THE PUBLISHING HOUSE
OF THE ROMANIAN ACADEMY

PROCEEDINGS OF THE ROMANIAN ACADEMY, Series A,
Volume 18, Special Issue 2017, pp. 293-306

CUBE ATTACK ON PRIMATES

Kay LUKAS[*], Joan DAEMEN[**]

[*]Eindhoven University of Technology, Proton Technologies AG
[**]Radboud University, STMicroelectronics
E-mail: kay@protonmail.ch

Abstract. GIBBON is an authenticated encryption mode of the CAESAR submission PRIMATEs. In this paper we present a state recovery attack in the nonce-reuse scenario. Our attack is a cube attack against GIBBON requiring roughly 235 10-byte input blocks. The amount of input blocks required is lower than the expected 264 10-byte input blocks. This reduction is achieved by exploiting a flaw in how the outer part is chosen in relationship to SubElements transformation that is used in the underlying PRIMATE permutation. By performing attacks on a reduced-round PRIMATE permutation, we make a prediction of the computational power required for the attack on GIBBON.

Key words: GIBBON, PRIMATEs, cube attack, CAESAR competition, higher-order differential attack, duplex construction.

1. INTRODUCTION

In this paper we contribute to the analysis of the authenticated encryption family PRIMATEs, submitted as a candidate to the CAESAR competition. However, while this paper was being written, the PRIMATEs family was deselected for the current round (round three) of the CAESAR competition. Nonetheless, we think our result can still be helpful for anyone wanting to use the PRIMATEs family of authenticated algorithms and can be helpful for the PRIMATEs designers.

To analyse the PRIMATEs family for authenticated encryption we will study differential attacks on symmetric cryptography. Differential attacks are attacks that exploit relations between differences induced in the input data and differences observed in the output data. This type of cryptanalysis was first introduced by Biham and Shamir in 1991 [4]. More specifically, we use higher-order differential cryptanalysis, which is a generalization of differential attacks and was introduced by Xuejia Lai in 1994 [11]. In higher-order differential cryptanalysis instead of using the difference between two input blocks, we study the effect of multiple differences. In this method, the concept of derivatives on a bit string mapping is used to generate linear relations between outputs of derivatives of this mapping and the input block. By solving these linear relations we can reconstruct the secret part of the input data. In Section 1.3 a closer look into higher-order differential attacks, sometimes called cube attacks, is given.

The authenticated encryption family PRIMATEs uses the duplex construction [1, p.16]. The sponge and duplex construction are constructions introduced in 2006 and 2011, respectively, by the Keccak team [3]. The SHA-3 algorithm (a subset of Keccak) uses a sponge construction and made them popular [13]. A sponge construction can be used to build an algorithm that accepts any input stream of finite length and can output a bitstring of a chosen length, dependent on the input stream. Internally, such a construction uses a finite state to store information about the input stream. This finite state is combined with a permutation to generate an arbitrarily long stream of data. The duplex construction extends this by allowing the interleaving of input data and output data. Duplex constructions can be used for authenticated encryption. The heart of both of these constructions is a permutation. Therefore, our attack exploit flaws in the permutation. The permutation used in PRIMATEs is described in Section 2.

The state of a duplex construction can be used to find information of the data that was output as well as the data that will be output. The security of a duplex construction relies on the adversary not knowing the state completely in keyed modes. Thus knowledge of the state allows predicting the output for given inputs. Therefore, our differential attack focuses on finding the state of these algorithms. This is done by performing a differential attack on the permutation of PRIMATEs. In Section 3, we document our attack on PRIMATEs.

1.1. Our contribution

We show in this paper that the six-round version of the PRIMATE permutation is not secure in a duplex construction. By using a flaw in the PRIMATE permutation our attack on the six-round version has a complexity one would expect of a five-round version: instead of requiring a small multiple of 2^{64} input blocks, we are able to perform this attack using only a small multiple of 2^{32} input blocks, showing a flaw in the PRIMATE permutation. The effect of our attack on the GIBBON encryption scheme is analysed. We found that the GIBBON encryption scheme is broken in case the adversary can reuse nonces.

This work can be related to other attacks against encryption schemes employing duplex constructions. For instance, attacks against reduced-round variants of Keyak [7] have been published.

Many examples of cube attacks on other type of constructions exists, for instance [8, 2, 14, 12, 9].

1.2. Duplex constructions

A duplex object is a stateful object with an interface supporting initialization and duplexing(M, l). It has a state with a width of b bits. The duplex object is dependent on the parameter r, the *bit rate*. The bit rate is the number of bits the duplex construction can absorb in its state before the permutation is called. The bits are absorbed into the state and so the bit rate must be smaller than the width b: $r < b$. duplexing(M, l) with $l \leq r$ is a function that, given a finite bit string M of a certain maximum length, outputs a bit string of length l dependent on M and the current state of the duplex object. This means that the output is dependent on all previous M's that were passed to the duplex object, including the order. The duplex operation should be one-way: it must be hard to reconstruct M, or any of the other previous blocks passed to the duplex object.

In Figure 1 the construction is described with a diagram. As can be seen in this figure, the duplex object has a state that can be divided into two parts. The first r bits are called the outer state. Every time the duplex operation is executed, this outer state will be output. Therefore, someone who has access to the output, knows the outer state. Furthermore, the input that is provided to the duplex operation will be absorbed by an exclusive-or into the outer state. Therefore, an adversary

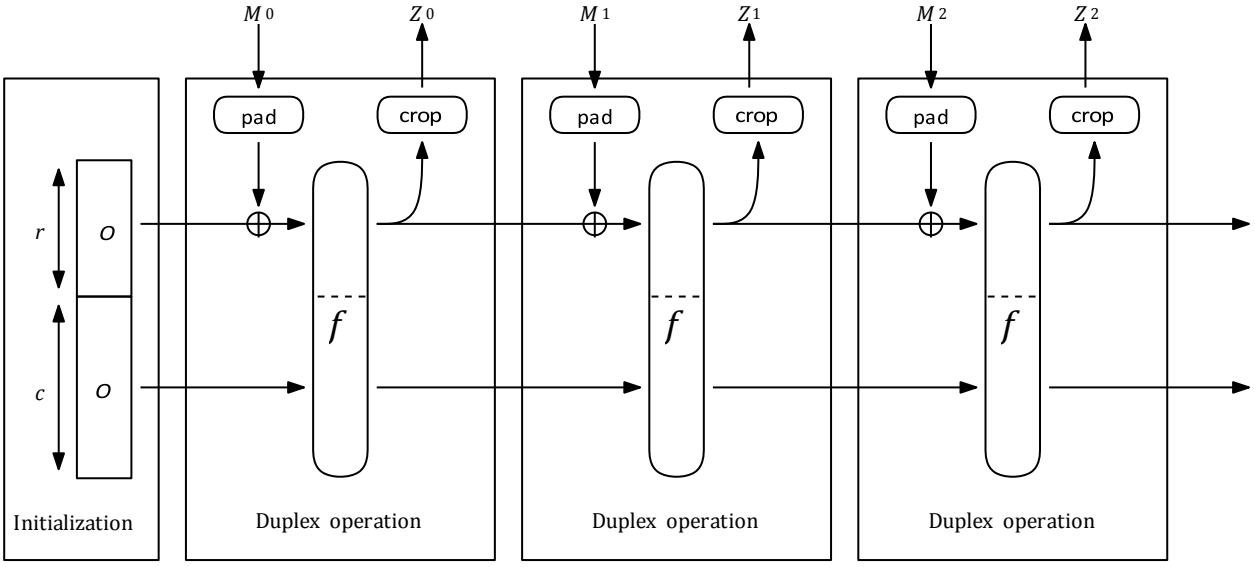

Fig. 1 – A diagram of the duplex construction.

controlling the input and having access to the output of a duplex object will know and be able to set the outer state.

The last $b - r$ bits of the state cannot be read from or written to by the user. This part of the state is referred to as the inner state, and its width is denoted by $c = b-r$, also called the *capacity*. After each duplex operation, the inner state, as well as the outer state, will depend on the previous outer and inner state, by applying the permutation f.

1.3. Higher-order differentials

The input to f is the state s and can be written as $s = (s_1, ..., s_b) = (\alpha_1, ..., \alpha_r, \beta_1, ..., \beta_c)$. Then $\alpha = (\alpha_1, ..., \alpha_r)$ is the outer state of s and $\beta = (\beta_1, ..., \beta_c)$ is the inner state. Sometimes we also speak of the outer part of the state and the inner part of the state.

To simplify the notation we define the zero vector O and the unit vectors e_j:

$$O = (0, ..., 0)$$

$$e_j = (0, \dots, 0, \overbrace{1}^{j^{\text{th}}\text{ element}}, 0, \dots, 0)$$

As explained before, an attacker can only read and write to the α_i values of s and calculate other states by performing $f(s)$. The permutation $f(s)$ generates a new state. In our analysis we split $f(s)$ per bit:

$$f(s)=(f_1(s), ..., f_i(s), ..., f_b(s)),$$

where $f_i : \mathbb{F}_2^b \to \mathbb{F}_2$. Here $\mathbb{F}_2$ is the Galois field of two elements, also called $GF(2)$. Because addition and multiplication is functionally complete in $\mathbb{F}_2$, we can write f_i as a polynomial. In $\mathbb{F}_2$ we will write a monomial $m(s) : \mathbb{F}_2^b \to \mathbb{F}_2$ as:

$$m(s) = m(s_1,\dots, s_b) = s_{i_1} \cdot s_{i_2} \cdot \dots \cdot s_{i_j} = s^H,$$

where $H = \{i_1, ..., i_j\}$ is the set of indices of all terms s_i that are in the monomial. Each index must be between 1 and b. As a shorthand, we write $\mathcal{I}_b^+$ to mean the set of all indices between 1 and b. Thus, $\mathcal{I}_b^+ = \{1, \dots, b\}$ and $H \subseteq \mathcal{I}_b^+$. We write f_i as:

$$f_i(s) = \sum_{H \subseteq \mathcal{I}_b^+} t_H s^H,$$

with $t_H \in \mathbb{F}_2$, this is called the algebraic normal form of f_i. Essentially, t_H is a boolean constant that determines if s^H is a term of f_i. Please note that f_i can also have a constant term, this is encoded as $t_\emptyset$.

By performing summation while varying various input variables, we can create a function with a lower degree than the degree of f_i. For instance, write f_i as follows:

$$\begin{aligned} f_i(s) &= \sum_{H \subseteq \mathcal{I}_b^+} t_H s^H \\ &= \sum_{H \subseteq \mathcal{I}_b^+ \setminus \{1\}} t_{(H \cup \{1\})} s^{(H \cup \{1\})} + \sum_{H \subseteq \mathcal{I}_b^+ \setminus \{1\}} t_H s^H \\ &= s_1 \cdot \sum_{H \subseteq \mathcal{I}_b^+ \setminus \{1\}} t_{(H \cup \{1\})} s^H + \sum_{H \subseteq \mathcal{I}_b^+ \setminus \{1\}} t_H s^H \\ &= s_1 \cdot p(s) + q(s), \end{aligned}$$

with $p(s)$ and $q(s)$ independent of s_1. We have split f_i into two parts: one dependent on s_1 and one independent of s_1.

We can extract $p(s)$ by changing only the first bit of s and performing an addition of the evaluations:

$$
\begin{aligned}
&f_i(0,s_2,\ldots,s_n) + f_i(1,s_2,\ldots,s_n)\\
&\quad = 0 \cdot p(0,s_2,\ldots,s_n) + q(0,s_2,\ldots,s_n) +\\
&\quad\quad 1 \cdot p(1,s_2,\ldots,s_n) + q(1,s_2,\ldots,s_n)\\
&\quad = 0 \cdot p(s) + q(s) + 1 \cdot p(s) + q(s) \qquad (p(s), q(s) \text{ independent of } s_1)\\
&\quad = p(s) + (1+1) \cdot q(s) = p(s).
\end{aligned}
$$

By extracting $p(s)$ from $f_i(s)$, we are effectively reducing the degree: if the degree of $f_i(s) = d$ then the degree of $p(s)$ must be equal to or smaller than $d - 1$.

This can be done in the same manner for other bits. Take $p(s)$ and $q(s)$ independent of bit j and such that:

$$f_i(s) = s_j \cdot p(s) + q(s).$$

For simplicity we introduce the differential notation δ_j:

$$
\begin{aligned}
&\delta_j(f_i)(s) = f_i(s + 0 \cdot e_j) + f_i(s + 1 \cdot e_j)\\
&\quad = f_i(s_1,\ldots,s_{j-1},s_j + 0,s_{j+1},\ldots,s_n)\\
&\quad\quad + f_i(s_1,\ldots,s_{j-1},s_j + 1,s_{j+1},\ldots,s_n)\\
&\quad = f_i(s_1,\ldots,s_{j-1},0,s_{j+1},\ldots,s_n) + f_i(s_1,\ldots,s_{j-1},1,s_{j+1},\ldots,s_n)\\
&\quad = 0 \cdot p(s) + q(s) + 1 \cdot p(s) + q(s)\\
&\quad = p(s)\\
&\quad = \sum_{H \subseteq (\mathcal{I}_b^+ \setminus \{j\})} t_{(H \cup \{j\})} s^H. \qquad (1)
\end{aligned}
$$

The differential operator δ_j can also be applied to the function $f = (f_1,\ldots,f_b)$:

$$
\begin{aligned}
&\delta_j(f) = f(s + 0 \cdot e_j) + f(s + 1 \cdot e_j)\\
&\quad = (f_1(s + 0 \cdot e_j) + f_1(s + 1 \cdot e_j),\ldots,f_b(s + 0 \cdot e_j) + f_b(s + 1 \cdot e_j))\\
&\quad = (\delta_j(f_1),\ldots,\delta_j(f_b))\\
&\quad = \left(\sum_{H \subseteq (\mathcal{I}_b^+ \setminus \{j\})} t^{(1)}_{(H \cup \{j\})} s^H, \ldots, \sum_{H \subseteq (\mathcal{I}_b^+ \setminus \{j\})} t^{(2)}_{(H \cup \{j\})} s^H \right)\\
&\quad = \sum_{H \subseteq (\mathcal{I}_b^+ \setminus \{j\})} t_{(H \cup \{j\})} s^H.
\end{aligned}
$$

The same properties then hold, but t_H will be in the set $\mathbb{F}_2^b$ instead of $\mathbb{F}_2$. Obviously, we need to apply the differential over multiple bits to generate a linear equation. To this end, let a set $G = \{g_1,\ldots,g_i,\ldots,g_m\} \subseteq \mathbb{N}$, $1 \leq g_i \leq n$ be given and we define:

$$
\begin{aligned}
\delta_G(f_i)(s) &= \delta_{g_1}(\delta_{g_2}(\ldots \delta_{g_m}(f_i)\ldots))(s)\\
&= \sum_{H \subseteq (\mathcal{I}_b^+ \setminus G)} t_{(H \cup G)} s^H. \qquad (2)
\end{aligned}
$$

For the appropriate choice of G, $\delta_G(f_i)$ becomes linear. However, to formulate an attack, we need to be able to determine the representation of $\delta_G(f_i)$. Firstly, we should note that $O = (0,0,...,0)^H$ is 0 except when $H = \emptyset$. When H is $\emptyset$, s^H becomes the empty product, which is 1. This gives a way to retrieve $t_\emptyset$:

$$f_i(O) = t_\emptyset.$$

To retrieve other t_H's we can combine this approach together with applying δ_G:

$$\delta_G(f_i)(O) = \sum_{H \subseteq (J_b^+ \setminus G)} t_{(H \cup G)} O^H = t_G. \quad (3)$$

One can also calculate $t_{(G \cup \{j\})}$ when t_G is known by using the following observation:

$$\delta_G(f_i)(e_j) = t_G + t_{(G \cup \{j\})}. \quad (4)$$

By generating linear equations and determining the representation of these equations, we can build a linear system. Elaborate proofs of these statements are omitted in this paper and one can read [11] or [15] for more information. Attacks generating linear systems in this way are called cube attacks.

2. DESCRIPTION OF THE PRIMATE PERMUTATION

In this paper, we will be attacking the GIBBON scheme for the authenticated encryption, which is part of the PRIMATEs family for authenticated encryption. As a crude description, GIBBON uses the duplex construction to encrypt a message M by first making the state dependent on the key. This is done by absorbing the secret key, as well as the nonce and possibly the associated data in the duplex object. Then the output stream of the duplex object is used to encrypt M. M is also fed into the duplex object. In any case, as the ciphertext of M is dependent on the state of the duplex construction, the security of the PRIMATEs encryption scheme requires that the inner state remains secret. When the inner state is known by an adversary it is possible to decrypt the output stream or forge a ciphertext including the authentication tag. Figure 2 provides the pseudo-code of the GIBBON encryption scheme. A more detailed explanation can be found in [1, p. 6].

A duplex construction is built by repeatedly applying a permutation on the state. If we can extract the inner state by setting the outer state to a certain value, applying the permutation and reading from the outer state again, we have found a vulnerability in the chosen-plaintext attack model. Therefore, we will analyse from now on the permutations of the PRIMATEs family, as such a vulnerability in the permutation will cause a vulnerability in the encryption algorithm.

The permutation PRIMATE used in the PRIMATEs authenticated encryption schemes is inspired by Fides [5] and its structure resembles the Rijndael block cipher [6][1, p. 7].

Two different permutations are defined, a 200-bit version (PRIMATE-80) and a 280-bit version (PRIMATE-120). The permutation itself uses an S-box to provide non-linearity. The state is divided into a grid of 5-bit elements, corresponding to the width of the S-box used. In PRIMATE-80, the state is divided into 5 × 8 elements of 5 bits and in PRIMATE-120, the state is divided into 7 × 8 elements of 5 bits. The PRIMATE permutation updates the internal state in rounds, where each round is a sequence of steps:

$$\text{PRIMATE-round} = \text{CA} \circ \text{MC} \circ \text{SR} \circ \text{SE}.$$

The round functions is applied multiple times, which gives the permutations p_1, p_2, p_3 and p_4. The operations applied to the state in one of the PRIMATE-rounds are:

- SE, the SubElements operation. This is the only non-linear operation in the PRIMATE permutation. This operation applies a 5-bit S-box to each element of the state. The polynomial degree of this operation is 2.

- SR, the ShiftRows step. This step is a linear operation, and circularly shifts the rows of the state. Each row is circularly shifted by a different number. The shift offset is determined by the sequence $h = (h_1, h_2, ...)$: row i is shifted left by h_i positions. In PRIMATE-80 we have $h = (0, 1, 2, 4, 7)$ and for PRIMATE-120 we have $h = (0, 1, 2, 3, 4, 5, 7)$.
- MC, the MixColumns operation. This step is a linear operation. The MixColumns operation multiplies each column with a 5×5 matrix in PRIMATE-80 or a 7×7 matrix in PRIMATE-120. This multiplication is done in the field $\mathbb{F}_2[x]/\langle x^5 + x^2 + 1\rangle$. The exact matrix is not important for our analysis and is left out.
- CA, the ConstantAddition step. This step is again a linear operation. It combines the second element of the second row with a constant *rc* using an exclusive-or operation. The purpose of CA is to make each round different and to break the symmetry of the other operations. This constant changes every round and is generated using a *Linear Feedback Shift Register* (LFSR). The ConstantAddition step can be used to generate several different permutations, by setting the initial value of the LFSR to different values. This is used, for instance, to define p_1, p_2, p_3 and p_4.

Algorithm 5: E_K(N,A,M)

Input: $K \in \boldsymbol{C}^{\frac{1}{2}}, N \in \boldsymbol{C}^{\frac{1}{2}}, A \in \{0,1\}^*$, $M \in \{0,1\}^*$
Output: $C \in \{0,1\}^*, T \in \boldsymbol{C}^{\frac{1}{2}}$

$V \leftarrow p_1(0^r||K||N)$
$V \leftarrow V_r||(K||0^{\frac{c}{2}}) \oplus V_c$
if $A \neq \emptyset$ then
  $V \leftarrow p_2(V)$
  $A[1]A[2] \dots A[u] \leftarrow A$
  $A[u] \leftarrow A[u]||10^*$
  for $i = 1$ to $u - 1$ do
    $V \leftarrow p_2(A[i] \oplus V_r||V_c)$
  end
  $V \leftarrow A[u] \oplus V_r||V_c$
end
$M[1]M[2] \dots M[w] \leftarrow M$
$l \leftarrow |M[w]|$
$M[w] \leftarrow M[w]||10^*$
$V \leftarrow p_3(V)$
for $i = 1$ to w do
  $C[i] \leftarrow M[i] \oplus V_r$
  $V \leftarrow p_3(C[i]||V_c)$
end
$V \leftarrow p_1(V_r||(K||0^{\frac{c}{2}}) \oplus V_c)$
$C \leftarrow C[1]C[2] \dots C[w-1]\lfloor C[w] \rfloor_l$
$T \leftarrow \lfloor V_c \rfloor_{\frac{c}{2}} \oplus K$
return (C, T)

Algorithm 6: D_K(N,A,C,T)

Input: $K \in \boldsymbol{C}^{\frac{1}{2}}, N \in \boldsymbol{C}^{\frac{1}{2}}, A \in \{0,1\}^*$, $C \in \{0,1\}^*, T \in \boldsymbol{C}^{\frac{1}{2}}$
Output: $M \in \{0,1\}^*$ or $\perp$

$V \leftarrow p_1(0^r||K||N)$
$V \leftarrow V_r||(K||0^{\frac{c}{2}}) \oplus V_c$
if $A \neq \emptyset$ then
  $V \leftarrow p_2(V)$
  $A[1]A[2] \dots A[u] \leftarrow A$
  $A[u] \leftarrow A[u]||10^*$
  for $i = 1$ to $u - 1$ do
    $V \leftarrow p_2(A[i] \oplus V_r||V_c)$
  end
  $V \leftarrow A[u] \oplus V_r||V_c$
end
$C[1]C[2] \dots C[w] \leftarrow C$
$l \leftarrow |C[w]|$
$V \leftarrow p_3(V)$
for $i = 1$ to $w - 1$ do
  $M[i] \leftarrow C[i] \oplus V_r$
  $V \leftarrow p_3(C[i]||V_c)$
end
$M[w] \leftarrow \lfloor V_r \rfloor_l \oplus C[w]$
$V \leftarrow p_3((M[w]||10^* \oplus V_r)||V_c)$
$M \leftarrow M[1]M[2] \dots M[w-1]M[w]$
$V \leftarrow p_1(V_r||(K||0^{\frac{c}{2}}) \oplus V_c)$
$T' \leftarrow \lfloor V_c \rfloor_{\frac{c}{2}} \oplus K$
return $T = T' ? M : \perp$

Fig. 2 – The GIBBON encryption E_K(N,A,M) and decryption D_K(N,A,C,T) algorithm. Here N is the nonce, A the associated data, M the message that is to be encrypted, C the ciphertext of M and T an authentication tag that proofs the authenticity of C. The pseudocode is taken directly from [1, p. 6] and therefore the notation is unchanged: $\oplus$ is the exclusive-or operator, $\|$ is the concatenation operator.

A visualization of the PRIMATE-round is given in Figure 3.

As we have seen, there is only one non-linear operation in a PRIMATE-round, and its degree is 2. Therefore, the degree of a permutation with a single PRIMATE-round is 2. In general, a permutation consisting of i PRIMATE-rounds has a polynomial degree of up to 2^i, but the degree is limited by the size of the permutation state. By varying the number of rounds and the initial value of *rc*, the designers of PRIMATE defined several permutations. These are given in Table 1.

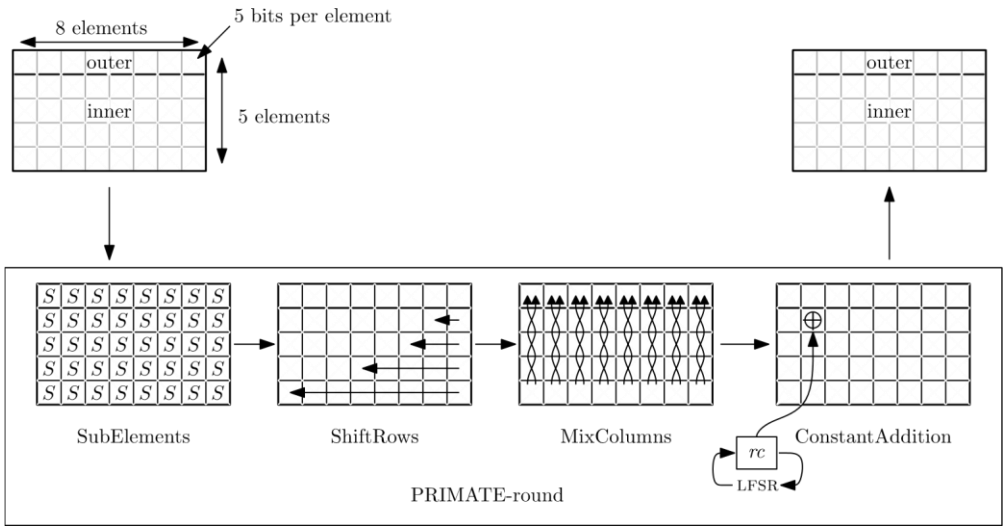

Fig. 3 − A diagram illustrating one of the PRIMATE-rounds in PRIMATE-80. The constant *rc* is updated and reused every round.

Table 1

The permutations p_1, p_2, p_3 and p_4. The polynomial degree of p_1 and p_4 is at most 199 for the 200-bit variant and 279 for the 280-bit variant as the maximum polynomial degree is limited by the number of input bits

Permutation-name	p_1	p_2	p_3	p_4
Number of rounds	12	6	6	12
Initial value of *rc*	1	24	30	24

3. OUR CUBE ATTACK

To attack the PRIMATEs scheme for authenticated encryption, we will use the cube attack defined in Section 3.1. Note that we need to vary $i-1$ bits to reduce an i bit polynomial to a linear polynomial. Unfortunately, all permutations defined have a higher polynomial degree than the number of bits in the outer state. But with a clever trick, we are still able to attack p_2 and p_3. In this section, we will only consider the permutations with security level 80, this means the permutation has an input and output size of 200 bits. From now on, we will refer to one of the PRIMATE permutations as f. We start with f as a permutation created by applying PRIMATE-round multiple times with $rc = 0$. The number of rounds f has should be clear from the context. In Section 3.3 we apply the attack for each *rc* value as described in Table 1. We abstract from this *rc* value initially (by taking $rc = 0$), as it does not make a difference for our type of attack. We limit our attack to the 200-bit variant but it can very probably be adapted to the 280-bit case too.

3.1. Attack procedure

Suppose we attack a duplex construction produced by the permutation f, with rate r, capacity c and width b. Our attack procedure has three phases:

Offline phase. Here we generate linear equations and calculate their polynomial representation. This phase does not require the usage of the actual state, and can be reused for every attack. Various sets $G_1,G_2,\ldots,G_m$ must be chosen, such that $\delta_{G_j}(f_i)(s)$ for $i \leq r$ are linear functions and reveal information about $\beta = (s_{r+1},s_{r+2},\ldots,s_b)$. Because $\delta_{G_j}(f_i)(s)$ for each $i \leq r$ can be calculated together, we generate r linear equations per set G_j. Please note that all values in G_j must be at most r, because we cannot set the values of the inner state. Write these linear equations as:

$$M \cdot x = o + d + M_\alpha \cdot \alpha.$$

Here x is a vector of unknowns:

$$x = \begin{pmatrix} s_{r+1} \\ \vdots \\ s_{r+c} \end{pmatrix}$$

o is the output of the $\delta_G(f_i)(s)$ functions and will be determined in the next phase, depending on the state,

$$o = \begin{pmatrix} \delta_{G_1}(f_1)(s) \\ \vdots \\ \delta_{G_1}(f_r)(s) \\ \delta_{G_2}(f_1)(s) \\ \vdots \\ \delta_{G_m}(f_r)(s) \end{pmatrix}$$

d are the constant values of the linear equations and α are the values that will be known when attacking the state (the outer state):

$$\alpha = \begin{pmatrix} s_1 \\ \vdots \\ s_r \end{pmatrix}$$

The linear terms in the equations are split into the known and unknown terms. The known terms will be written as M_α, the unknown terms will be written as M.

Lastly, the sets $G_1,G_2,...,G_m$ must be chosen such that the rank of M is c or is close to c, otherwise not all inner state values can be correctly determined without requiring too much computational power. We will select the sets adaptively.

Online phase. In this phase we are given the duplex construction with some state s. The first step is to manipulate the outer state values and calculate $\delta_{G_j}(f_i)$ by applying the definition given in Equation (2). Furthermore, by reading the outer state of s, we can derive the value of α. In summary, we calculate o and α in this phase.

Solving phase. This phase uses the information collected in the previous phases to derive the inner state. Firstly, we can calculate $M_\alpha \cdot \alpha$. Combining this with the vector o (the $\delta_{G_j}(f_i)$ calculated in the online phase) and c gives the constant vector $b = o + d + M_\alpha \cdot \alpha$. Then we apply a Gaussian elimination method to solve the equation $M \cdot x = y$ for x. The value of x is the inner state $\beta = (s_{r+1},...,s_b)$.

3.2. Attack method

The initial approach is to perform a basic cube attack without any modifications. This is done, by generating linear equations in the manner described in Section 1.3. This means, that we repeatedly apply the δ function to the permutation f until a linear equation is found. As a start, we tried to attack the first one and two rounds of f.

After analysing the polynomial of the one and two-round permutation, we see that no linear function can be extracted. For instance, the polynomial representation of the first bit of one round of PRIMATEs is of the form:

$$\begin{aligned} &\alpha_1\alpha_3 + \alpha_2\alpha_5 + \alpha_1 + \alpha_4 + \\ &\beta_6\beta_7 + \beta_6\beta_{10} + \beta_8\beta_9 + \beta_8\beta_{10} + \beta_{10} + \\ &\beta_{51}\beta_{54} + \beta_{52}\beta_{53} + \beta_{52}\beta_{55} + \beta_{53}\beta_{55} + \beta_{52} + \beta_{53} + \beta_{54} + \\ &\beta_{101}\beta_{104} + \beta_{102}\beta_{103} + \beta_{102}\beta_{105} + \beta_{103}\beta_{105} + \beta_{102} + \beta_{103} + \beta_{104} + \\ &\beta_{156}\beta_{157} + \beta_{156}\beta_{160} + \beta_{158}\beta_{159} + \beta_{158}\beta_{160} + \beta_{160} + 1. \end{aligned}$$

In this equation, the α variables are the outer part of the state, β variables are the inner part of the state. If we would differentiate towards an α-term, no β variables are left in the linear polynomial. This means that deriving a linear equation from this would not give any information about the inner part, which is the part where we need information about. In the two-round permutation, something similar happens.

This approach fails because the S-box is the only non-linear operation in the permutation. The S-box is only applied to groups of 5 bits at a time. The S-box does not create useable mix terms between α and β, because the SubElements step is the first step in the PRIMATE-round. When more rounds are applied, the highest degree terms will always have either at least two β-variables, or none. It can, thus, be seen that increasing the number of rounds, will not create better attack opportunities.

If we would be able to skip the SubElements step, mix terms between the inner state and outer state variables should be created. Another advantage would be that the polynomial degree would be halved, making the degree of p_2 and p_3 32, which would allow us to attack it by varying bits in the outer state.

Fortunately, it is possible to skip the first SubElements step. The PRIMATE-round was defined as:

$$\text{PRIMATE-round} = \text{CA} \circ \text{MC} \circ \text{SR} \circ \text{SE}.$$

Write the input state s as $(\alpha_1, \ldots, \alpha_{40}, \beta_1, \ldots, \beta_{160})$, write the S-box used as S and its inverse S^{-1} and we can derive:

$$\begin{aligned}\text{PRIMATE-round}(s) &= \text{CA}(\text{MC}(\text{SR}(\text{SE}(s)))) \\ &= \text{CA}(\text{MC}(\text{SR}(\text{S}(\alpha_1, \ldots, \alpha_5)) | \ldots | \text{S}(\alpha_{36}, \ldots, \alpha_{40}) \mid \text{S}(\beta_1, \ldots, \beta_5) | \ldots | \text{S}(\beta_{156}, \ldots, \beta_{160})))).\end{aligned}$$

Now instead of inputting $\alpha_1, \ldots, \alpha_{40}$ we first apply S^{-1} to each 5 bit element:

$$\begin{aligned}\text{PRIMATE-round}(s) &= \text{CA}(\text{MC}(\text{SR}(\text{S}(\text{S}^{-1}(\alpha_1, \ldots, \alpha_5)) \mid \ldots \mid \text{S}(\text{S}^{-1}(\alpha_{36}, \ldots, \alpha_{40})) \mid \text{S}(\beta_1, \ldots, \beta_5) \mid \ldots \mid \text{S}(\beta_{156}, \ldots, \beta_{160}))) \\ &= \text{CA}(\text{MC}(\text{SR}((\alpha_1, \ldots, \alpha_{40}) | \text{S}(\beta_1, \ldots, \beta_5) | \ldots | \text{S}(\beta_{156}, \ldots, \beta_{160})))).\end{aligned}$$

Now set $\text{S}(\beta_1, \ldots, \beta_5) | \ldots | \text{S}(\beta_{156}, \ldots, \beta_{160})$ as our unknown variables. If we apply our attack this way, we will find $\text{S}(\beta_1, \ldots, \beta_5) \mid \ldots \mid \text{S}(\beta_{156}, \ldots, \beta_{160})$ and we just need to apply S^{-1} to find the values of $\beta_1, \ldots, \beta_{160}$. This method is able to find linear equations and the results are discussed in the following section.

3.3. Results

The attack was implemented by first attacking two rounds, then three rounds and so on. Because the SubElements step was skipped, the polynomial representation of a single round instantly gives 40 linear equations. Unfortunately, we cannot generate more than 40 equations as we cannot apply δ_G with different values for G. Please note, that G is the set of indices of the bits that we apply the differentiation function δ on. Because there are more than 40 unknowns, we cannot use these 40 equations to find the inner state.

In our attack, we have experimented with different values $G_1,\ldots,G_n$, and the smallest number of sets giving a fully determined system are written down here. We aim to find $G_1,\ldots,G_n$ with n as small as possible, as this reduces the computation time. The exact results, and the $G_1,\ldots,G_n$ sets chosen for each round are documented in Section 3.5, including the exact specifications of the computer on which the tests are done.

In Table 2 some statistics of the attack applied on each round are given. We see that most rounds can be attacked with 8 $\delta_{G_i}(f)$ computations. It can be seen that the computation time of the offline phase and the computation time of the online and solving phase increases exponentially. Furthermore, when a small number of rounds is attacked, most of the computation time of the solving phase is caused by starting the matlab engine. Please note, as said before, that we omit the first SubElements step in f and so we half the degree of f.

For six rounds of the PRIMATE-permutation, the polynomial degree of f_i is 32. Therefore, our sets $G_1,\ldots,G_n$ should be of size 31. Calculating the linear equations for five rounds of the PRIMATE-permutation already takes a considerable amount of time. Because the calculation time will probably grow exponentially with the size of G, it is reasonable to first determine the expected calculation time of the representation of $\delta_G(f)$ with $|G| = 31$ before attempting to calculate it. This gives us the calculation time of one $\delta_G(f)$ with $|G| = 31$. It is likely that we at least need to calculate $\delta_{G_i}(f)$ for 8 different G_i's to generate a fully determined system.

Furthermore, finding a somewhat optimal $G_1, \ldots, G_n$ that generates a fully determined system also requires some trial-and-error: not all sets G with $|G| = 31$ will result in useful linear equations.

Table 2

Various statistics about the implemented cube attack on PRIMATE. The attack was performed for different rounds, which are shown in this table. $|G|$ indicates the number of sets in $G = \{G_1, ..., G_n\}$ we require to make the linear system generated by $\delta_{G_i}(f)$ fully determined. $t_{offline}$ is the time required to compute the offline phase, $t_{online} + t_{solv}$ is the time required to perform the online phase and the solving phase

Rounds	deg(f)	$\|G\|$	$t_{offline}$	$t_{online} + t_{solv}$
2	2	8	< 1 s	19 s
3	4	16	4 s	19 s
4	8	8	20 s	20 s
5	16	8	44 m	29 s

To determine the calculation time of $\delta_G(f)$ with $|G| = 31$, we have calculated $\delta_G(f)$ with $1 \leq |G| \leq 21$. As expected, for $|G|$ large enough, the calculation time will approximately double if $|G|$ increases by 1. We have extrapolated the results found and determined the approximate time to calculate $\delta_G(f)$ with $|G| = 31$: this will approximately take 234 days on our computer. This can be seen in Figure 4. While 234 days per $\delta_G(f)$ calculation is a long time, it is certainly not infeasible. Furthermore, the attack is highly parallelizable: a small to medium company could rent 234 servers of equal computing power to the used laptop to perform such a calculation in one day. Another approach would be to implement the attack using a graphics card: as the attack is highly parallelizable, a good speed-up might be achieved. As the attack only needs to compute the sum of various permuted input blocks, which can be calculated on demand, the amount of memory and the throughput required is fairly limited. This makes an implementation on a graphics card feasible.

For more than six rounds of the PRIMATE-permutation, it is impossible to apply this attack: we cannot do a cube attack using more than 40 bits, as the outer state of the duplexes used in PRIMATEs is 40 bits.

3.4. Impact on the PRIMATE-family

Only the encryption scheme GIBBON of the PRIMATEs family uses the PRIMATE permutation with 6 rounds. Therefore, we could try to attack the GIBBON scheme by finding the inner state of the permutation. For the online part of the attack, the adversary must be able to reach the same state for the duplex object. This implies that the nonce must be reused.

If the nonce would be reused multiple times we can find the inner state at line 18 of the encryption algorithm (Figure 2). Because we use a chosen plaintext attack, we know the associated data A and the input message M. Using this, we can calculate the inner state backwards until line 2. We cannot calculate the inner state at line 1, as we cannot reverse the operation at line 2 because we do not know K. Therefore, we cannot find K but we can deduce the inner state at line 2. This inner state is always the same if the same key K and nonce N are used. Therefore, this attack reveals a vulnerability in GIBBON if one would reuse the nonces.

It must be noted that in the specification of GIBBON [1, p. 4], it was stated that GIBBON was not resistant against nonce reuse. However, this vulnerability can be avoided easily at the cost of adding a single round to p_2 and p_3. Another option would be to change the SubElements step, so that it does create mix terms between the inner state and the outer state when applied as the first step. This can be done by changing layout of the 5-bit elements in such a way that each element contains bits of the inner state. We are not sure if other problems would arise, if one would change the layout in this manner.

3.5. Attack details

This section describes the details of the PRIMATE attack. The computations documented here have been implemented in C++. Solving the linear system in the last phase is performed using matlab.
The computations have been carried out on a laptop with the following specs:

- **Memory**: 8GB of RAM,
- **cpu**: Intel Core i7 5500U.

The implementation uses only a single thread. The source code can be found here: [10]. The polynomial representation of the linear equations has been calculated using Equation 3 on page 5. Please note, as said before, that we skip the first SubElements step: by skipping this step, we half the polynomial degree of f_i for any number of rounds that we attack.

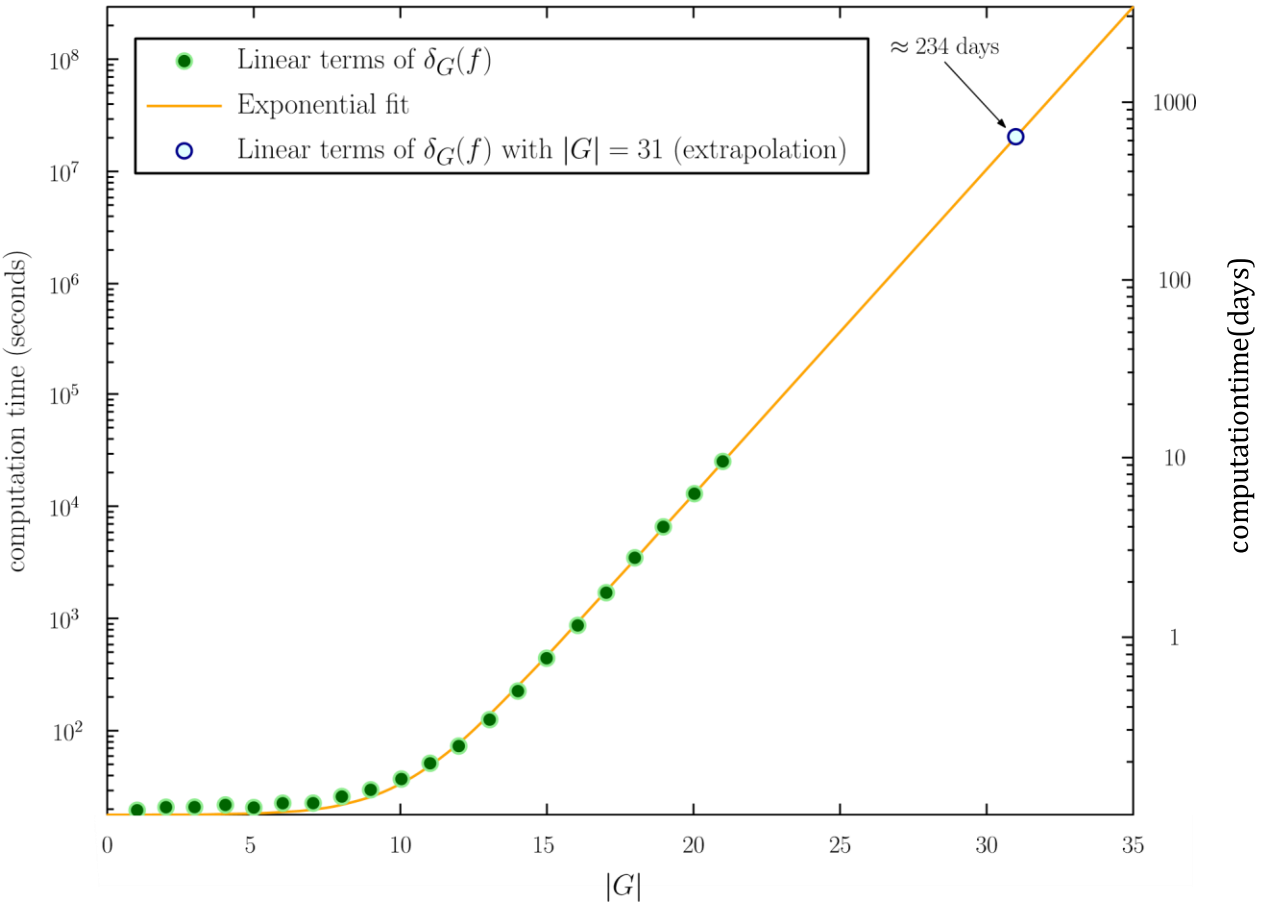

Fig. 4 – The approximate calculation time of $\delta_G(f)$ with $|G| = 31$ was calculated by extrapolating the calculation time of $\delta_G(f)$ with $1 \leq |G| \leq 21$.

For two rounds of the PRIMATE-permutation, the polynomial degree of f_i is 2. Therefore, our sets $G_1,\ldots,G_n$ should be of size 1. After experimentation, we found that a fully determined system is generated by the sets $G_1,\ldots,G_n$ only if for each 5-bit element of the outer state a set G_i contains the index of one bit that is inside that 5-bit element. In other words, this means that for each integer set $H = \{5 \cdot j + 1,\ldots,5 \cdot j + 5\}$ with $0 \leq j < 8$, there must be a set G_i such that $G_i = \{g_1\}$ with $g_1 \in H$. So we needed to perform $\delta_{G_i}(f)$ for 8 different sets in total to generate a fully determined system. The actual used G_i sets or *cube indices* can be seen in Table 3.

For three rounds of the PRIMATE-permutation, the polynomial degree of f_i is 4. Therefore, our sets $G_1,\ldots,G_n$ should be of size 3. After experimentation, several manners to generate a fully determined system have been found. These different manners all use the same number of sets. The first manner is to systematically choose sets G_i with all indices in the G_i pointing to the same 5-bit element in the state. After a linear system of rank 20 has been generated with all indices in G_i all pointing to bits of the same 5-bit element, this method can be extended to generate a fully determined linear system. Each set G_i could be chosen such that the indices in G_i point to bits of three different 5-bit elements. This is, however, more tricky to choose correctly, as there are more combinations possible. Just as with the first approach, this approach requires us to perform $\delta_{G_i}(f)$ for 16 sets in total to generate a fully determined system.

For four and five rounds of the PRIMATE-permutation, the sets $G_1,\ldots,G_n$ have been found using experimentation. The actual value of the cube indices can again be found in Table 3.

Table 3

The cube indices that were used to perform a working cube attack on the PRIMATE. The attack was performed for different rounds, which are shown in this table. Cube indices indicates the sets $G = \{G_1,...,G_n\}$ we use to make the linear system generated by $\delta_{G_i}(f)$ fully determined

Rounds	Degree	Cube indices $(G_1,...,G_n)$
2	2	{1}, {5}, {6}, {10}, {11}, {15}, {16}, {20}, {21}, {25}, {26}, {30}, {31}, {35}, {36}, {40}
3	4	{1,2,3}, {2,4,5}, {3,4,5}, {1,2,4}, {6,7,8}, {7,9,10}, {8,9,10}, {6,7,9}, {11,12,13}, {12,14,15}, {13,14,15}, {11,12,14}, {16,17,18}, {17,19,20}, {18,19,20}, {16,17,19}, {21,22,23}, {22,24,25}, {23,24,25}, {21,22,24}, {26,27,28}, {27,29,30}, {28,29,30}, {26,27,29}, {31,32,33}, {32,34,35}, {33,34,35}, {31,32,34}, {36,37,38}, {37,39,40}, {38,39,40}, {36,37,39}
4	8	{1,...,7}, {6,...,12}, {11,...,17}, {16,...,22}, {21,...,27}, {26,...,32}, {31,...,37}, {36,...,40,1,2}
5	16	{1,...,9,11,...,14,16,17}, {6,...,14,16,...,19,21,22}, {11,...,19,21,...,24,26,27}, {16,...,24,26,...,29,31,32}, {21,...,29,31,...,34,36,37}, {26,...,34,36,...,39,1,2}, {31,...,39,1,...,4,6,7}, {36,...,40,1,...,4,6,...,9,11,12}

4. CONCLUSION

We have seen a vulnerability in the GIBBON algorithm of the PRIMATEs family of authenticated encryption algorithms. Furthermore, a proof of concept has been developed, showing how such a vulnerability could be exploited and showing the computational power required to perform such an attack. The GIBBON family uses a permutation that performs a particular round function six times. The vulnerability shown in this permutation was shown to only exist in permutations using the round function at most six times. If the round function would be applied seven times, the vulnerability would not exist. Furthermore, we have seen that this vulnerability can only be exploited if the nonce would be reused. Therefore, the attack is only applicable to implementations that deviate from the specification. However, when the same nonce is reused many times, the attacker can deduce the internal state at the start of the encryption. This internal state can be used to decrypt any message using the same nonce. Furthermore, while only implementations that deviate from the specification are affected, this attack should still be of interest for the designers of PRIMATEs: the attack can be prevented by changing the SubElements step without decreasing the speed of the encryption schemes.

REFERENCES

1. E. ANDREEVA, B. BILGIN, A. BOGDANOV, A. LUYKX, F. MENDEL, B. MENNINK, N. MOUHA, Q. WANG, K. YASUDA, *PRIMATEs v1.1*. Tech. rep., KU Leuven (July 2016).
2. S. BEDI and N.R. PILLAI, *Cube attacks on trivium*. IACR Cryptology ePrint Archive 2009, **15** (2009), https://eprint.iacr.org/2009/015.pdf.
3. G. BERTONI, J. DAEMEN, M. PEETERS and G. VAN ASSCHE, *Duplexing the sponge: Single-passauthenticated encryption and other applications*. In: Miri, A., Vaudenay, S. (eds.), *Selected Areas in Cryptography*: 18th International Workshop (SAC 2011), Toronto, ON, Canada, August 11-12, 2011. Revised Selected Papers. Lecture Notes in Computer Science, **7118**, pp. 320–337, Springer, Berlin, Heidelberg, 2012. http://dx.doi.org/10.1007/ 978-3-642-28496-0_19.
4. E. BIHAM and A. SHAMIR, *Differential cryptanalysis of DES-like cryptosystems*. In: Menezes, A.J.,Vanstone, S.A. (eds.), *Advances in Cryptology-CRYPT0' 90: Proceedings*. Lecture Notes in Computer Science, **537**, pp. 2–21, Springer, Berlin, Heidelberg, 1991, http://dx.doi.org/10.1007/3-540-38424-3_1.
5. B. BILGIN, A. BOGDANOV, M. KNEŽEVIĆ, F. MENDEL and Q. WANG, *Fides: Lightweight authenticatedcipher with side-channel resistance for constrained hardware*. In: Bertoni, G., Coron, J.S. (eds.), *Cryptographic Hardware and Embedded Systems - CHES 2013, 15th International Workshop*, Santa Barbara, CA, USA, August 20-23, 2013, Proceedings. Lecture

Notes in Computer Science, **8086**, pp. 142–158, Springer, Berlin, Heidelberg, 2013, http://dx.doi. org/10.1007/978-3-642-40349-1_9.

6. J. DAEMEN and V. RIJMEN, *The block cipher Rijndael*. In: Quisquater, J.J., Schneier, B. (eds.), *Smart Card Research and Applications: Third International Conference, CARDIS'98*, Louvainla-Neuve, Belgium, September 14-16, 1998, Proceedings. Lecture Notes in Computer Science, **1820**, pp. 277–284, Springer, Berlin, Heidelberg, 2000, http://dx.doi. org/10.1007/10721064_26.
7. I. DINUR, P. MORAWIECKI, J. PIEPRZYK, M. SREBRNY and M. STRAUS, *Cube attacks and cube-attacklike cryptanalysis on the round-reduced Keccak sponge function* (2015), http://eprints.qut. edu.au/101021/.
8. I. DINUR and A. SHAMIR, *Cube attacks on tweakable black box polynomials*. In: Joux, A. (ed.), *Advances in Cryptology - EUROCRYPT 2009: 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Cologne, Germany, April 26-30, 2009. Proceedings. Lecture Notes in Computer Science, **5749**, pp. 278–299, Springer Berlin, Heidelberg, 2009, http://dx.doi.org/10.1007/978-3-642-01001-9_16.
9. I. DINUR and A. SHAMIR, *Breaking Grain-128 with dynamic cube attacks*. In: Joux, A. (ed.), *Fast Software Encryption: 18th International Workshop*, (FSE 2011), Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers, pp. 167–187, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2011, http://dx.doi.org/10.1007/978-3-642-21702-9_10
10. *KAYLukas*: *Kaylukas/PRIMATEs-attack: Cube attack on PRIMATEs* (June 2017), https://doi. org/10.5281/zenodo.803141.
11. X. LAI, *Higher Order Derivatives and Differential Cryptanalysis*, The Springer International Series in Engineering and Computer Science, **276**, chapter Communications and Cryptography, pp. 227–233, Springer US, Boston, MA, 1994, http://dx.doi.org/10.1007/ 978-1-4615-2694-0_23.
12. P. MROCZKOWSKI and J. SZMIDT, *Cube attack on courtois toy cipher*. IACR Cryptology ePrintArchive 2009, 497, 2009.
13. *NIST Computer Security Division: SHA-3 standard: Permutation-based hash and extendableoutput functions*. FIPS Publication 202, National Institute of Standards and Technology, U.S. Department of Commerce, May 2014, http://csrc.nist.gov/publications/drafts/fips-202/ fips_202_draft.pdf.
14. F.M. QUEDENFELD and C. WOLF, *Advanced algebraic attack on trivium*. In: Kotsireas, I.S., Rump,S.M., Yap, C.K. (eds.), *Mathematical Aspects of Computer and Information Sciences: 6th International Conference*, (MACIS 2015), Berlin, Germany, November 11-13, 2015, Revised Selected Papers, pp. 268–282, Lecture Notes in Computer Science, Springer International Publishing, Cham 2016, http://dx.doi.org/10.1007/978-3-319-32859-1_23.
15. M. SEIDLOVÁ, *Algebraic-differential analysis of Keccak*. Master's Thesis, Charles University, 2015.