Another Look at Complementation Properties

Charles Bouillaguet[1], Orr Dunkelman[1,2],
Gaëtan Leurent[1], and Pierre-Alain Fouque[1]

[1] Département d'Informatique
École normale supérieure
45 Rue D'Ulm,
75320 Paris, France.
{charles.bouillaguet, gaetan.leurent, pierre-alain.fouque}@ens.fr
[2] Faculty of Mathematics and Computer Science
Weizmann Institute of Science
P.O. Box 26, Rehovot 76100, Israel
orr.dunkelman@weizmann.ac.il

Abstract. In this paper we present a collection of attacks based on generalisations of the complementation property of DES. We find symmetry relations in the key schedule and in the actual rounds, and we use these symmetries to build distinguishers for any number of rounds when the relation is deterministic. This can be seen as a generalisation of the complementation property of DES or of slide/related-key attacks, using different kinds of relations. We further explore these properties, and show that if the relations have easily found fixed points, a new kind of attacks can be applied.

Our main result is a self-similarity property on the SHA-3 candidate *Lesamnta*, which gives a very surprising result on its compression function. Despite the use of round constants which were designed to thwart any such attack, we show a distinguisher on the full compression function which needs only *one* query, and works for *any* number of rounds. We also show how to use this self-similarity property to find collisions on the full compression function of *Lesamnta* much faster than generic attacks. The main reason for this is the structure found in these round constants, which introduce an interesting and unexpected symmetry relation. This casts some doubt on the use of highly structured constants, as it is the case in many designs, including the AES and several SHA-3 candidates.

Our second main contribution is a new related-key differential attack on round-reduced versions of the XTEA block-cipher. We exploit the weakness of the key-schedule to suggest an iterative related-key differential. It can be used to recover the secret key faster than exhaustive search using two related keys on 37 rounds. We then isolate a big class of weak keys for which we can attack 51 rounds out of the cipher's 64 rounds.

We also apply our techniques to *ESSENCE*, $\mathcal{PURE}$, *SHAvite-3*, and Lucifer.

1 Introduction

In this paper with study the existence of simple relations that can go trough the rounds of a cipher with a very high probability. For example, in DES if all the key bits are flipped, then all the subkeys are flipped as well. Moreover, if we also negate the plaintext, then all the F functions receive the original input, and the ciphertext is also negated: $DES_{\overline{K}}(\overline{P}) = \overline{(DES_K(P))}$. This is known as the complementation property of DES.

A similar property is present in one round of AES [19]. If one rotates the columns of an AES state, this rotates the column of the state after SubBytes, ShiftRows, and MixColumns. A study of similarity relations of the AES round operations is done in [19]. The authors show that the rotations by 1, 2 or 3 columns are the only byte-permutation to commute with the AES round. The AES key-schedule is responsible for breaking those symmetry relations and one should be very careful when using the AES round in a new construction.

Another well-known example is based on related-key attacks [4,5,16] and slide attacks [8]. In the latter, two plaintexts such that one is the encryption of the other by one round are used. If this is the case, then the ciphertexts are also separated by one round of encryption. Hence, a slid pair (or a related-key plaintext pair) suggests two equations for the round function, which in many cases is sufficient to retrieve the secret key. We note that slide attacks were also adapted for hash functions, where the slide property is used for several attack scenarios [11].

In this paper we show new kinds of self-similarity properties in block ciphers and hash functions. The new ideas generalize the previous attacks, by treating a wider set of relations. Moreover, some of the similarity relations we use have fixed points, which is not the case for the complementation property. The keys which are mapped to themselves by the similarity relation can be treated as weak-keys, and they even allow to mount various attacks when the cipher is used to build a hash function.

Deterministic self-similarity properties can usually be detected with a very small number of queries (one or two), over *any* number of rounds of a cipher, making them very interesting properties to study. However, it should be noted that most attacks involving self-similarity properties are expected to be in the related-key setting and/or will only affect classes of weak keys. This restriction is less problematic in the context of a hash function: there is no secret involved, and the adversary has a greater control over the inputs to the primitive (depending on how exactly the hash function is built, and the attack model).

We also stress that our distinguishers are very simple and efficient, so that they can be practical if the block cipher is used in an unusual setting. For example, a well-known self-similarity property of TEA is that each key has four equivalent keys [15].[3] This does not seem to be a practical threat for the block cipher (up to a loss of two bits of security in exhaustive search). However, Microsoft used

[3] We note that this property is commonly described as a related-key differential with probability 1, or a complementation property.

TEA as a hash function in Davies-Meyer mode to enforce limitations on the Xbox, and this weakness of TEA has been used *in practice* to bypass the security limitations [28].

We show an example of self-similarity properties in *Lesamnta*, and discuss a probabilistic self-similarity for *ESSENCE*. Using this approach we have identified an iterative related-key differential for XTEA. We also show a self-similarity property in Lucifer, and find a class of weak keys in the block-cipher $\mathcal{PURE}$.

One of the interesting outcomes of this research, is a new way to tackle round constants. While differing round constants seem to thwart slide attacks, they are not necessarily sufficient to protect against our more generalized approach, as we present in the attack on the compression function of *Lesamnta*.

1.1 Road-map

First we formally define the notion of self-similarity in Section 2, and we show some ways to exploit this property. In particular we discuss new attacks based on keys which are fixed under the similarity relation. Then we show concrete example of self-similarity relation: we study *Lesamnta* in Section 3, *ESSENCE* in Section 4, $\mathcal{PURE}$ in Section 5, round-reduced XTEA in Section 6, Lucifer in Section 7, and for a weak message, salt and counter combination in *SHAvite-3* in Section 8. We conclude the paper in Section 9.

2 Self-Similarity

Following [2], we define self-similarity as follows:

Definition 1 (Self-similarity relation in a block cipher). *A block cipher E encrypts the plaintext P under the key K to $E_K(P)$. A self-similarity relation is given by invertible and easy to compute transformations*[4] *ϕ, ψ and θ such that:*

$$\forall K, P: \quad \theta(E_K(P)) = E_{\psi(K)}(\phi(P))$$

If such relations exists then the cipher is self-dual according to the terminology of [2]. Similarly, we can define self-similarity for compression functions and for stream ciphers:

Definition 2 (Self-similarity relation in a compression function). *A compression function H maps a chaining value X and a message M to a new chaining value $H(X, M)$. A self-similarity relation is given by invertible and easy to compute transformations ϕ, ψ and θ such that:*

$$\forall X, M: \quad \theta(H(X, M)) = H(\phi(X), \psi(M))$$

[4] In this context, and for the reminder of the paper, we require that at least one of the three transformations ϕ, ψ, and θ is not the identity.

Definition 3 (Self-similarity relation in a stream cipher). *A stream cipher G generates the key-stream $G_K(I)$ with the key K and the IV I. A self-similarity relation is given by invertible and easy to compute transformations ϕ, ψ and θ such that:*

$$\forall K, I: \quad \theta(G_K(I)) = G_{\psi(K)}(\phi(I))$$

We say that a set of transformation is a *weak self-similarity* if this relation only holds with a probability which is less than 1 (but higher than for a random permutation). These definitions are wide enough to include several types of known attacks such as the complementation property of DES, for which $\phi = \psi = \theta$, and $\phi(x) = \overline{x}$. Other known results also fit our framework of self-similarity properties:

- The complementation property of LOKI [18]:

  $$\psi(K) = K \oplus \Delta \qquad \phi(P) = P \oplus \Delta \qquad \theta(C) = C \oplus \Delta$$

  For several values of Δ.
- The equivalent keys of TEA [15]

  $$\psi(K) = K \oplus \Delta \qquad \phi(P) = P \qquad \theta(C) = C$$

  For several values of Δ.
- The recently found weakness in the compression function of the SHA-3 candidate CHI [1]:

  $$\psi(K) = \overline{K} \qquad \phi(P) = \overline{P} \qquad \theta(C) = C$$

It is possible to consider high probability differentials as (weak) self-similarity properties. For example, in Section 6 we present some iterative related-key differential for XTEA that was found using the self-similarity approach.

In this paper, we will consider iterated self-similarity properties. Let E be a cipher defined by the iteration of a round function F, with the subkeys RK_i derived for the master key K by a function G: $RK_i = G(K, i)$, *i.e.*, the cipher can be described as:

$$RK_i = G(K, i) \qquad X_0 = P \qquad X_{i+1} = F(X_i, RK_i) \qquad E_K(P) \stackrel{\triangle}{=} X_r.$$

We look for a self-similarity property of the round function F, *i.e.*, two transformations Θ, Ψ such that $\Theta(F(X, RK)) = F(\Theta(X), \Psi(RK))$. Then assuming we can find (or construct) K, K' such that $G(K', i) = \Psi(G(K, i))$, we have $E_{k'}(\theta(P)) = \theta(E_k(P))$. Note that the relation we are using on the subkeys is defined as $RK'_i = \Psi(RK_i)$: each subkey of the second cipher is related to the corresponding subkey of the original cipher. This is in contrast with related-key attacks where the relation is $RK'_{i+1} = RK_i$.

2.1 Attacks Based on Self-Similarity

Self-similarity properties obviously offer an efficient distinguisher in the related-key setting. If one is given access to an oracle E_{K^*} and an oracle $E_{\psi(K^*)}$ for an unknown K^*, he can distinguish the block cipher E from an ideal cipher by querying $E_K(P)$ and $E_{\psi(K)}(\phi(P))$ for a random P.

Moreover, a self-similarity property can be used to speed up the exhaustive search of the key by a small factor, as explained in [4]. Let n be the size of the longest cycle of the permutation ψ. In most cases, n will be quite small (if n is big then the attack will actually be more efficient). Then, query $C_i = E_{K^*}(\phi^{(i)}(P))$ for $i \in 0, 1, \ldots, n-1$, and compute $\widehat{C_i} = \theta^{(-i)}(C_i)$. Now, compute $E_K(P)$ for a set of keys K and look for a match with one of the $\widehat{C_i}$. If there is a match, then $\psi^{(i)}(K)$ is likely to be the key:

$$\begin{aligned} E_K(P) = \widehat{C_i} &\iff E_K(P) = \theta^{(-i)}(C_i) \\ &\iff E_{\psi^{(i)}(K)}(\phi^{(i)}(P)) = C_i = E_{K^*}(\phi^{(i)}(P)). \end{aligned}$$

The idea of the attack is that the set of tested keys has to include only one key per each cycle of ψ. Each time a new $E_K(P)$ is computed, this allows to test all the key candidates in the cycle of K, by applying θ iteratively to the obtained ciphertext, and comparing the resulting ciphertext with the respective pre-computed ciphertext. Hence, if the evaluation of θ is faster than $E_{\psi(K)}$, one can reduce the time of exhaustive search.

2.2 Attacks Based on Fixed points of Self-Similarity relations

Some of the ϕ and ψ relations that we show have a large number of fixed points. These points can be used in several different attack scenarios. The fixed points of ψ will be weak key, so we consider the fixed points of ϕ as weak plaintexts.

First, let us show that the set of the fixed points of ψ is a weak-key class. If a weak plaintext is encrypted under a weak key, the ciphertext will also be a fixed point of the similarly relation, *i.e.*, $\theta(C) = \theta(E_K(P)) = E_{\psi(K)}(\phi(P)) = E_K(P) = C$. This allows to distinguish the weak keys with a single query using a weak plaintext.

In the context of hash functions, fixed points in the similarity relations allow to find collisions in the compression function more efficiently than exhaustive search. One just has to evaluate the compression function on weak inputs, and the output would lie in the set of fixed point of θ. Note that the attack becomes more efficient when the number of fixed-points in θ becomes smaller. For example, this can be used on a Matyas-Meyer-Oseas (MMO) transformation of Lucifer into a compression function (cf. Section 7).

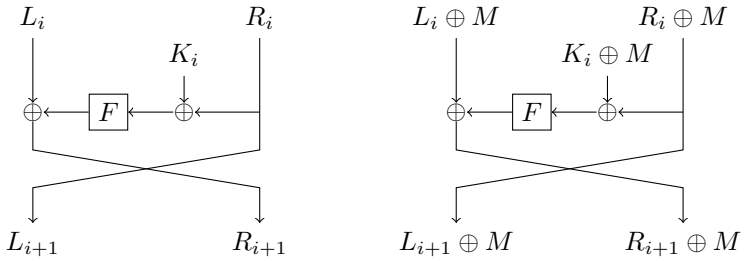
In Section 3.1 we show how to use the self-similarity property of *Lesamnta* to reduce its security in several usage scenarios. These attacks are based on the idea that one can randomly reach a fixed point of ϕ, and from there it is easy to reach any fixed point of θ. In this case, the attacks become more efficient when the number of fixed points of ϕ grows.

2.3 How to Find Self-Similarity Properties

The main part of a self-similarity attack is to find a suitable self-similarity property. Here are a few candidates that should be considered:

Related-key/Slide Attack. If the key schedule can be seen as the iteration of a fixed function ψ, then we can have a self-similarity by guessing the subkey used in the first round and after the last round. We define ϕ and θ that correspond to one round of the cipher with the guessed keys, and if the guess is right we have $\theta(E_K(C)) = E_{\psi(K)}(\phi(C))$. This idea was first in [4,5,16] against LOKI and Lucifer.

Bit Flipping. An obvious possibility is to negate some or all of the bits of the state or of the key. It is very efficient against Feistel schemes where the key schedule only selects some subset of the key bits for each round because the flipping in the key is cancelled by the flipping in the state and the input of the non-linear function is not changed. This self-similarity is known as the complementation property of DES and LOKI [18]: $DES_{\overline{K}}(\overline{P}) = \overline{DES_K(P)}$. It has also been used to build a distinguisher of GOST in [17].



Some differential attacks on hash functions can also been seen as a self-similarity property with bit flipping. Den Boer and Bosselaers show an attack on MD5 [10] based on the fact $\mathsf{MD5\text{-}Compress}(H \oplus \Delta_{\mathsf{msb}}, M \oplus \Delta_{\mathsf{msb}}) = \mathsf{MD5\text{-}Compress}(H, M)$ with probability 2^{-64}. Recently, it has been shown that the SHA-3 candidate CHI has a similar weakness, *i.e.*, $\mathsf{CHI\text{-}Compress}(\overline{H}, \overline{M}) = \mathsf{CHI\text{-}Compress}(H, M)$ with probability 1 [1].

Rotating or Swapping Parts of State. Another type of self-similarity relations may be using rotations. This is useful if the cipher uses layers of identical S-Boxes or bitwise functions. A noteworthy example of transformation that commutes with some rotations of the state is the AES's round function. We use this property in Section 3. Rotations can also give a weak self-similarity property on ciphers using a linear function based on an LFSR, as shown in Section 4.

Algebraic Relations. If the cipher has a very strong algebraic structure, it might be possible to find algebraic relations that interact nicely with the cipher operations. In Section 5 we show a new attack using the squaring operation in a field of characteristic 2.

Combination of Relations. It is possible to combine two relations of different kinds to build a new one. For example, our attacks of Sections 3 and 5 combine bit-flipping with a relation specific to the structure of the round function.

3 Application to *Lesamnta*

Our most interesting result is a self-similarity property of *Lesamnta*. This self-similarity property is based on swapping the two halves of the state and XOR-ing them with a constant. The most surprising part about this property is that it can actually deal with the round constants, which are supposed to break all symmetry relations.

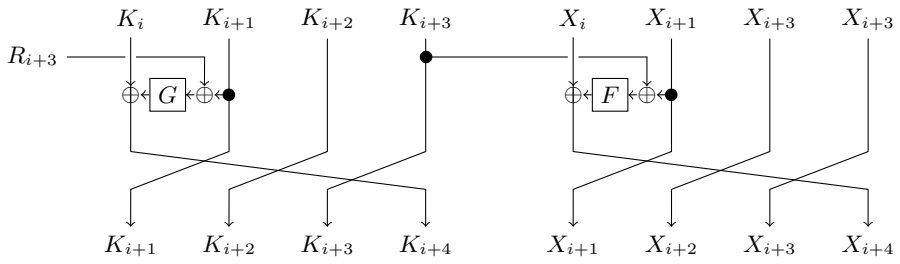


Fig. 1. The Round Function of *Lesamnta*. This Round Function is Iterated 32 Times.

A Short Description of *Lesamnta*. *Lesamnta* is a hash function proposal by Hirose, Kuwakado, and Yoshida as a candidate in the SHA-3 competition [12]. It is based on a 32-round unbalanced Feistel scheme with four registers used in MMO mode. The key schedule is also based on a similar Feistel scheme. The round function is described by Figure 1 and can be written as:

$$X_{i+4} = X_i \oplus F\left(X_{i+1} \oplus K_{i+3}\right)$$
$$K_{i+4} = K_i \oplus G\left(K_{i+1} \oplus R_{i+3}\right)$$

where $R_0, \ldots, R_{31}$ are round constants, the state register X is initialized with the message in X_{-3}, X_{-2}, X_{-1}, X_0, and the key register is initialized with the chaining value in K_{-3}, K_{-2}, K_{-1}, K_0. The output of the compression function is $X_{-3} \oplus X_{29}, X_{-2} \oplus X_{30}, X_{-1} \oplus X_{31}, X_0 \oplus X_{32}$.

Lesamnta has two main variants: *Lesamnta*-256 with 64-bit registers (hence, a message block and chaining value of 256 bits), and *Lesamnta*-512 with 128-bit registers (hence, a message block and chaining value of 512 bits).

The round functions F and G are inspired by the AES round function. F uses four round of transformations similar to SubBytes, ShiftRows and MixColumns, while G uses only one round of similar transforms. The transformations used in F and G are different, even though they are both heavily inspired by the AES. In *Lesamnta*-256 a 64-bit register is represented by a 2-by-4 byte matrix in F, and by a 4-by-2 matrix in G. In *Lesamnta*-512, a 128-bit register is seen as a 4-by-4 matrix. The round constants are defined by a simple counter: $R_i = 2i + (2i+1) \cdot 2^{32}$ for *Lesamnta*-256 and $R_i = 2i + (2i+1) \cdot 2^{64}$ for *Lesamnta*-512. For more details, we refer the reader to the full specification [12].

The Self-Similarity Relation of *Lesamnta*. The round functions F and G are very similar to the AES round function, and they have the same self-similarity property: if the two halves of the input are swapped, then the output is also swapped. Indeed, it is easy to see that SubBytes, ShiftRows and MixColumns do have this property. However, the key-schedule of *Lesamnta* includes round constants R_i to avoid symmetry-based attacks.

Luckily, these constants are word-symmetric up to the least significant bit. More precisely, if $R_i = (R_i^\top \| R_i^\bot)$, where the top and bottom parts of R_i are 32-bit (64-bit in *Lesamnta*-512), then the only difference between $R_i^\top$ and $R_i^\bot$ is in the least significant bit.

Let us introduce a few notations. We say that two words x and y are "half-swapped" if the top half of x (denoted by $x^\top$) is the bottom half of y (denoted by $y^\bot$), and vice-versa. We denote the half-swapped value of $x = (x^\top \| x^\bot)$, i.e., $(x^\bot \| x^\top)$, by $\overleftrightarrow{x}$. We also formalize the structural property of the round constants: we define $\widetilde{x}$ to be $(x^\bot \oplus 1 \| x^\top \oplus 1)$. Then, the following property of rounds constants holds: $R_i = \widetilde{R_i}$. We extend these two relations to vectors of words in the natural way, namely, $\widetilde{(x, y)} = \widetilde{(x', y')}$ if $x = \widetilde{x'}$ and $y = \widetilde{y'}$.

Our idea is to combine the swapping of halves of the state with flipping the least significant bit of each half to compensate the difference in the constants. The swapping commutes with the round functions F and G, and the masking is canceled by the Feistel structure. For this purpose, let us mention the following useful properties:

$$\begin{aligned} &i) \quad \widetilde{x} \oplus \widetilde{y} = \overleftrightarrow{x \oplus y} \\ &ii) \; \widetilde{x} \oplus \overleftrightarrow{y} = \widetilde{x \oplus y} \end{aligned}$$

Hence, consider a master keys K of 4 words, and let $K' = \widetilde{K}$, *i.e.*, by definition, $K'_{-3} = \widetilde{K_{-3}}, \ldots, K'_0 = \widetilde{K_0}$. The first property implies in particular that $K'_{-2} \oplus R_0 = \overleftrightarrow{K_{-2} \oplus R_0}$. Therefore, when computing K_1 and K'_1, the two values that enter G are half-swapped. This property goes through all the successive operations in G (SubWords, KeyLinear, and ByteTranspose[5]). The output of G is therefore

[5] We note that in the submission document [12] the term "ByteTranspos" is used.

half-swapped as well. Then, thanks to the second property, and thanks to the fact that $K'_{-3} = \widetilde{K_{-3}}$, we find that $K'_1 = \widetilde{K_1}$. This argument can be iterated, and shows that if the master keys are related, then all the other subkeys are related (we have $K'_i = \widetilde{K_i}$ for all i).

Now, it is easy to see that as the subkeys in the two concurrent hash processes are related and at the same time so are the "plaintexts", then the same relation will be maintained through the rounds. Specifically, let the plaintexts be P and $P' = \widetilde{P}$. By definition again, we have $X'_{-3} = \widetilde{X_{-3}}$, ..., $X'_0 = \widetilde{X_0}$. The argument above repeats: we have $X'_{-2} = \widetilde{X_{-2}}$, $K'_0 = \widetilde{K_0}$, and thanks to the first property, the input of F is half-swapped. This property goes trough F, and since $X'_{-3} = \widetilde{X_{-3}}$, the second property grants us $X'_1 = \widetilde{X_1}$. This argument can be iterated again, and shows that $X'_i = \widetilde{X_i}$, for all i.

Lastly, there is a feed-forward operation: the output of the compression function is $Y_0 = X_{-3} \oplus X_{29}, \ldots, Y_3 = X_0 \oplus X_{32}$. Thanks to the first property again, we find that $Y' = \overleftrightarrow{Y}$. This yields:

$$CF(\widetilde{K}, \widetilde{M}) = \overleftrightarrow{CF(K, M)}$$

Self-Similarity of *Lesamnta*. Finally, we note that if we pick a chaining value h which is weak (*i.e.*, $h = \widetilde{h}$), and a message block m which is also weak (*i.e.*, $m = \widetilde{m}$), then $CF(h, m)$ is weak as well, but in a different manner (we have $CF(h, m) = \overleftrightarrow{CF(h, m)}$), and this can be easily identified (the top and bottom halves of each output word are the same). Hence, it is possible to distinguish the compression function of *Lesamnta* using one single query. For example, in *Lesamnta*-256

$$h = m = \begin{pmatrix} (\texttt{00000000}, \texttt{00000001}), (\texttt{00000000}, \texttt{00000001}), \\ (\texttt{00000000}, \texttt{00000001}), (\texttt{00000000}, \texttt{00000001}) \end{pmatrix}$$

leads to

$$CF(h, m) = \begin{pmatrix} (\texttt{52afa888}, \texttt{52afa888}), (\texttt{61c0aebc}, \texttt{61c0aebc}), \\ (\texttt{1c9d4d3a}, \texttt{1c9d4d3a}), (\texttt{95f45a98}, \texttt{95f45a98}) \end{pmatrix}$$

and in *Lesamnta*-512

$$h = m = \begin{pmatrix} (\texttt{0000000000000000}, \texttt{0000000000000001}), \\ (\texttt{0000000000000000}, \texttt{0000000000000001}), \\ (\texttt{0000000000000000}, \texttt{0000000000000001}), \\ (\texttt{0000000000000000}, \texttt{0000000000000001}) \end{pmatrix}$$

leads to

$$CF(h, m) = \begin{pmatrix} (\texttt{b0421baf4899c67e}, \texttt{b0421baf4899c67e}), \\ (\texttt{e6b528589fadd0ce}, \texttt{e6b528589fadd0ce}), \\ (\texttt{3547c4021eb4c7ee}, \texttt{3547c4021eb4c7ee}), \\ (\texttt{a8188b26052d044d}, \texttt{a8188b26052d044d}) \end{pmatrix}$$

Following our findings, the designers of *Lesamnta* decided to tweak the algorithm by changing the round constants. For the tweaked version we refer the reader to [26].

3.1 Using These Properties on the Full *Lesamnta*

Faster Collisions in the Compression Function. It is possible to use the above property to find collisions in the compression function faster than exhaustive search. We pointed out that if we pick weak inputs, $h = \widetilde{h}$ and $m = \widetilde{m}$, then each of the four output words has the same top and bottom halves. In other words, the output of the compression function is restricted to a subspace of size $2^{n/2}$ for *Lesamnta-n*.

Hence, by taking $2^{n/4}$ pairs of (chaining values, message blocks) which are weak, we expect to find a collision in the output of the compression function.

Second Preimage Attack on Weak Messages. The self-similarity property of the compression function induces a set of weak messages. In such messages, one of the chaining values h_i (which is the output of the compression function) encountered during the hash process is such that $h_i = \overleftrightarrow{h_i}$, *i.e.*, it is of the form $h_i = (S||S), (U||U), (W||W), (Y||Y)$. A random $(r+1)$-block long message is in the weak set with probability $r \cdot 2^{-n/2}$. We now discuss how to find a second preimage of a weak message in time $2^{n/2}$ and negligible memory. Finding a new message of $(i-1)$ blocks yielding the chaining value h_i immediately yields a second preimage on the full hash function. To do so, an adversary may follow these steps:

1. Choose an arbitrary prefix of $(i-2)$ message blocks.
2. Find a message block which yields a weak chaining value h_{i-1}. Since there are $2^{n/2}$ such chaining values, it is expected that $2^{n/2}$ random trials are sufficient.
3. Find a weak message block m such that $CF(h_{i-1}, m) = h_i$. On average, for each starting chaining value, there is one such message block, amongst $2^{n/2}$ possible choices. Thus, about $2^{n/2}$ random trials are expected to be necessary.
4. Concatenating all parts with the suffix of the original message starting from the i-th block yields a second preimage.

The total complexity of the process is $2^{n/2}$.

Herding Attack on *Lesamnta*. The self-similarity property of the compression can be used to propose a better herding attack on *Lesamnta* than the one of [14]. In the herding attack, the adversary commits to a hash value H^*, and is given a challenge message M afterwards. His goal is to find a suffix S such that $H(M\|S) = H^*$. The generic herding attack on hash functions of n bits presented in [14] requires an offline time complexity of $2^{(n+\ell)/2+2}$ online time complexity of $2^{n-\ell}$ and memory of $2^{\ell+1}$. We now present a customized herding attack using self-similarity for *Lesamnta* with complexity $2^{n/2}$ and negligible precomputation and memory.

1. Choose a weak chaining value h^*, and choose an upper-bound L on the number of message blocks of the prefix M. Then, compute the finalization

function on h^*, assuming a message of $L+2$ full blocks. This yields the committed value H^* (of a message of length $L+2$ blocks).
2. Receive the challenge M, and append random suffixes S of $L+1-|M|$ blocks, until hitting a weak chaining value h. It is expected that $2^{n/2}$ random trials are sufficient.
3. Try random weak message blocks S'. We know that $H(M\|S\|S')$ is weak, with $H(M\|S\|S') = \overleftrightarrow{H(M\|S\|S')}$. Therefore, we expect to reach h^* after only $2^{n/2}$ trials.
4. Output $M\|S\|S'$ as the answer to the challenge.

Distinguishing-H Attack on HMAC. The self-similarity property can also be used to distinguish HMAC-*Lesamnta* from HMAC instantiated with another PRF. The distinguisher follows the ideas of Wang *et. al* in [29], and the complexity would be $2^{3n/4}$. We note that this line of research may not be harmful, as distinguishing HMAC from PRF (for any hash function) has a complexity of $2^{n/2}$. We also note that the claimed security level of HMAC-*Lesamnta* is only $2^{n/2}$.

4 Application to *ESSENCE*

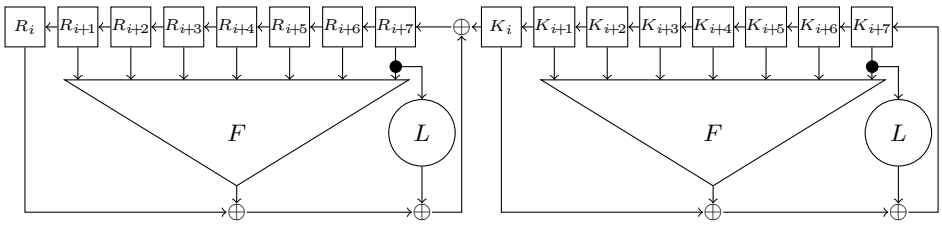


Fig. 2. *ESSENCE* Round Function (iterated 32 times).

Description of *ESSENCE*. *ESSENCE* is a hash function proposal by Martin as a candidate in the SHA-3 competition [21]. The design is based on two shift registers with 8 words each. One shift register is used to expand the message by generating subkeys, while the other processes the chaining value with the subkeys.

$$R_{i+8} = R_i \oplus F\left(R_{i+1}, R_{i+2}, \ldots, R_{i+7}\right) \oplus L(R_{i+7}) \oplus K_i$$
$$K_{i+8} = K_i \oplus F\left(K_{i+1}, K_{i+2}, \ldots, K_{i+7}\right) \oplus L(K_{i+7})$$

The feedback function is designed with two functions: a non-linear bit-wise function F, and a linear function L that mixes the bits inside the words. The linear function is based on clocking an LFSR a fixed number of times. The

chaining value is loaded into R_{-7}, R_{-6}, ..., R_0, while the message block is loaded into K_{-7}, K_{-6}, ..., K_0. After 32 rounds, the output is computed as $R_{-7} \oplus R_{25}$, $R_{-6} \oplus R_{26}$,..., $R_0 \oplus R_{32}$.

ESSENCE has two main variants: *ESSENCE*-256 with 32-bit words (hence a message block and a chaining value of 256 bits each), and *ESSENCE*-512 with 64-bit words (hence a message block and chaining value of 512 bits). The best known attack on *ESSENCE* is a collision attack with complexity 2^{68} [24]. The design of *ESSENCE* uses no constants. Therefore, it is possible to build a distinguisher against *ESSENCE* using a slide attack [22].

The Self-Similarity Relation. Most of the components used in *ESSENCE* are bitwise, and the L is the only part responsible of mixing the bits of the words. Moreover, it is based on an LFSR, and LFSRs have a good behavior with regards to rotations. More precisely we have $\Pr(L(x^{\lll 1}) = L(x)^{\lll 1}) = 1/4$ as shown in Figure 3. Therefore, we can build a new self-similarity attack based on rotating the message and the chaining value. This gives subkeys such that $K'_i = K_i^{\lll 1}$ as opposed to $K'_i = K_{i+1}$ in the slide attack. More precisely, we consider the following relation:

$$K'_i = K_i^{\lll 1} \qquad R'_i = R_i^{\lll 1}$$

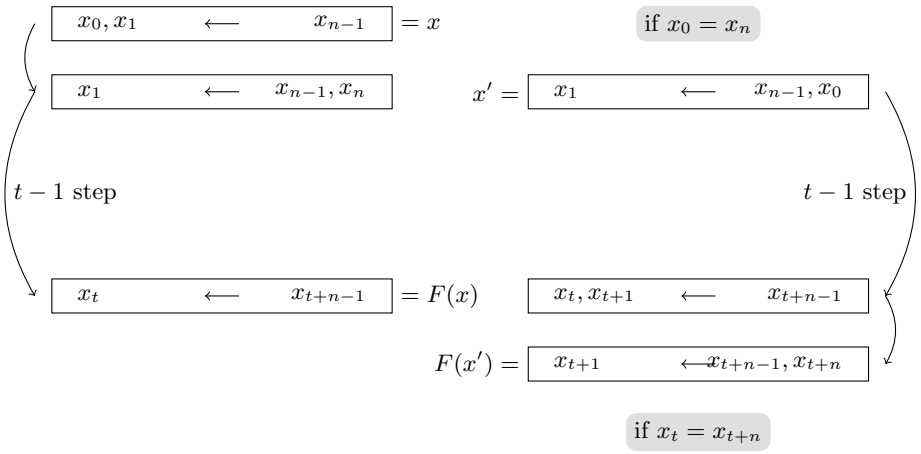


Fig. 3. Symmetry relation on an LFSR-based function. L is a linear function defined by clocking an LFSR a fixed number of times. If the initial state of the LFSR is rotated by 1 place, there is a probability 2^{-2} that the output state is rotated as well.

Constructing a Good Pair. Let us rotate the master key: $K'_{-7} = K_{-7}^{\lll 1}$, $K'_{-6} = K_{-6}^{\lll 1}$, ..., $K'_0 = K_0^{\lll 1}$. For each round, there is a probability 1/4 that

the new subkey K'_i is equal to $K_i^{\lll 1}$, because we only need the LSFR-based function to commute with a rotation of 1 bit. The full compression function uses the subkeys K_{-7} to K_{24}, so a random message K and its related message K' will give related subkeys with probability 2^{-48}.

To find a suitable chaining value we will use something similar to message modification techniques to get 8 rounds for free. We start from round 31 (one before the last round) and make a random choice of R_{24}, R_{25}, ..., R_{31} such that each of these value satisfy $L(R_i^{\lll 1}) = L(R_i)^{\lll 1}$. We use the related state $R'_{24} = R_{24}^{\lll 1}$, $R'_{25} = R_{25}^{\lll 1}$,..., $R'_{31} = R_{31}^{\lll 1}$. We first compute round 32 forward, and R_{32} follows the relation with probability 1 because the non bitwise part is $L(R_{31})$ and R_{31} was chosen so that $L(R_{31}^{\lll 1}) = L(R_{31})^{\lll 1}$. Then we compute the remaining round backwards and check that the new values still satisfy $R'_i = R_i^{\lll 1}$:

$$R_i = R_{i+8} \oplus F\left(R_{i+1}, R_{i+2}, \ldots, R_{i+7}\right) \oplus L(R_{i+7}) \oplus K_i.$$

In rounds 23, 22, ..., 17, the non-bitwise part is respectively $L(R_{30})$, $L(R_{29})$, ..., $L(R_{24})$ so they go through with probability 1. Then, we compute rounds 16 to -7, and each of cost a probability 1/4. In total we have a probability 2^{-48} that a random choice of R_{25}, R_{26}, ..., R_{32} gives a correct chaining value.

Hence, with complexity 2^{48}, we can construct a pair of messages and chaining values such that:

$$K' = K^{\lll 1} \qquad R' = R^{\lll 1} \qquad G(K', R') = G(K, R)^{\lll 1}$$

It might be possible to use advanced message modifications to further improve this complexity.

5 Application to $\mathcal{PURE}$

$\mathcal{PURE}$ is a block cipher introduced by Jakobsen and Knudsen to demonstrate the interpolation attack [13]. It is designed as a block cipher with a very strong algebraic structure, and good resistance to differential and linear cryptanalysis. However, it is weak against algebraic attacks, which was the point of the article. Later, Buchmann *et al.* defined FLURRY, which can be seen as a generalized version of $\mathcal{PURE}$ with a key schedule [9]. Again, the point of FLURRY was to show that a block cipher can be secure against differential and linear cryptanalysis, but weak against algebraic attack (Gröbner basis techniques in the case of FLURRY). $\mathcal{PURE}$ and FLURRY are not to be used as real ciphers, but serve as demonstration that algebraic attacks can be useful against ciphers with a very strong algebraic structure.

By applying a self-similarity attack to $\mathcal{PURE}$, we intend to show that the algebraic structure of a block cipher can be used to mount a simple self-similarity attack that does not need complex polynomial computation (as opposed to the interpolation attack or Gröbner basis techniques). Moreover, we build a distinguisher with only one query as well as a class of weak keys, both for any number of round, whereas the previous algebraic attacks could only break a limited number of rounds.

Description of $\mathcal{PURE}$. $\mathcal{PURE}$ is a simple Feistel cipher with a monomial S-Box. All the operations are carried in the finite field $\mathbb{F}_{2^m}$;[6] the plaintext is composed of two field elements, and the key is given as r field elements, where r is the number of rounds:

$$\begin{aligned} L_0 &= P_L & R_0 &= P_R \\ L_{i+1} &= R_i & R_{i+1} &= L_i \oplus (R_i \oplus K_i)^3. \end{aligned}$$

There is no key schedule in $\mathcal{PURE}$, the key is given as r field elements K_0, K_1, $\ldots$, K_{r-1} which are used as round subkeys.

The Self-Similarity Relation. Because of the strong algebraic structure of $\mathcal{PURE}$, it is natural to look for algebraic relations. In particular we can use the Frobenius mapping ($x \mapsto x^2$) in the field $\mathbb{F}_{2^m}$: it commutes with any monomial S-Box, and it is linear. It is straightforward to check that the following is a self-similarity relation for $\mathcal{PURE}$:

$$\theta(z) = \psi(z) = \phi(z) = z^2$$

More precisely, if $K'_i = K_i^2$ for all rounds and $P'_L = P_L^2, P'_R = P_R^2$, then we will have $L'_i = L_i^2$ and $R'_i = R_i^2$ for all rounds. This gives a very efficient distinguisher for $\mathcal{PURE}$ in the related-key setting. However, we note that since $\mathcal{PURE}$ has no key-schedule, it is trivial to make a related-key attack. Nevertheless, a slight generalization of this initial observation leads to the discovery of a class of weak keys for $\mathcal{PURE}$.

A Class of Weak Keys. Given some $\alpha \in \mathbb{F}_{2^m}$, and $0 < k < m$, we now consider the following self-similarity relation:

$$\theta(z) = \psi(z) = \phi(z) = z^{2^k} \oplus \alpha \oplus \alpha^{2^k}$$

Again, it is easy to check that this is actually a self-similarity. If $K'_i = \psi(K_i)$, $L'_i = \phi(L_i)$ and $R'_i = \phi(R_i)$, then we have:

$$\begin{aligned} R'_{i+1} &= L'_i \oplus (R'_i \oplus K'_i)^3 \\ &= L_i^{2^k} \oplus \alpha \oplus \alpha^{2^k} \oplus \left((R_i \oplus K_i)^{2^k}\right)^3 \\ &= \left(L_i \oplus (R_i \oplus K_i)^3\right)^{2^k} \oplus \alpha \oplus \alpha^{2^k} \\ &= \phi(R_{i+1}) \end{aligned}$$

It must be noted that ϕ cannot be the identity, thus it can actually be used to distinguish $\mathcal{PURE}$ used with a weak key from a random function. The weak keys are the fixed points of ψ. Now, if k divides m, then $x \mapsto x^{2^k}$ admits all the

[6] In the original description of $\mathcal{PURE}$, m was in fact 32, which yields a 64-bit cipher.

subfield $\mathbb{F}_{2^k}$ as fixed points, which means that when $K_i = \alpha + x_i$, with $x_i \in \mathbb{F}_{2^k}$, then the key is weak. It is possible to check this with only one query: just encrypt (α, α), and test the ciphertext for self-similarity. Testing all the possible α's requires 2^m queries. With $k = m/2$, this yields $2^{m+rm/2}$ weak keys out of 2^{mr}.

Bad Key-Schedules. A consequence of the previous observation is that there many bad key-schedules for $\mathcal{PURE}$. For example, let us consider the following key schedule based on a Feistel scheme with the same monomial S-Box:

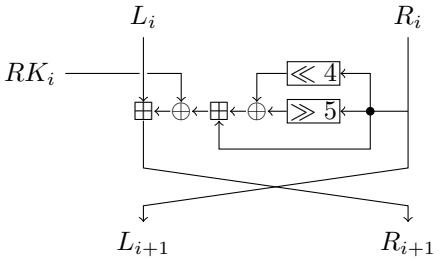
$$K_0 = K_L, \qquad K_1 = K_R$$
$$K_{i+2} = K_i \oplus (K_{i+1} \oplus C_i)^3 \quad (\text{if } i < r-1)$$

The C_i's are round constants. Their purpose is to break the regularity and avoid slide attacks. However, we now know that if the round constants can be written $C_i = \alpha + x_i$, with x_i in a subfield, then weak keys exist for the full scheme. For example, $C_i = \alpha$ and $C_{r-1} = \alpha + 1$ avoids slide properties, but is still a very bad choice since it exhibit weak keys.

6 Application to XTEA

XTEA is a block cipher designed by Needham and Wheeler. It is a Feistel network that encrypts 64-bit plaintexts with a a 128-bit key. The round function is pretty simple, and the security relies more on the high number of times it is iterated: 64 rounds are the recommended setting.

Description of XTEA. The Feistel structure of XTEA can be described as:



$$L_0 = P_L$$
$$R_0 = P_R$$
$$L_{i+1} = R_i$$
$$R_{i+1} = L_i \boxplus (F(R_i) \oplus RK_i)$$
$$F(x) = ((x \lll 4) \oplus (x \ggg 5)) \boxplus x$$

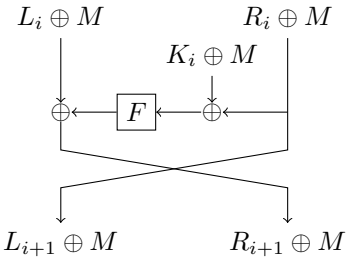
The key schedule is rather simple: the master key K is made of four 32-bit words, and the 32-bit round key RK_i is generated from the master key:

$$RK_{2i} = (i \cdot \delta) \boxplus K_{((i \cdot \delta) \ggg 11) \bmod 4}$$
$$RK_{2i+1} = ((i+1) \cdot \delta) \boxplus K_{((i+1) \cdot \delta) \bmod 4}$$

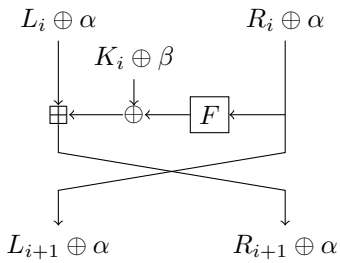
where $\delta = \texttt{0x9E3779B9}$ is a constant derived from the golden ratio.

Currently, the best known attack on XTEA is the one from [20], which can break up to 36 rounds of XTEA in the related-key model (with four keys) using 2^{65} chosen plaintexts and $2^{126.44}$ time (or 2^{64} chosen plaintexts and 2^{104} time for a weak-key class of 2^{111} weak keys).

The Self-Similarity Relation. The idea of our attack on XTEA is inspired by the complementation property of the DES. In a complementation property, the difference in the plaintext and the difference in the key cancel each other before entering the F function. Such attacks are possible if the key schedule allows a fixed difference in all the subkeys. The key schedule of XTEA is weak enough to do so, but the key is added *after* the F function, to prevent the complementation property. However, if there is a good differential characteristic $\alpha \to \beta$ in the F function, we can put a difference α in the plaintext, and a difference β in the key to cancel it after the F function. This gives an iterative related-key differential characteristic.



Complementation property



RK iterative differential on XTEA

More precisely, in the case of XTEA, we use the following weak self-similarity:

$$\psi(K_i) = K_i \boxplus 2^{31} \boxplus 2^{26}$$
$$\phi(L_i, R_i) = \theta(L_i, R_i) = \left(L_i \boxplus 2^{31}, R_i \boxplus 2^{31}\right)$$

Because of the key-schedule, we have that if $K' = \psi(K)$, then $RK'_i = \psi(RK_i)$. We also use the fact that:

$$F(x \boxplus 2^{31}) = \begin{cases} F(x) \boxplus 2^{31} \boxplus 2^{26} & \text{or} \\ F(x) \boxplus 2^{31} \boxminus 2^{26} \end{cases}$$

Therefore, the differences will cancel out (and the self-similarity will hold for the round function) as long as the carries in $F(x) \boxplus \left(2^{31} \pm 2^{26}\right)$ are the same as the carries in $K_i \boxplus 2^{31} \boxplus 2^{26}$. For a given key K, we can compute the carries in each round, and deduce the probability of the differential characteristic (which will be the probability that the self-similarity relation holds). For example if there is no carry in the key, then there is a probability $1/2$ that there will also be no carry in the F function. If there is a one-bit carry in the key, there is a probability $1/4$ that there will be exactly a one-bit carry in the F function.

The probability of the differential path is therefore quite dependent on the key. However, we know exactly the subsets of the key space that correspond to a given set of carries. Therefore, we can count the number of keys giving a specific probability for the self-similarity relation (Figure 4 shows the number of keys yielding a specific number of carries). We discuss two possible attack scenarios:

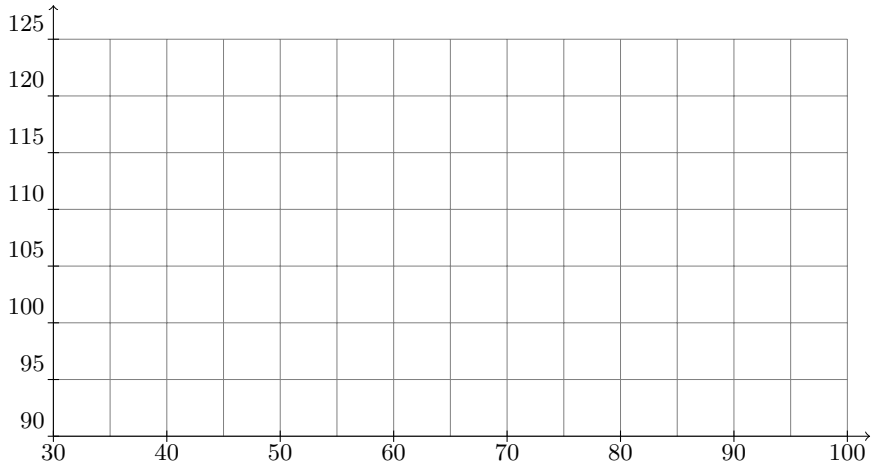


Fig. 4. Distribution of the number of carries in rounds 20–50 of XTEA. $\log_2$ of the number of keys with a given number of carries. 48% of the keys have less than 60 carries, and 52% have less than 61 carries.

attacking all the keys, and attacking a class of weak keys. Note that due to the irregular key schedule, we may obtain better attacks by selecting a subset of the rounds that do not start at round 0.

Attacking All the Keys. If we consider rounds 20 to 50, the self-similarity relation holds with probability greater than 2^{-60} for about half of the keys. If we consider 2^{62} message pairs, there is a good probability to have a right pair. If so, we can use it to recover the key, with 5 extra rounds by guessing K_2 and K_3 (which are the two keys words affecting the subkeys of these rounds).

Now assume that the self-similarity holds with probability greater than 2^{-60} for a given key, then we expect at least 4 right pairs. A wrong subkey guess is expected to lead to about $2^{-2} = 1/4$ pairs which allegedly satisfy the self-similarity property. Hence, if we consider only key proposals for K_2 and K_3 which suggest two or more pairs, we expect to deal with $2^{64} \cdot 2^{-5.23} < 2^{59}$ remaining wrong subkeys (while the probability of discarding the wrong value is only 9.2%). Hence, exhaustive search over the remaining possibilities would take less time than required for the partial decryption. Moreover, it is possible to discard wrong guesses for K_2 and K_3, as all the pairs must offer the same number of carries in round 20 and round 50. If this is not the case, then the subkey guess is necessarily wrong.

Finally, we note that if the above attack fails, the adversary learns with very good probability that the key is in the subset with more than 60 carries, which reduces the search space by 1/2. This gives an attack on 36 rounds (20 to 55), with data complexity 2^{63}, and time complexity 2^{125} on average.

This attack can be extended to more rounds if we allow for a smaller set of weak keys.

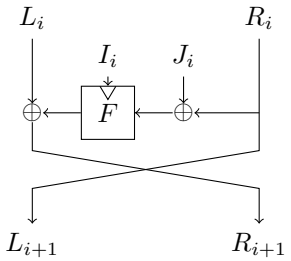
Attacking a Class of Weak Keys. If we consider rounds 10 to 55, we have a weak key class with only 60 carries, which contains $2^{107.5}$ keys out of 2^{128}. We can attack a key in this weak key class using 2^{62} message pairs. Once a good pair exist, we can use it to recover the key, with 4 extra rounds by guessing K_2 and K_3. This gives an attack on 50 rounds (10 to 59), with data complexity of 2^{63} chosen plaintexts, and time complexity 2^{123} on average, for a weak key class of size $2^{107.5}$.

A similar attack can be applied to XXTEA by changing $\psi(K_i) = K_i \boxplus 2^{26} \boxplus 2^{28}$. However, as the difference may cause more complex carry chains whose probabilities are lower, this attack can be applied only to a significantly reduced version.

7 Application to Lucifer

Lucifer is one of the first Feistel block ciphers,[7] and is usually considered as the predecessor of DES. There are many different versions of Lucifer in the literature, and we study the one from [27].

Description of Lucifer. Lucifer encrypts a 128-bit plaintext with a 128-bit key through 16 rounds.



Each round is defined as:

$$L_{i+1} = R_i$$
$$R_{i+1} = L_i \oplus F_{I_i}(R_i \oplus J_i)$$

The F function is a permutation of a 64-bit block parameterized by 8 *interchange control bits (ICB)* I_i. The F function is byte-oriented, and applies the following operations:

- For each byte, the two nibbles are swapped if the ICB $I_i^{[k]}$ is zero.
- Each nibble enters a 4-bit to 4-bit S-Box: S_0 for the low order nibble, and S_1 for the high order nibble.
- The outputs are concatenated and bit-permuted according to the following permutation:

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

[7] Actually the name "Feistel cipher" comes from Horst Feistel, one of the designers of Lucifer

The key schedule just selects key bytes as following:

$$I_i = K^{[7i \bmod 16]}$$
$$J_i = K^{[7i \bmod 16]}, K^{[7i+1 \bmod 16]}, \ldots, K^{[7i+7 \bmod 16]}$$

Because of the regularity of the key schedule, a related-key attack was shown by Biham in [4]. The best attack in the standard model is a differential attack by Ben Aroya and Biham [3] which can recover a class of weak keys that covers more than half of the key space with complexity 2^{36}.

The Similarity Relation. Because the ICB's I_i interact in a very different way than the subkeys K_i, we could not find a similarity relation for an arbitrary key. Instead, we restrict ourselves to the case where the master keys K and $\psi(K)$ give the same ICB I_i. Actually, all the key bytes are used as ICB, so we are looking for relations where $K = \psi(K)$, *i.e.*, we are looking at weak keys.

Because of the structure of the F function with layers of identical S-Boxes, we can use a self-similarity relation based on rotations. If we swap the halves of the state, then the rotation will commute with the conditional swapping of the nibbles and with the layer of S-boxes if and only if the two nibbles of the ICB are equal. Now, the diffusion layer π has the following interesting property:

$$\pi_{16..31}(x) = \pi_{0..15}(x \ggg 16)$$
$$\pi_{32..47}(x) = \pi_{0..15}(x \ggg 32)$$
$$\pi_{48..63}(x) = \pi_{0..15}(x \ggg 48)$$

This means that if we rotate the state by a multiple of 16 bits, then the rotation commutes with the round function F. So, we will set θ and ϕ to swap the two halves of the 32-bit state.

Thus, we require the subkeys J_i to be left invariant by such rotations, and by swapping the nibbles in each byte. Because the subkeys are always formed of 8 consecutive bytes of the master key (with indices taken modulo 16), the subkeys will be left invariant by rotation if and only if the master key itself is left invariant by the same rotation and nibble swapping. This is the case when the master key is of the form $AAAA$, where A is a 32-bit constant which is fixed if you swap the nibbles. This means that our class of weak key is of size 2^{16}. Given such a key, and a pair of related plaintexts, the ciphertexts will be swapped just as the plaintexts.

8 self-similarities in *SHAvite-3*

A Short Description of *SHAvite-3*. *SHAvite-3* is a proposal for the SHA3 competition by Biham and Dunkelman [6]. The hash function is a Davies-Meyer transformation of a (generalized) Feistel block cipher based on the AES. In the 256-bit variant, the block cipher is a 12-round Feistel block cipher, where the round function is composed of three-round AES (with an additional AddRoundKey

operation before the first round, and the last AddRoundKey operation omitted). The 512-bit variant is a 4-thread Feistel where the round function is applied twice (*i.e.*, to two words) in parallel in all the fourteen rounds.

SHAvite-3 received a very little cryptanalytic attention, and the only known results [23,25] involve a specific set of weak message, salt, and counter values, which cause all the subkeys of the block cipher to be equal to 0 (which leads to the introduction of slide properties, as well as fixed points). Following these findings the message expansion of *SHAvite-3* was tweaked. The following results concern only the original untweaked version of *SHAvite-3*.

Self-Similarity in *SHAvite-3*. As in *Lesamnta*, one can swap the two halves of each of the chaining value words, and as the all zero constant is fixed under the similarity relation, we obtain that all the inputs to the round functions are swapped. Recalling the results on *Lesamnta* and of [19], it is easy to see that the compression functions of *SHAvite-3* contain the same self-similarity issues when the special message (all zeroes), salt (all salt bytes are 52_x) and counter values (zero) are used.

Another self-similarity property for *SHAvite-3* is the rotation of each chaining value word by either 32 bits or 96 bits. Following [19] and the fact that the subkey zero is also a fixed point of this relation, results in the same phenomenon. Formally, let the 128-bit chaining value words be (w_0, w_1) (for *SHAvite-3*$_{256}$) and (w_0, w_1, w_2, w_3) (for *SHAvite-3*$_{512}$), and let the corresponding output chaining values be (o_0, o_1) and (o_0, o_1, o_2, o_3), respectively, then:

$$(w_0, w_1) \rightarrow (o_0, o_1) \Rightarrow (R(w_0), R(w_1)) \rightarrow (R(o_0), R(o_1))$$

and

$$(w_0, w_1, w_2, w_3) \rightarrow (o_0, o_1, o_2, o_3) \Rightarrow (R(w_0), R(w_1), R(w_2), R(w_3)) \rightarrow (R(o_0), R(o_1), R(o_2), R(o_3))$$

where $R(\cdot)$ is a rotation by 32, 64, or 96 bits.

We note that unlike the case of *Lesamnta*, this attack cannot be used on the actual *SHAvite-3*, as the counter value is not used with any message block which the adversary can control. Moreover, the tweak of *SHAvite-3* suggested in [7] thwarts this attack (due to the change of the message expansion).

9 Conclusion

This work shows one more time that symmetry in the building blocks of a cryptographic primitive can be dangerous. We have shown new ways to build symmetry relations based on rotations or algebraic expressions. We have also described new attacks based on these relations when there exists key and/or plaintexts which are fixed under the similarity relations. The common way to avoid such symmetries is to include round constants (either in the key schedule or in the actual rounds). However, our results on the hash function *Lesamnta* show that if the constants suggest some self-similarity property, it may interact with

other components to suggest a self-similarity property of the entire primitive. Hence, we conclude that some round constants can be weaker than others, and that highly structured round constants should be avoided.

Acknowledgements

We would like to thank the *Lesamnta* team, and especially to Hirotaka Yoshida for the fruitful discussions.

Part of this work is supported by the European Commission through ECRYPT, and by the French government through the Saphir RNRT project.

References

1. Aumasson, J.P., Bjørstad, T.E., Meier, W., Mendel, F.: Observation on the PRE-MIXING step of CHI-256 and CHI-224. OFFICIAL COMMENT (2009)
2. Barkan, E., Biham, E.: In How Many Ways Can You Write Rijndael? In Zheng, Y., ed.: ASIACRYPT. Volume 2501 of Lecture Notes in Computer Science., Springer (2002) 160–175
3. Ben-Aroya, I., Biham, E.: Differtial Cryptanalysis of Lucifer. In Stinson, D.R., ed.: CRYPTO. Volume 773 of Lecture Notes in Computer Science., Springer (1993) 187–199
4. Biham, E.: New Types of Cryptoanalytic Attacks Using related Keys (Extended Abstract). In: EUROCRYPT. (1993) 398–409
5. Biham, E.: New Types of Cryptanalytic Attacks Using Related Keys. J. Cryptology **7**(4) (1994) 229–246
6. Biham, E., Dunkelman, O.: The SHAvite-3 Hash Function. Submission to NIST (2008)
7. Biham, E., Dunkelman, O.: The SHAvite-3 Hash Function (Tweaked Version). Submission to NIST (2009)
8. Biryukov, A., Wagner, D.: Slide Attacks. In Knudsen, L.R., ed.: FSE. Volume 1636 of Lecture Notes in Computer Science., Springer (1999) 245–259
9. Buchmann, J., Pyshkin, A., Weinmann, R.P.: Block Ciphers Sensitive to Gröbner Basis Attacks. In Pointcheval, D., ed.: CT-RSA. Volume 3860 of Lecture Notes in Computer Science., Springer (2006) 313–331
10. den Boer, B., Bosselaers, A.: Collisions for the Compression Function of MD5. In: Proc. EUROCRYPT '93. (1993) 293–304
11. Gorski, M., Lucks, S., Peyrin, T.: Slide Attacks on a Class of Hash Functions. In Pieprzyk, J., ed.: ASIACRYPT. Volume 5350 of Lecture Notes in Computer Science., Springer (2008) 143–160
12. Hirose, S., Kuwakado, H., Yoshida, H.: SHA-3 Proposal: Lesamnta. Submission to NIST (2008)
13. Jakobsen, T., Knudsen, L.R.: The Interpolation Attack on Block Ciphers. In Biham, E., ed.: FSE. Volume 1267 of Lecture Notes in Computer Science., Springer (1997) 28–40
14. Kelsey, J., Kohno, T.: Herding Hash Functions and the Nostradamus Attack. In Vaudenay, S., ed.: EUROCRYPT. Volume 4004 of Lecture Notes in Computer Science., Springer (2006) 183–200

15. Kelsey, J., Schneier, B., Wagner, D.: Key-Schedule Cryptoanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In Koblitz, N., ed.: CRYPTO. Volume 1109 of Lecture Notes in Computer Science., Springer (1996) 237–251
16. Knudsen, L.R.: Cryptanalysis of LOKI91. In Seberry, J., Zheng, Y., eds.: AUSCRYPT. Volume 718 of Lecture Notes in Computer Science., Springer (1992) 196–208
17. Ko, Y., Hong, S., Lee, W., Lee, S., Kang, J.S.: Related Key Differential Attacks on 27 Rounds of XTEA and Full-Round GOST. In Roy, B.K., Meier, W., eds.: FSE. Volume 3017 of Lecture Notes in Computer Science., Springer (2004) 299–316
18. Kwan, M., Pieprzyk, J.: A General Purpose Technique for Locating Key Scheduling Weakness in DES-like Cryptosystems (Extended Abstract). In Imai, H., Rivest, R.L., Matsumoto, T., eds.: ASIACRYPT. Volume 739 of Lecture Notes in Computer Science., Springer (1991) 237–246
19. Le, T.V., Sparr, R., Wernsdorf, R., Desmedt, Y.: Complementation-Like and Cyclic Properties of AES Round Functions. In Dobbertin, H., Rijmen, V., Sowa, A., eds.: AES Conference. Volume 3373 of Lecture Notes in Computer Science., Springer (2004) 128–141
20. Lu, J.: Related-key rectangle attack on 36 rounds of the XTEA block cipher. Int. J. Inf. Sec. **8**(1) (2009) 1–11
21. Martin, J.W.: ESSENCE: A Candidate Hashing Algorithm for the NIST Competition. Submission to NIST (2008)
22. Mouha, N., Thomsen, S.S., Turan, M.S.: Observations of non-randomness in the ESSENCE compression function. Available online (2009)
23. Nandi, M., Paul, S.: OFFICIAL COMMENT: SHAvite-3. Available online (2009)
24. Naya-Plasencia, M., Röck, A., Aumasson, J.P., Laigle-Chapuy, Y., Leurent, G., Meier, W., Peyrin, T.: Cryptanalysis of ESSENCE. Cryptology ePrint Archive, Report 2009/302 (2009) urlhttp://eprint.iacr.org/.
25. Peyrin, T.: Chosen-salt, chosen-counter, pseudo-collision on SHAvite-3 compression function. Available online (2009)
26. Shoichi Hirose, Hidenori Kuwakado, H.Y.: Security Analysis of the Compression Function of Lesamnta and its Impact. Available online (2009)
27. Sorkin, A.: Lucifer, a cryptographic algorithm. Cryptologia **8**(1) (1984) 22–42
28. Steil, M.: 17 Mistakes Microsoft Made in the Xbox Security System. In: CCC22. (2005)
29. Wang, X., Yu, H., Wang, W., Zhang, H., Zhan, T.: Cryptanalysis on HMAC/NMAC-MD5 and MD5-MAC. In Joux, A., ed.: EUROCRYPT. Volume 5479 of Lecture Notes in Computer Science., Springer (2009) 121–133