{out})| = DDT(\delta^{in}, \delta^{out})$). On the other hand, since the algebraic degree of the inverse Sbox is 3, which is reduced to 2 (rather than 1) after differentiation, the output values that satisfy an input and an output difference do not necessarily form an affine subset.

The Third Property For any non-zero 5-bit output difference δ^{out} to a Keccak Sbox, the set of possible input differences, $\{\delta^{in} | DDT(\delta^{in}, \delta^{out}) > 0\}$, contains at least 5 (and up to 17) 2-dimensional affine subspaces. These affine subspaces can be easily pre-computed using the DDT, for each one of the 31 possible non-zero output differences. However, we note that there is no output difference for which the set of possible input differences contains an affine subspace of dimension 3 or higher.

4.2 Formulating the Problem

Given Δ_T, an arbitrary message pair (M^1, M^2) in which $|M^1| = |M^2| = r - 8$ is a solution to our problem if $R(\overline{M}^1) + R(\overline{M}^2) = \Delta_T$. This can be formulated using two constraints on the 1600-bit words $(\overline{M}_1, \overline{M}_2)$:

1. The $2n+8$ MSBs of $\overline{M}^1$ and $\overline{M}^2$ are equal to $p||0^{2n}$, where p denotes the 8-bit pad 10000001.
2. $R(\overline{M}^1) + R(\overline{M}^2) = \Delta_T$ (where R is the permutation round of Keccak).

We can easily formulate the first constraint using linear equations on the bits of $\overline{M}_1$ and $\overline{M}_2$. Since Keccak's Sbox has an algebraic degree of 2 over $GF(2)$, we can formulate the second constraint as a system of quadratic equations on these bits. Standard heuristic techniques for solving such systems include using the available degrees of freedom to fix some message values (or values before the first Sbox layer) in order to linearize the system. However, these techniques require many more than the available number of degrees of freedom. For example, in order to get linear equations after one round of Keccak's permutation, we can fix 3 out of the 5 bits entering an Sbox (after the first linear layer), such that there are no two consecutive unknown input bits entering the Sbox. Using this technique reduces the single quadratic term in the symbolic form of each of the Sbox'es output bits to a linear term. However, this requires fixing $320 \cdot 3 = 960$ bits per massage, and $2 \cdot 960 = 1920$ bits in total, which is significantly more than the 704 available degrees of freedom for Keccak-224 (and clearly more than the available number of degrees of freedom for the other Keccak versions). Consequently, we have to repeat the linearization procedure a huge number of times, with different fixed values, in order to find a solution.

A Two-Phase Algorithm Although we expect our quadratic system to have many solutions, solving all the equations at once seems difficult. Thus, we split the problem into easier tasks by exploiting the low algebraic degree of Keccak's Sbox to a greater extent than in the standard techniques: as described in the second property of Section 4.1 given an input difference and an output difference to an Sbox, all the pairs of input values that satisfy them form an affine subset.[2] This suggests an algorithm with two phases, where in the first phase (called the *difference phase*) we find an input difference to all the Sboxes, and in the second phase (called the *value phase*) we obtain the actual values of the message pairs that lead to the target difference.

Using this two-phase approach, the ordered pairs produced by our algorithm satisfy two additional properties: the 1600-bit *input difference* of the initial states is fixed to some 1600-bit value Δ_I (i.e. $\overline{M}_i^1 + \overline{M}_i^2 = \Delta_I \ \forall i \in \{1, 2, ..., k\}$), and the set composed of all the initial states defined by the first message in each ordered pair (i.e. $\bigcup\{\overline{M}_i^1\} \ \forall i \in \{1, 2, ..., k\}$), forms an affine subset. The algorithm outputs the ordered pairs as the fixed 1600-bit input difference Δ_I, and some basis for the affine subset $\bigcup\{\overline{M}_i^1\} \ \forall i \in \{1, 2, ..., k\}$. We note that the large number of degrees of freedom allows us to restrict the set of solutions (i.e. the set of message pairs that satisfy the target difference) to a smaller subset (but still large enough for our purposes) that can be found relatively easily. In particular, the algorithm considers only message pairs with a fixed difference Δ_I, for which all the solutions can be found by solving linear equations.

The two constraints above, which define our quadratic equation system, are broken into two sets of constraints, since we have to simultaneously enforce two *difference constraints* (given as constraints on the 1600-bit word Δ_I):

Difference Constraint 1 *The $2n + 8$ most significant bits (MSBs) of Δ_I are equal to zero.*
Difference Constraint 2 *$L(\Delta_I)$ is a valid input difference to the Sbox layer, i.e. there exists some 1600-bit word W such that $\chi(W) + \chi(W + L(\Delta_I)) = \Delta_T$ (note that since L is a linear function, $L(\Delta_I)$ is well-defined).*

The first difference constraint simply equates bits of the input difference Δ_I to zero (456 bits for Keccak-224 and 520 bits for Keccak-256), while the second difference constraint assigns to every 5 bits of $L(\Delta_I)$ that enter an Sbox, several possible values which are not related by simple affine equations.

In the second phase, we enforce additional *value constraints* (given on the 1600-bit word $\overline{M}^1$):

Value Constraint 1 *The $2n + 8$ MSBs of $\overline{M}^1$ are equal to $p||0^{2n}$, where p denotes the 8-bit pad* 10000001*.*
Value Constraint 2 *$R(\overline{M}^1) + R(\overline{M}^1 + \Delta_I) = \Delta_T$.*

Note that the first difference constraint and the first value constraint on each $\overline{M}_i^1$ also ensure that the same value constraint holds for $\overline{M}_i^2$ (i.e., the $2n + 8$ MSBs of $\overline{M}_i^2$ are equal to $p||0^{2n}$).

[2] Similar observations were used in [7] to suggest that when $DDT(\delta^{in}, \delta^{out}) = 2$ or 4, the same holds. In the specific case of Keccak, we also use 3-dimensional affine subsets of pairs that satisfy the Sbox difference transition $(\delta^{in}, \delta^{out})$, for which $DDT(\delta^{in}, \delta^{out}) = 8$.

Given a single 1600-bit Sbox layer input difference, the second property of Section 4.1 implied that enforcing the two value constraints simply reduces to solving a union of two sets of linear equations. On the other hand, it is not clear how to simultaneously enforce both of the difference constraints, since given an output difference to an Sbox δ^{out}, all the possible input differences δ^{in} such that $DDT(\delta^{in}, \delta^{out}) > 0$, are not related by simple affine relations.

4.3 The Difference Phase

Unsuccessful Attempts to Enforce the Difference Constraints We can try to enforce both difference constraints by assigning the undetermined $1600 - 2n - 8$ bits of Δ_I, in such a way that the second difference constraint will hold. This usually involves iteratively constructing an assignment for Δ_I, by guessing several undetermined bits at a time, and filtering the guesses by verifying the second difference constraint. However, this is likely to have a very large time complexity, since L diffuses the bits of Δ_I in a way that forces us to guess many bits before we can start filtering the guesses. Moreover, for any Δ_T, the fraction of input differences satisfying the first difference constraint that also satisfy the second difference constraint is very small. Thus, most of the computational effort turns out to be useless, since the guesses are likely to be discarded at later stages of the algorithm. Another approach is to guess $L(\Delta_I)$ by iteratively guessing the 5-bit Sbox input differences, and filtering the guesses by verifying the first difference constraint. For similar reasons, this approach is likely to have a very large time complexity.

A Better Approach Both of these approaches are very strict, since each guess made by the algorithm commits to a specific value for some of the bits of Δ_I, or $L(\Delta_I)$, and restricts the solution space significantly. Thus, we use the third property of Section 4.1, which gives us more flexibility, and significantly reduces the time complexity: given any non-zero 5-bit output difference to a Keccak Sbox, the set of possible input differences contains at least five 2-dimensional affine subspaces. Consequently, in order to enforce the second difference constraint, for each Sbox with a non-zero output difference (i.e., an active Sbox), we choose one of the affine subsets (which contains 4 potential values for the 5 Sbox input bits of $L(\Delta_I)$), instead of choosing specific values for these bits. This enables us to maintain an affine subspace of potential values for $L(\Delta_I)$, starting with the full 1600-dimensional space, and iteratively reducing its dimension by adding affine equations in order to enforce the second difference constraint for each Sbox. In addition to these affine equations that we add per active Sbox, we also have to add the linear equations for the non-active Sboxes (which equate their 5 input difference bits to zero), and the additional $2n + 8$ linear equations that enforce the first difference constraint. All of these equations are added to a linear system of equations that we denote by E_Δ.

Since the $2n + 8$ equations that enforce the first difference constraint do not depend on the target difference, we add them to E_Δ before we iterate the Sboxes. While iterating over the active Sboxes, we add equations on $L(\Delta_I)$ in order to enforce the second difference constraint and hope that for each Sbox, we can add equations such that E_Δ is consistent. Note that the equations in E_Δ in each stage of the algorithm depend on the order in which we consider the active Sboxes, and on the order in which we consider the possible affine subsets of input differences for each Sbox. Thus, if we reach an Sbox for which we cannot add equations in order to enforce the second constraint (while maintaining the consistency of E_Δ), it does not imply that it is impossible to satisfy the difference constraints. In this case, we can simply change the order in which we consider the active Sboxes, or the order in which we consider the affine subsets for each Sbox, and try again. Since we cannot predict in advance the orderings that give the best result, we choose them heuristically, as described in Appendix A.

4.4 The Value Phase

In case the difference phase procedure described above succeeds, it actually outputs an affine subspace of candidate input differences, rather than a single value for Δ_I. Next, we can commit to a specific value for Δ_I and run the value phase, hoping that the set of all linear equations defined by the value constraints has a solution. Namely, we allocate another system of equations, which we denote by E_M, and add the equations on $\overline{M}^1$ that enforce the first value constraint. We then add the additional linear equations that enforce the second value constraints for all the Sboxes, and output the solution to the system, if it exists. However, once again, this approach is too strict, and may force us to repeat the value phase a huge number of times with different values for Δ_I, until we find a solution. Thus, we do not choose a single value for Δ_I in advance. Instead, we reduce the linear subset of candidates for Δ_I gradually by fixing the input difference to each one of the active Sboxes, until a single value for Δ_I remains. Thus, we continue to maintain E_Δ throughout the value phase, and iteratively add the additional 2 equations which are required to uniquely specify a 5-bit input difference for each active Sbox, among the 2-dimensional affine subsets chosen in the difference phase. Once we fix the input difference to an Sbox, we immediately obtain linear equations on $\overline{M}^1$, and we can check their consistency with the current equations in E_M. In case the equations in E_M are not consistent for a certain Sbox, we can try to choose another input difference for it. This gives different equations on $\overline{M}^1$, which may be consistent and allow us to continue the process.

Similarly to the difference phase, the equations in E_M in each stage of the algorithm depend on the order in which we consider the active Sboxes, and on the order in which we consider the possible input differences for each Sbox. Thus, once again, if at some stage of the value phase we cannot add any consistent equations to E_M, we can change one of these orderings and try again, hoping to obtain a valid solution.

We stress again that both phases of the algorithm are not guaranteed to succeed. The success of each phase depends on the target difference, and on orderings which are chosen heuristically. As a result, we may have to iterate both phases of the algorithm an undetermined number of times with modified orderings, hoping to obtain better results.

5 Application of the Target Difference Algorithm to Round-Reduced Keccak

Since we would like to use the target difference algorithm in order to find collisions and near-collisions in Keccak, it is crucial to verify the algorithm's success on target differences which lead to these results. Thus, before we run the algorithm, we have to find such high probability differential characteristics, and to obtain the target differences which are likely to be the most successful inputs to the algorithm. As described in the introduction, once we find a high probability differential characteristic with a low Hamming weight starting state difference, we extend it backwards to obtain the target difference (while maintaining its high probability). We then use the target difference algorithm to link the extended characteristic backwards to the initial state of Keccak's permutation, with an additional round. Thus, any low Hamming weight characteristic for r rounds of Keccak's permutation can be used to obtain results on a round-reduced version of Keccak with $r + 2$ round. Specifically, in this section we demonstrate how we use 2-round characteristics in order to find collisions for 4 rounds of Keccak-224 and Keccak-256, and how to use 3-round characteristics in order to find near-collisions for 5 rounds of these Keccak versions.

5.1 Searching for Differential Characteristics

We reuse the notion of a *column parity kernel* or *CP-kernel* that was defined in the Keccak submission document [4]: a 1600-bit state difference is in the CP-kernel if all of its columns have even parity. It is easy to see that such state differences are fixed points of the function θ, which does not increase their Hamming weight. Since ρ and π just reorder the bits of the state, the application of L to a CP-kernel does not change its total Hamming weight. In addition, there is a high probability that such low Hamming weight differential states are fixed points of χ. Thus, when we start a differential characteristic from a low Hamming weight CP-kernel, we can extend it beyond the Sbox layer, χ, to one additional round of the Keccak permutation, with relatively high probability and without increasing its Hamming weight. However, extending such a characteristic to more rounds in a similar way is more challenging, since we have to ensure that the state difference before the application of θ remains in the CP-kernel at the beginning of each round.

Using Previous Results In [9] and [14], the authors propose algorithms for constructing low Hamming weight differential characteristics for Keccak. Both of these algorithms successfully find differential characteristics that stay in the CP-kernel for 2 rounds (named *double kernel trails* in [14]), some of which lead to collisions on the n-bit extract taken from the final state after 2 rounds, with high probability. However, when trying to extend each one of these characteristics by another round, the state difference is no longer in the CP-kernel and thus its Hamming weight increases significantly (from less than 10 to a few dozen bits). Nevertheless, the Hamming weight of the characteristics is still relatively low, and they can lead with reasonably high probability to near-collisions on the n output bits extracted. Beyond 3 rounds, the Hamming weight of the characteristics becomes very high (more than 100), and it seems unlikely that they can be extended to give collisions or near-collisions with reasonable probability. The currently known double kernel differential trails only extend forward to at most three rounds with reasonably high probability (higher than 2^{-100}). Finding new high probability differential characteristics, starting from a low Hamming weight state difference and extending forwards more than 3 rounds, remains a challenging task, which we do not deal with in this paper.

Our attacks on round-reduced Keccak make use of the type of differential characteristics that were found in [9] and [14], namely low Hamming weight characteristics that stay in the CP-kernel for 2 rounds. The double kernel trails with the highest probability have Hamming weight of 6 at the input to the initial round, and due to their low hamming weight, we could easily find all these characteristics within a minute on a standard PC. There are 571 such characteristics out of which, 128 can give collisions for Keccak-224 and 64 can give collisions for Keccak-256. However, when trying to extend the characteristics by an additional round, we were not able to find any characteristic that gives collisions for Keccak-224 (or Keccak-256) with reasonable probability. Thus, our best 3-round characteristics lead only to near-collisions, rather than collisions. The characteristics that give the near-collisions with the smallest difference Hamming weight for Keccak-224 and Keccak-256 are, again, double kernel trails with 6 non-zero input bits. The best 3-round characteristics for Keccak-224 lead to near-collisions with a difference Hamming weight of 5, and for Keccak-256, the best 3-round characteristics leads to a near-collision difference Hamming weight of 8. Examples of these characteristics are found in Appendix C.

Extending the characteristics Backwards Since the characteristics that we use start with a low Hamming weight state difference, we can extend them backwards by one round without reducing their probability significantly (as done in [9]): we take this low Hamming weight initial state difference, and choose a valid state difference input to the previous Sbox layer which could produce it. We then apply L^{-1}, and obtain a new initial state difference for the extended characteristic, which serves as a target difference for our new algorithm. Note that the target difference is not in the CP-kernel (otherwise, we would have found a low Hamming weight differential characteristic that stays in the CP-kernel for 3 rounds). Thus, when we apply L^{-1} to the state difference that enters the Sbox layer, we usually obtain a roughly balanced target difference, with only a few non-active Sboxes. This is significant to the success of the target difference algorithm, which strongly depends on the number of active Sboxes in the target difference.[3] In case the target difference obtained from a characteristic has too many non-active Sboxes, we can try to select another target difference for the characteristic, by tweaking the state difference input to the second Sbox layer.

Assuming that the algorithm succeeds and we obtain a sufficiently large linear subspace of message pairs (such that it contains at least one pair whose difference evolve according to the characteristic), we can find collisions for 4 rounds and near-collisions for 5 rounds of Keccak-224 and Keccak-256. For example, if we have an extended characteristic which can give collisions for 3 round of Keccak-256 with probability 2^{-24}, we need a linear subspace which contains at least 2^{24} message pairs in order to find a collision on 4-round Keccak-256 with high probability.

5.2 Applying the Target Difference Algorithm to the Selected Differential Characteristics

We tested our target difference algorithm using a standard PC, on dozens of double-kernel trails with Hamming weight of 6. For each one of them, after tweaking the state difference input to the second Sbox layer at most once, we could easily compute a target difference where all of the 320 Sboxes are active. We then ran the target difference algorithm on each one of these targets. For both Keccak-224 and Keccak-256, the target difference algorithm eventually succeeded: the basic procedure of the difference phase always succeeded within the first two attempts (after changing the order in which we considered the Sboxes), while the value phase was more problematic, and we had to iterate its basic procedure dozens to thousands of times in order to find a good ordering of the Sboxes and obtain results. For Keccak-224, the algorithm typically returned an affine subspace of message pairs with a dimension of about 100 within one minute. For Keccak-256, the dimension of the affine subspaces of message pairs returned was typically between 35 and 50, which is smaller compared to the typical result size for Keccak-224 (as expected since we have fewer degrees of freedom). In addition, unlike Keccak-224, for Keccak-256 we had to rerun the algorithm (starting from the difference phase) a few times, when the value phase did not seem to succeed for the choice of candidate input difference subset. Hence, the running time of the algorithm was typically longer – between 3 and 5 minutes, which is completely practical.

5.3 Obtaining Actual Collisions and Near-Collisions for Round-Reduced Keccak-224 and Keccak-256

Obtaining Collisions After successfully running the target difference algorithm, we were able to find collisions for 4-round Keccak for each tested double-kernel trail with Hamming weight of 6 (which leads to a collision). Since the probability of each one of these differential characteristics is greater than 2^{-30}, the probability that a random pair which satisfies its corresponding target difference leads to a collision, is greater than 2^{-30}. Thus, we expect to find collisions quickly for both Keccak-224 and Keccak-256, once the target difference algorithm returns a set of more than 2^{30} message pairs. However, even though the subsets we used contained more than 2^{30}

[3] As demonstrated in Section 4.1, we expect a large number of non-active Sboxes to foil the target difference algorithm. This should be contrasted to differential attacks, where the attacker searches for differential characteristics with many non-active Sboxes, which ensure that the differential transitions occur with high probability.

message pairs, we were not able to find collisions within several of these subsets for Keccak-224, and for many of the subsets for Keccak-256. As a result, we had to rerun the target difference algorithm and obtain additional sets of message pairs, until a collision was found. Thus, the entire process of finding a collision typically takes about 2–3 minutes for Keccak-224, and 15–30 minutes for Keccak-256. The reason that there were no 4-round collisions within many of the subsets of message pairs, is the incomplete diffusion of the Keccak permutation within the first two rounds. Since our subsets of message pairs are relatively small (especially for Keccak-256), and the values of all the message pairs within a subset are closely related, some close relations between a small number of bits still hold before the Sbox layer of the second round (e.g., the value of a certain bit is always 0, or the XOR of two bits is always 1). Some of these non-random relations make the desired difference transition into the second Sbox layer impossible, for all the message pairs within a subset. We note that we were still able to find collisions rather quickly, since it is easy to detect the cases where the difference transition within the second Sbox layer is impossible[4](which allowed us to immediately rerun the target difference algorithm). In addition, when this difference transition is possible, we were always able to find collisions within the subset. Two concrete example of colliding message pairs for Keccak-224 and Keccak-256 are given in Appendix D.

Obtaining Near-Collisions In order to obtain near-collisions on 5-round Keccak-224 and Keccak-256, we again start by choosing suitable differential characteristics. Out of all the characteristics that we searched, we chose the differential characteristics described in Appendix C, which lead to near-collisions of minimal Hamming weight for the two versions of Keccak. The results of the target difference algorithm when applied to targets chosen according to these characteristics, were similar to the results described in Section 5.2. However, compared to the probability of the characteristics leading to a collision, the probability of these longer characteristics is lower: the probability of the characteristics are 2^{-57} and 2^{-59} for Keccak-224 and Keccak-256, respectively. Thus, obtaining message pairs whose differences propagate according to these characteristics, and lead to 5-round near-collisions, is more difficult than obtaining collisions for 4 rounds of Keccak-224 and Keccak-256. However, for each such main characteristic, there are several secondary characteristics which diverge from the main one in final two rounds and give similar results. Thus, the probabilities of finding near collisions with a small Hamming distance for 5 rounds of Keccak-224 and Keccak-256, are higher than the ones stated above. In addition, by using some simple message modification techniques within the subsets returned by the target difference algorithm, we were able to improve these probabilities further. Thus, for Keccak-224, we obtained near-collisions with a Hamming distance of 5, which is the same as the output Hamming distance of the main characteristic that we used. For Keccak-256, the main characteristic that we used has an output Hamming distance of 8, but we were only able to find message pairs which give a near-collision with a slightly higher Hamming distance of 10. All of these near-collisions were found within a few days on a standard PC. Examples of such near-collisions are given in Appendix D.

6 Conclusions and Future Work

In this paper, we presented practical collision and near-collision attacks on reduced-round variants of Keccak-224 and Keccak-256. Our attacks are based on a novel target difference algorithm,

[4] In order to detect that the difference transition within the second Sbox layer is impossible for all the pairs in our subset, we try several arbitrary pairs in the subset, and observe if at least one has the desired difference after two rounds. Since we only need to check one Sbox layer transition, we expect that if this transition is indeed possible, we will find a corresponding message pair very quickly. Otherwise, we have to find a different set of message pairs by running the difference phase again.

which is used to link high probability differential characteristics for the Keccak internal permutation to legal initial states of the hash function. Consequently, we were able to significantly improve the best known previous results on Keccak, by doubling (from 2 to 4) the number of rounds for which collisions can be found in a practical amount of time.

Our target difference algorithm is clearly limited by the number of available degrees of freedom, and it seems difficult to extend it to reach target differences spanning 2 or more rounds of the Keccak permutation. However, it seems very likely that the algorithm will be useful in the future if longer high probability differential characteristics are found for the Keccak permutation.

References

1. Jean-Philippe Aumasson and Willi Meier. Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. NIST mailing list, 2009.
2. Daniel J. Bernstein. Second preimages for 6 (7? (8??)) rounds of keccak? NIST mailing list, 2010.
3. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions. Presented at the ECRYPT Hash Workshop, 2007.
4. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak SHA-3 submission. Submission to NIST (Round 3), 2011.
5. Christina Boura and Anne Canteaut. Zero-Sum Distinguishers for Iterated Permutations and Application to Keccak-*f* and Hamsi-256. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2010.
6. Christina Boura, Anne Canteaut, and Christophe De Cannière. Higher-order differential properties of keccak and luffa. Cryptology ePrint Archive, Report 2010/589, 2010. http://eprint.iacr.org/.
7. Joan Daemen and Vincent Rijemn. Plateau Characteristics. *IET Information Security*, 1(1):11–17, 2007.
8. Ming Duan and Xuajia Lai. Improved zero-sum distinguisher for full round Keccak-f permutation. Cryptology ePrint Archive, Report 2011/023, 2011.
9. Alexandre Duc, Jian Guo, Thomas Peyrin, and Lei Wei. Unaligned rebound attack - application to keccak. Cryptology ePrint Archive, Report 2011/420, 2011.
10. Dmitry Khovratovich. Cryptanalysis of Hash Functions with Structures. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 108–125. Springer, 2009.
11. Dmitry Khovratovich, Alex Biryukov, and Ivica Nikolic. Speeding up Collision Search for Byte-Oriented Hash Functions. In Marc Fischlin, editor, *CT-RSA*, volume 5473 of *Lecture Notes in Computer Science*, pages 164–181. Springer, 2009.
12. Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In Orr Dunkelman, editor, *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 260–276. Springer, 2009.
13. Pawel Morawiecki and Marian Srebrny. A SAT-based preimage analysis of reduced KECCAK hash functions. Cryptology ePrint Archive, Report 2010/285, 2010.
14. María Naya-Plasencia, Andrea Röck, and Willi Meier. Practical Analysis of Reduced-Round Keccak. In Daniel J. Bernstein and Sanjit Chatterjee, editors, *Progress in Cryptology - INDOCRYPT 2011*, volume 7107 of *LNCS*. Springer, Heidelberg, 2011.

A Details of the Target Difference Algorithm

In this section, we give the full details of the target difference algorithm. The general structures of the difference phase and the value phase of the algorithm are similar: both include a main procedure, which iterates a basic procedure, until it succeeds.

A.1 Calculating a Set of Candidate Input Differences

We give the details of the difference phase, which finds a set of candidate input differences $(\Delta_{I_1}, \Delta_{I_2}, ...\Delta_{I_i}, ...)$, given a target difference Δ_T. To find such a set, we have to simultaneously enforce the two difference constraints. As described in Section 4, we maintain a linear system of equations, which we denote by E_Δ, whose solution basis spans some affine subspace of potential values for $L(\mathit{\Delta}_I)$. We can easily obtain a basis for the set of potential values for $\mathit{\Delta}_I$ by multiplying each element of the basis for $L(\mathit{\Delta}_I)$ by the matrix which defines L^{-1}.

We assume that $\mathit{\Delta}_T$ contains t active Sboxes and $320 - t$ non-active Sboxes whose input and output differences are zero. Before executing the algorithm, we have to initialize a data structure that is used repeatedly in both phases of the target difference algorithm. This data structure stores the subsets of input differences for each active Sbox (which are precalculated per possible non-zero Sbox output difference, as described in the third property in Section 4.1). In addition, it specifies the order in which we consider the specific subsets for each one of the t active Sboxes, and the order in which we consider the active Sboxes themselves. As described in Section 4, these orderings have a significant effect on the success and running time of the algorithm.

The input difference subsets, for each of the t active Sboxes, are stored in a sorted *input difference subset list*, or *IDSL*. The heuristic algorithm we use to pick the sorting order of each IDSL is specified in Appendix A.3, but since it is irrelevant in this section, we assume some arbitrary ordering of each one of the IDSLs. In addition, each one of the IDSLs contains a pointer to an input difference subset, initialized to point at the first element in the list. The IDSLs are stored in the main *input difference subset data structure*, or *IDSD*. The IDSD contains t entries (one entry per active Sbox), sorted according to an IDSD order (which may differ from the natural order of the Sboxes). The initial IDSD order is chosen randomly, and is updated during both phases of the algorithm.

The Basic Procedure The steps of the basic procedure of the difference phase are given below:

1. Initialize an empty linear equation system $E_{\mathit{\Delta}}$ with 1600 variables for the unknown bits of $L(\mathit{\Delta}_I)$.
2. By inverting L, compute the coefficients of the $2n+8$ linear equations that equate the $2n+8$ MSBs of $\mathit{\Delta}_I$ to zero, and add the equations to $E_{\mathit{\Delta}}$.
3. For each of the $320 - t$ non-active SBoxes, add to $E_{\mathit{\Delta}}$ the 5 equations which equate the corresponding bits to zero.
4. Check if $E_{\mathit{\Delta}}$ is consistent, and if not, output "Fail".
5. Iterate over the t active SBoxes according to the IDSD order, and for each one of them:
   (a) Obtain the current 2-dimensional subset from the Sbox IDSL according to the pointer, and obtain the $5 - 2 = 3$ affine equations that define this subset.
   (b) If $E_{\mathit{\Delta}}$ is consistent after adding the 3 equations, add the equations and continue to the next Sbox. Otherwise, continue to the next subset in the Sbox IDSL, by incrementing the pointer and going to step a. If the end of the IDSL is reached, output "No Solution".
6. Output $E_{\mathit{\Delta}}$.

Our initial linear equation system contains $2n+8$ linear equations. Since each 2-dimensional subset of an active Sbox contributes 3 affine equations, the active Sboxes contribute a total of $3t$ equations. The non-active Sboxes contribute $5(320 - t)$ linear equations, since each non-active Sbox contributes 5 equations. Altogether, we have $2n + 8 + 3t + 5(320 - t) = 1600 + 8 + 2n - 2t$ linear equations in $E_{\mathit{\Delta}}$. Thus, for t large enough, if the algorithm succeeds, the solution subset is of dimension at least $1600 - (1600 + 8 + 2n - 2t) = 2t - 2n - 8$ (for example, for $t = 310$, $n = 224$ we get a solution subset of dimension at least $620 - 448 - 8 = 164$). If all the coefficients of the equations were chosen uniformly at random, then for $t > n+4$ we expect no dependencies among the equations, and thus the dimension of the solution subset is exactly $2t - 2n - 8$. However, our equations are clearly not random, and thus we can only verify the procedure's success experimentally.

Note that the equations added in step 2 are independent, since L is invertible (and thus all the 1600 linear equations that define L^{-1} are independent) and equating each bit to zero reduces the remaining dimension by 1. In addition, the equations of all the Sboxes are independent, since each Sbox gives equations on a distinct set of 5 variables. However, there is no guarantee that the combined equation set is independent, or consistent. In addition, note that dependencies among the equations may foil the procedure in case they are inconsistent. On the other hand, in case the dependent equations are consistent, they do not decrease the dimension of the solution subset, which gives us more flexibility in the value phase.

The Main Procedure In case the basic procedure outputs "Fail", then clearly there are no 1600-bit valid state pairs that satisfy the target difference, and the algorithm aborts. If the procedure outputs "No Solution", then our current choice of 2-dimensional subsets gives inconsistent equations, and we change our choice, using a heuristic algorithm, hoping to obtain better results. The main idea behind the heuristic is to change the order in which we consider the Sboxes, such that the "problematic" groups of Sboxes (which tend to produce the inconsistent equations) are pushed to the front of the IDSD order. When considering the first few Sboxes, the number of equations in E_{Δ} is relatively small, and we have more options to choose consistent equations for these Sboxes. Thus, when the "problematic" Sbox groups are considered first, it seems more likely that the basic procedure will succeed. The main procedure of the difference phase is given below, where $T1$ is some fixed constant threshold:

1. Initialize a counter to 0. Initialize the IDSD by resetting all the IDSL pointers to the beginning of the lists, and randomizing the IDSD Sbox order.
2. Execute the basic procedure. If the procedure succeeds, output the current E_{Δ}. If the procedure outputs "Fail", abort. Otherwise (i.e., when the procedure outputs "No Solution"), go to step 3.
3. Increment the counter. If the counter's value is equal to $T1$, go to step 1.
4. Reset the pointer of the failed Sbox IDSL to its value before the last basic procedure.
5. Change the IDSD order by bringing the failed Sbox to the front (and pushing the rest of the Sboxes one position backwards).
6. Go to step 2.

Since there is nothing that ensures that the main procedure of the difference phase halts, it may run a very long time without finding a solution (either because there is no solution, or it is extremely difficult to find one). The procedure may even enter a loop by setting the state of the IDSD in steps 4 and 5 to a previously considered state. Thus, after $T1$ executions of the basic procedure which output "No Solution", we start over by re-randomizing the IDSD order and resetting all the IDSL pointers.

A.2 Choosing Δ_I and the Set of Message Pairs

After we successfully enforce the difference constraints on the input difference Δ_I in the difference phase and obtain a set of candidate input differences, we are ready to execute the value phase. As described in Section 4, in this phase we iteratively narrow down the dimension of the set of candidate input differences, until only one candidate for Δ_I remains. While doing so, we simultaneously narrow down the set of pairs of possible initial states: we start with a set containing all the possible 1600-bit initial state pairs with the $2n+8$ MSBs equal to $p||0^{2n}$, and finally obtain the set $\{(\overline{M}_1^1, \overline{M}_1^2), (\overline{M}_2^1, \overline{M}_2^2), ..., (\overline{M}_k^1, \overline{M}_k^2)\}$, where all pairs have the difference Δ_I and satisfy the target difference (the actual message can be easily obtained from the corresponding 1600-bit value, by removing these fixed $2n+8$ MSBs).

Similarly to the procedure that we use for the difference phase, we continue to maintain the set of linear equations which define the set of possible input differences, E_{Δ}. We iterate over the active Sboxes according to the current IDSD order, and for each Sbox we choose a 5-bit input difference that is consistent with the current equations in E_{Δ}. We then narrow the set of possible input differences down by adding the additional $5-3=2$ equations that define this 5-bit input difference to E_{Δ}.

In addition to maintaining E_{Δ}, we maintain another system of linear equations, whose solution basis spans the set of possible values for $L(\overline{M}^1)$ (the 1600-bit state obtained by invoking L on the first message in each one of the ordered pairs). This set of equations is used to simultaneously enforce the two value constraints given in Section 4, and is denoted by E_M. Given E_M, we can easily obtain a basis for the set $\overline{M}_i^1$ $\forall i \in \{1,2,...,k\}$, by multiplying each element of the basis for $L(\overline{M}_i^1)$ by the matrix that defines L^{-1}.

Initially, we add to E_M all the $2n+8$ linear equations that enforce the first value constraint on the first initial state of the ordered pairs. We then enforce the second value constraint on each Sbox independently: for each Sbox with output difference δ^{out} (computed from Δ_T), once we determine its input difference δ^{in}, we also obtain the linear equations that restrict the values of the 5 variables of $L(\overline{M}^1)$ to the affine subset $A(\delta^{in}, \delta^{out})$. We then enforce the second value constraint on the Sbox by adding these linear equations to E_M. Note that in the difference phase, we ensured that in each stage of the value phase, for each Sbox with a fixed output difference, there exists some input difference whose equations can be added to E_{Δ}, while maintaining its consistency. In addition, we ensured that the affine subset of values that corresponds to these input and output differences is non-empty. However, we did not ensure that the equations that restrict the values of the variables of $L(\overline{M}^1)$ to this subset are consistent with the equations that are currently in E_M.

The Basic Procedure The steps of the basic procedure of the value phase are given below:

1. Initialize an empty linear equation system E_M with 1600 variables for the unknown bits of $L(\overline{M}^1)$.
2. By inverting L, compute the coefficients of the $2n+8$ linear equations that equate the $2n+8$ MSBs of $L(\overline{M}^1)$ to $p||0^{2n}$ (p is the 8-bit pad 10000001), and add the equations to E_M.
3. Iterate the t active SBoxes according to the IDSD order:
   (a) Using Δ_T, obtain δ^{out} for the Sbox.
   (b) Obtain all the Sbox input differences that are consistent with E_{Δ} (denoted by $\{\delta_i^{in}\}$), and sort them in descending order according to the size of the affine subset of values that satisfy the input-output difference (i.e., make sure that $DDT(\delta_i^{in}, \delta^{out}) \geq DDT(\delta_{i+1}^{in}, \delta^{out})$).
   (c) Iterate the sorted input differences obtained in the previous step, starting with δ_1^{in} (which gives the largest affine subset of values):
      i. Using the current input difference, δ_i^{in} obtain the linear equations that define the affine subset $A(\delta_i^{in}, \delta^{out})$.
      ii. If E_M is consistent with these linear equations, add the equations to E_M. In addition, add the additional equations on the input difference (that were not added in the difference phase), which are defined by δ_i^{in}, to E_{Δ}, and continue to the next Sbox in step 3.
      iii. Otherwise, continue to the next input difference in step c. If no more input differences remain, output "No Solution".
4. Output the fully determined 1600-bit value of Δ_I, in addition to the equation system that defines the message values, E_M.

The algorithm makes a heuristic choice in step c, by considering the input differences for each Sbox, starting with the one that gives the largest affine subset of values. The idea is to keep the number of equations in E_M as small as possible, in order to reduce the probability of failure. In addition, when we generate fewer independent equations in E_M at the end of the process, we get a larger set of pairs that satisfy the target difference.

The Main Procedure Similarly to the basic procedure of the difference phase, there is no guarantee that this procedure succeeds. Hence, we may need to repeat it several times with different choices of input differences for each Sbox, which result in different systems of equations, E_M. The steps of the main procedure of the value phase are given below, where $T2$ is some fixed constant threshold:

1. Initialize a counter to 0.
2. Set E_Δ to the equation system returned by the last successful execution of the difference phase.
3. Execute the basic procedure of the value phase. If the procedure succeeds, output Δ_I and E_M. Otherwise (i.e., the procedure outputs "No Solution"), continue to step 4.
4. Increment the counter. If the counter's value is equal to $T2$, output "No Solution".
5. Change the IDSD order by bringing the failed Sbox to the front (and pushing the rest of the Sboxes one position backwards).
6. Go to step 2.

Note the difference between the structures of the two phases of the algorithm: the only input to the difference phase is the value of Δ_T (which we assume to be fixed). Thus, unless it returns "Fail", there is no reason to stop its execution at any point. On the other hand, the value phase depends on the output of the difference phase, E_Δ. Since the particular choice of candidate set for Δ_I may foil the procedure, we terminate the value phase after the number of unsuccessful executions of its basic procedure reaches some threshold, $T2$. In this case, we restart the difference phase, hoping that another choice of candidate set for Δ_I will give better results.

A.3 Sorting the Input Difference Subset Lists

In this section, we give the details of how we sort the IDSLs, by specifying how to compare two input difference subsets in the list. Given a non-zero Sbox output difference, all the maximal possible input difference subsets are of dimension 2, and add 3 linear equations to E_Δ. This does not give an obvious reason to prefer one subset over another in the difference phase. However, the input differences within each input difference subset give affine subsets of different sizes: there are 20 specific non-zero Sbox output differences, δ^{out}, whose DDT entries $DDT(\delta^{in}, \delta^{out})$, have values of 2, 4 and 8. For the remaining 11 output differences, the non-zero DDT entries attain only the values 2 and 4. In the value phase, we prefer input differences that give large subsets of values. Thus, in the difference phase, we give precedence to input difference subsets containing such input differences: assume that Δ_T assigns a specific Sbox an output difference δ^{out}, and we want to compare two input difference subsets $\{\delta_1^{in}, \delta_2^{in}, \delta_3^{in}, \delta_4^{in}\}$ and $\{\delta_5^{in}, \delta_6^{in}, \delta_7^{in}, \delta_8^{in}\}$ such that $DDT(\delta_1^{in}, \delta^{out}) \geq DDT(\delta_2^{in}, \delta^{out}) \geq DDT(\delta_3^{in}, \delta^{out}) \geq DDT(\delta_4^{in}, \delta^{out}) > 0$ and $DDT(\delta_5^{in}, \delta^{out}) \geq DDT(\delta_6^{in}, \delta^{out}) \geq DDT(\delta_7^{in}, \delta^{out}) \geq DDT(\delta_8^{in}, \delta^{out}) > 0$. We first compare the sizes of the largest subsets of values, $DDT(\delta_1^{in}, \delta^{out})$ and $DDT(\delta_5^{in}, \delta^{out})$, and give precedence to the input difference subset for which the size is bigger. If the sizes are equal, we compare $DDT(\delta_2^{in}, \delta^{out})$ and $DDT(\delta_6^{in}, \delta^{out})$, and so forth.

B Appendix: An Alternative Value Phase

The value phase presented in section A.2 chooses Δ_I and the set of message pairs iteratively, which gives it more options, and reduces its running time compared to more restrictive algorithms which choose Δ_I in advance. However, it has two disadvantages: it restricts the messages to one block (by assuming a zero initial value for the inner part of the state), and more significantly, has no known bound on its expected running time. We can easily overcome the first disadvantage by choosing a prefix of $m-1$ arbitrary blocks, and calculating the inner part of the state obtained after running the permutation on the prefix. We then use the value of the inner state in order to apply the algorithm on an additional final block (which is the only one in which we use a difference between the two messages). The only change from the single-block version is that we initialize the values of the equations in step 2 of the basic procedure of the value phase according to the new inner state value.

Given that we choose a common prefix of several blocks for all message pairs, and run the target difference algorithm only on the last block, the difference in the inner states obtained after the prefix for each pair, is zero. Thus, the difference phase of the multi-block variant presented above is completely identical to the original single-block version. In fact, since the difference phase is completely independent of the inner state value, we can apply it and obtain suggestions for Δ_I before we even choose the prefix. This simple observation allows us to design a completely different value phase, whose expected complexity can be easily calculated using reasonable assumptions (and given that the difference phase succeeds).

In the alternative value phase, we first pick a fixed Δ_I from the input difference candidate set computed in the difference phase. This fixed input difference gives many linear equations that involve message bits which we can easily control, and thus we can satisfy a large portion of the equations regardless of the value of the inner part of the state. However, after fixing many message bits in order to satisfy equations, we may remain with some linear equations which only involve the inner part of the state, that cannot be directly controlled. Thus, we compute the inner part of the state for arbitrary message prefixes, hoping that one of them satisfies the remaining equations by chance. The complexity of the value phase is measured by the expected number of invocations of Keccak's internal permutation, which depends on the number of linear equations that cannot be satisfied by the message bits (i.e., involve only the variables of the inner part of the state). Thus, in the alternative value phase, we exploit the possibility to vary the inner part of the state, instead of varying the input difference Δ_I (as we do in the original value phase). The steps of the alternative value phase are given below:

1. Choose an input difference Δ_I from the candidate input differences outputted by the difference phase.
2. Initialize an empty linear equation system E_M with 1600 variables for the unknown bits of $L(\overline{M}^1)$.
3. For each Sbox:
   (a) Using Δ_I and Δ_T, compute the Sbox input and input differences $(\delta^{in}, \delta^{out})$.
   (b) Add the equations that define $A(\delta^{in}, \delta^{out})$ to E_M.
4. By inverting L, compute the coefficients of the 8 linear equations that define the padding. If these equations are consistent with E_M, add them and continue to the next step. Otherwise, go to step 1.
5. Compute the coefficients of the $2n$ linear equations that define the inner part of the state. Add them to E_M, by allocating additional $2n$ variables for their undetermined values (the vector of values of E_M contain $2n$ undetermined values).
6. Perform Gauss Elimination on E_M, and obtain the dependent equations, where the coefficients of the variables of $L(\overline{M}^1)$ are zero. These give linear equations on the $2n$ variables of the inner state.

7. Choose a common message prefix of $m-2$ blocks, each containing r bits, $M_1^*, M_2^*, ..., M_{m-2}^*$, where $m > 1$.
8. Compute the inner part of the state, C_{m-2}, after running the Keccak permutation on the prefix.
9. Iterate over the 2^r possible values of the message block with index $m-1$, M_{m-1}^*, in some order:
   (a) Execute the Keccak permutation on the message block M_{m-1}^* and the inner state C_{m-2}, and obtain the next inner state C_{m-1}.
   (b) Check if the equations obtained in step 6 hold for C_{m-1}. If they do, output Δ_I and E_M (without the trivial equations), in addition to the message prefix $M_1^*, M_2^*, ..., M_{m-1}^*$. Otherwise, go to the beginning of step 9 (choose the next value of M_{m-1}^*).

Note that in step 2, we add no equations to E_M for non-active Sboxes, and for active Sboxes we add at most 4 equations (since all the non-zero DDT entries that correspond to a non-zero output difference have a value of at least 2). Thus, after step 3, E_M contains a subset of dimension at least $1600 - (4 \cdot 320) = 320$. Since L^{-1} has a very fast diffusion, it is very unlikely that non of the initial states defined by this large subset attain the value of the 8-bit pad. Consequently, it is very unlikely that the procedure will fail even once on step 4. Thus, we ignore this possibility and assume that step 9 is the time complexity bottleneck of the procedure. Assuming that we get q dependent equations in step 6 of the attack, we obtain q linear equations on the inner state bits after $m-1$ blocks, C_{m-1}. Thus, assuming that the Keccak permutation behaves randomly, we expect to run it 2^q times in step 9, until all the q equations are satisfied. Note that we can calculate the expected running time of the algorithm only after selecting the exact value of Δ_I from the candidate set of input differences. Hence, it may be useful to run steps 1–6 several times, and choose the value of Δ_I which gives the best expected running time.

Although we cannot bound its expected complexity, in practice, the running time of the original value phase can be much lower than the running time of the alternative value phase. Thus, it is advisable to run the original value phase in case the expected running time of the alternative value phase is too high, and to calculate the expected running time of the alternative value phase in case the original value phase seems to be unsuccessful.

C Appendix: Differential Characteristics for Keccak

In this section, we give examples of 3-round differential characteristics, which lead to collisions on 4-round Keccak-224 and Keccak-256, and 4-round characteristics, which lead to near-collisions on 5-round Keccak-224 and Keccak-256.

The differential characteristics are given as a sequence of the starting state differences in each round. In all the presented characteristics, all the active Sboxes get an input difference with a Hamming weight of 1, and we assume that they produce the same differences as outputs (which occurs with probability 2^{-2}). In order to calculate the probability of the final transition, we only consider active Sboxes which effect the output bits (since we truncate the final state to obtain the hashed output). Each state difference is given as a matrix of 5×5 lanes of 64 bits, ordered from left to right, where each lane is given in hexadecimal using the little-endian format. The symbol '-' is used in order to denote a zero 4-bit difference value. For example, consider the second state difference in Characteristic 1: each of the first two lanes has a zero difference, and only the LSB of the third lane contains a non-zero difference.

```
|26978AF134CB835E|AF224C4D78366789|C4DAE35E2656F26B|357C4789AF3-6AF1|78D3526BC6A74C4D|
|26978AF134CB835E|AF224C4D78366789|C4DAE35E2656F26B|357C4789AF3-6AF1|78D3526BC6A74C4D|
|26978AF134CB835E|AF224C4D78366789|C4DAE35E2676F26B|357C4789AF3-6AF1|78D3526BC4A74C4D|
|26978AF134CB835E|AF224C4D78366789|C4DAE35E265EF26B|357C4789AF3-4AF1|78D3526BC6A74C4D|
|26978AF134CB835E|AF226C4D78366789|C4DAE35E2656F26B|35FC4789AF3-6AF1|78D3526BC6A74C4D|

|----------------|----------------|---------------1|-------4--------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|-------4--------|----8-----------|
|----------------|----------------|---------------1|----------------|----8-----------|

|----------------|----------------|----------------|--8-------------|2---------------|
|4---------------|----------------|----------------|----------------|2---------------|
|----------------|----------------|----------------|--8-------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|4---------------|----------------|----------------|----------------|----------------|

|----------------|----------------|----------------|----------------|----------------|
|-----------8----|-----------2----|----------------|----------------|----------------|
|----------------|----------------|-----------1----|----------------|-----------1----|
|---------1------|-------4--------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
```

The probability of each one of the first two transitions is 2^{-12}. The probability of the third transition is 1, since there are no active Sboxes which affect the output.

Characteristic 1: A 3-round characteristic leading to collisions on Keccak-224 and Keccak-256 with probability 2^{-24}

```
|BD135E2FA6BD1346|12D789A92F12D78F|D7E26BC344D7E224|E69AF134B5E69AD5|98BC4D6BF898BC58|
|BD135E2FA6BD1346|12D789A82F12D78F|D7E26BC344D7E264|E69AF134B5E69AD5|98BC4D6BF898BC58|
|BD135E2FA6BD1346|12D789AB2F12D78F|D7E26BC344D7E224|E69AF134B5E29AD5|98BC4D6BF898BC58|
|BD135E2FA6BD1346|12D789A92F12D78F|D7E26BC344D7E224|E69AF134B5E69AD5|98BC4D6BF898BC58|
|BD135E2FA6BD1346|12D789A92F12D78F|D7E26BC344D7E224|E29AF134B5E69AD5|98BC4D7BF898BC58|

|----------------|------------1---|----------------|----------------|---4------------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|------------1---|-----8----------|----------------|----------------|
|----------------|----------------|-----8----------|----------------|---4------------|
|----------------|----------------|----------------|----------------|----------------|

|----------------|----------------|----------4-----|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|------------2---|----------------|----------------|-4--------------|----------------|
|------------2---|----------------|----------4-----|-4--------------|----------------|
|----------------|----------------|----------------|----------------|----------------|

|----------------|----------------|----------------|------------8---|----------------|
|----------------|----------------|-----------1----|----------------|----------------|
|----------------|----------------|-----------8----|----------------|----------------|
|----------------|----------------|----------------|------2---------|----------------|
|----------1-----|----------------|----------------|--4-------------|----------------|

|----------------|2---------------|48-----4---2----|-4---12---------|---8---82-----1-|
|----98----------|-2---2-8-----4--|----------------|4---------------|--1----8----2---|
|-----------4----|2---1-----------|----12----------|4---2---2-8-----|------4---------|
|---1-4-----2---1|----------------|---------8------|--2-----8-----4-|----4--------9--|
|--2---1-----4---|-------------48-|1-4-----2---1---|----------------|-----------8----|
```

The characteristic leads to near-collisions with a Hamming distance of 5 for Keccak-224, and 8 for Keccak-256. The probability of each one of the first three transitions is 2^{-12}. The probability of the final transition is 2^{-21} for Keccak-224 and 2^{-23} for Keccak-256. The total probability is 2^{-57} for Keccak-224 and 2^{-59} for Keccak-256.

Characteristic 2: A 4-round characteristic leading to near-collisions on Keccak-224 and Keccak-256

D Appendix: Actual Collisions and Near-Collisions for Round-Reduced Keccak-224 and Keccak-256

We give several examples of collisions and near-collisions for Keccak-224 and Keccak-256. The padded messages and output values are given in blocks of 32-bits ordered from left to right, where each block is given in hexadecimal using the little-endian format.

```
M1=
C4F31C32 4C59AE6D 5D19F0F4 25C4E44B D8853032 8D5E12F2 BB6E6EE2 27C33B1E 6C091058 EB9002D5
3BA4A06F 4A0CC7F1 CCB55E51 8D0DD983 2B0A0843 9B21D3B0 53679075 526DDED2 48294844 6FF4ED2C
1ACE2C15 471C1DC7 D4098568 F1EBF639 EAF7B257 09FDAE87 688878E6 4875EB30 C9C32D80 3C9E6FCB
3C2ABCFA E6A4807B 2AB281B8 812332B3

M2=
A4D30EF7 80BB8F69 90C048DF EB7213B9 A6650424 3A65F63E 8C268881 B651B81F AADAFA3C EE2CA5C3
DB78EAC2 C8EAE779 442F9C35 3221E287 B3017A5A 90790712 1B1C8BDC E08B10A8 9A9D25CA 1BE7AAAC
4E2F3E9C 73717DAD 5566015A A198CFB9 5A1CA8C2 A0E3348A AE6C0BB1 3980F9E4 A4FA8B91 6E81A989
89A9BCAA E12BF1F1 30EF9595 812E8B45

Output=
61FB1891 F326B6D5 24DD94DF 73274984 05DA9A1D 3FD359B9 78B8393B F2E7990B
```

The messages were found using the target difference algorithm on the target difference given by Characteristic 1.

Collision 1: A collision for 4-round Keccak-256

```
M1=
FAC7AC69 2710BE04 8462C382 7ABF1BF9 D065CD30 DB64DEB8 1410CD30 C837D79B 22E446B7 31E9BD55
A6B2074C C86E32CC DE50A10A F7BAAA58 D96CBC88 9FBD75F6 5E0D735A D22AA663 16A574AA 7DB08692
558AB029 109B4D30 86CE5DCA 13A295C7 E7C9D94B 648794D2 62EE3CF8 69439337 8CAB9F15 AC7C3267
90F41CBE A20E6893 B4781F24 0BA37647 F29A67A0 81F628D0

M2=
CE5FBC81 47710FCC 462C92E0 48F5D2CF F92F6EC3 053E64E1 570780B9 F838EC54 8F74809F 66B4AC6F
70DD1843 BF34F0C5 5010C89A D8791148 D5CC073E 3239AEBC 7DF48D79 0EC7767B FB081604 AFA975B9
F8EFAE0F ED793473 479E931C F2F80A74 7192D08F 5EB5AB27 F1CAC04E F394232D 48656B2A A3471644
DB74E60A 05FB3B18 41DC27C3 8384BF53 32534C3E 811C00B5

Output=
826B10DC 0670E4E1 5B510CDA AB876AA8 B50557ED 267932FB AA4D38E8
```

The messages were found using the target difference algorithm on the target difference given by Characteristic 1.

Collision 2: A collision for 4-round Keccak-224

```
M1=
23296F07 44536A2B 16E1E363 09B509F9 639CA324 2B834133 61457E6D 9CF07597 6797B3D4 D1715ABA
6D8F4F9F 70D12920 E014BB37 54C32ADE 6117B3FB 30114566 4BA7D70A 00F055F0 71CFFDD4 B53F2563
E223A16D CC8DDAC4 7A59836B A53FBDDE 9FFEC45F 6A3476DC 7349BB92 56AF6E92 83866932 56624032
A936E410 60AC00FA 7E7C61F9 81583CAC

M2=
49D48DE2 9FA843CA 747C88E0 55425134 098CA5B3 C97DC68A B82BC6FD 0F864996 26B13425 D9F73B75
932CD02F FB12E036 47706100 9DEFFFE4 79435F9C DA727EF0 D9CA67C6 520BE2D1 19CF3933 3136D1A9
EEBEA9DD 150CA247 D494BF4A 492EFB26 11CB4C8D F5A10A05 69128FF4 B142742F CA59FE32 4FE68436
068F76AB 041A673E 461575B5 81AA2A54

Output1=
407D4466 FEA8B231 EC968181 DF902165 23C219FF 54571D70 2800F506 E818644B

Output2=
407D4466 FEA8B231 EC928181 FF902165 23C019FF 1C571D74 2800F516 E810656B
```

The messages were found using the target difference algorithm on the target difference given by Characteristic 2.

Near-Collision 1: A near collision with Hamming distance of 10 for 5-round Keccak-256

```
M1=
7DBC1AA9 62A70B2A C2BDAF81 4A4D484B 672F6FAF ED312C83 24BC1974 16946039 6B46EDF6 1AE571A0
EDA59D6E 7561766D 8F0B4C57 3C05C569 715B7DF9 53F81761 F6D43507 6E040495 9B5C08AB 5130BA66
22AF7F5C 755840F2 2E893F59 4C4A730F 8C4F425D 182F8D00 E98515ED E29251AD 853AB863 DC46A7AC
9FB7BB08 14767EFC 5345C7AF AA774E81 8A01A570 81D65453

M2=
5659C936 AF3BA787 809C1CE6 B287F81B E0A5E769 ECCEB8A0 72506F44 1A1B2A02 EE9AE408 D16A9358
BF03C4D6 90845C46 0C0441CC 8203EA8D 6D122EB1 9193F64F 55C3A6A7 47377ED6 D26E806F DEC2CBF8
A3B8949E A91B248D 420B13BC BEAB4166 EE348CF6 DB6CCD82 122F6BDA 2FBFA7E4 75E8A429 F397BC46
7E9DE824 6A973A22 371FD02D 92035083 267D1C7A 812EDE70

Output1=
85373497 97D871C2 FBD0A823 584C0ED4 C1B3BF4F BC408766 0584B08D

Output2=
85373497 97D871C2 FBD0A823 784C0ED4 E1B1BF5F BC408776 0584B08D
```

The messages were found using the target difference algorithm on the target difference given by Characteristic 2.

Near-Collision 2: A near collision with Hamming distance of 5 for 5-round Keccak-224