Interpolation Attacks on Round-Reduced Elephant, Kravatte and Xoofff

HAIBO ZHOU1, RUI ZONG3, XIAOYANG DONG3, KETING JIA∗4 AND WILLI MEIER5

1 Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, China
2 School of Cyber Science and Technology, Shandong University, Qingdao 266237, China
3 Institute for Advanced Study, Tsinghua University, China
4 Department of Computer Science and Technology, Tsinghua University, China
5 FHNW, Switzerland

Email: Corresponding author: ktjia@tsinghua.edu.cn

We introduce an interpolation attack using the Moebius Transform. This can reduce the time complexity to get a linear system of equations for specified intermediate state bits, which is general to cryptanalysis of some ciphers with update function of low algebraic degree. Along this line, we perform an interpolation attack against Elephant-Delirium, a round 2 submission of the ongoing NIST lightweight cryptography project. This is the first third-party cryptanalysis on this cipher. Moreover, we promote the interpolation attack by applying it to the Farfalle pseudo-random constructions Kravatte and Xoofff. Our attacks turn out to be the most efficient method for these ciphers thus far.

Keywords: Interpolation Attack, Moebius Transform, Elephant, Kravatte, Xoofff

1. INTRODUCTION

Authenticated encryption (AE) can provide confidentiality, integrity and authenticity for messages simultaneously. Recently, NIST [1] has initiated a process to solicit, evaluate, and standardize lightweight authenticated encryption algorithms with associated data (AEAD) and hashing algorithms, that are suitable for use in constrained environments where the performance of current NIST cryptographic standards is not acceptable. Until September 10, 2019, there are 32 out of 56 candidates selected to Round 2.

ELEPHANT [2] is a family of lightweight authenticated encryption schemes which has reached the 2nd round of NIST lightweight cryptography project. The mode of ELEPHANT is a nonce-based encrypt-then-MAC construction, where encryption is performed using counter mode and internally uses a cryptographic permutation masked using LFSRs. The mode is permutation-based and only evaluates the permutation in the forward direction. As such, there is no need to implement multiple primitives or the inverse of the primitive. This allows it to rely and build on the sponge-based lightweight hashing. Moreover, ELEPHANT is parallelizable by design, easy to implement due to the use of LFSRs for masking (no need for finite field multiplication). Because of the parallelism property, there is no need to instantiate ELEPHANT with a large permutation. Thus, the original three instantiations all use a state of no more than 200 bits.

KRAVATTE [3] and XOOFF [4] are both based on the Farfalle construction. Farfalle [3] is an efficiently parallelizable permutation based construction of a pseudorandom function (PRF), which is first introduced in ToSC 2018. It takes as input a key and a (sequence of) string(s), and produces an arbitrary-length output. Moreover, when instantiated with a secret key, those output bits look like independent uniformly-drawn random bits. It is efficient because its permutation calls can be performed in parallel as soon as the input masks have been generated. Such a PRF is a powerful primitive that can readily be used as a message authentication code (MAC), a stream cipher or a key derivation function.

Our Contribution. In this paper, we utilize the optimized interpolation attack which was first introduced by Dinur et al. [10], and give a method called improved interpolation attack to analyze the Farfalle constructions. For ELEPHANT-Delirium encryption algorithm, we give a 8 out of 18 rounds attack and this is the first third party attack. For KRAVATTE updated version...
Kratoville Achouffe, our attacks have less time complexities when compared to the attacks of Chaigneau et al. [5] on its simple version Kratoville 6644. Similarly, we apply this method to Xoooff, which is another Farfalle construction cipher. The results of our attacks are summarized in Table 1. Moreover, our improved method is not only used for cryptanalysis of the above three ciphers, but also with a broad applicability in some other ciphers whose intermediate state bit’s ANF can be expressed as a function \( F_K(C) \) with low degrees.

In the next sections, we discuss our attacks in more details. Section 2 gives the notations and the brief description of the ciphers. Section 3 describes methods and tools used in our attack. Section 4 presents the method of our attack and we apply it in Section 5. We conclude in Section 6.

2. PRELIMINARIES

2.1. Notation

Some notational conventions of state are shown in the following.

\[
\begin{align*}
(i, j, k) & \quad \text{index of a bit,} \\
(*, j, k) & \quad \text{index of a row,} \\
(i, *, k) & \quad \text{index of a column,} \\
(i, j, *) & \quad \text{index of a lane,} \\
(*, j, *) & \quad \text{index of a plane,} \\
A_{i,j,k} & \quad \text{the bit indexed by } (i, j, k) \text{ of state } A, \\
A_{i,j} & \quad \text{the lane indexed by } (i, j, *) \text{ of state } A, \\
A_j & \quad \text{the plane indexed by } (*, j, *) \text{ of state } A, \\
S_i & \quad \text{a plaintext subspace,} \\
C_{S_i} & \quad \text{the ciphertext subspace obtained by encrypting the plaintexts of } S_i, \\
P_S & \quad \text{the plaintext superstructure,} \\
C_S & \quad \text{the ciphertext superstructure obtained by encrypting the plaintexts of } P_S.
\end{align*}
\]

2.2. The Keccak-p permutation

The permutation Keccak-p is derived from Keccak-f [6] with a variable number of rounds, which is mainly defined by two parameters: the width \( b = 25 \times 2^l \) and the number of rounds \( n_r \), where \( b \in \{25, 50, 100, 200, 400, 800, 1600\} \). The permutation Keccak-p is denoted as Keccak-p\([b, n_r]\]. The round function of Keccak-p consists of five operations, denoted as \( \circ \), \( \circ \circ \), \( \circ \circ \circ \circ \circ \circ \), and the details are as follows:

\[
\begin{align*}
\theta : A_{x,y,z} & = A_{x,y,z} + \sum_{j=0}^{l} (A_{x-1,j,z} + (A_{x+1,j} \ll 1)). \\
\rho : A_{x,y} & = A_{x,y} \lll \rho[x, y]. \\
\pi : A_{y,2z+3y} & = A_{x,y}. \\
\chi : A_{x,y,z} & = A_{x,y,z} + ((\neg A_{x+1,y} + 1) \wedge A_{x+2,y}). \\
t : A_{0,0} & = A_{0,0} + RC.
\end{align*}
\]

Here, we use \( A \) to represent the state of the permutation Keccak-p with the size of \( b \) bits, which is also expressed by \( 5 \times 5 \) bit lanes, as shown in Fig.

\[
\begin{align*}
\theta : A_{x,y,z} & = A_{x,y,z} + \sum_{j=0}^{l} (A_{x-1,j,z} + (A_{x+1,j} \ll 1)). \\
\rho : A_{x,y} & = A_{x,y} \lll \rho[x, y]. \\
\pi : A_{y,2z+3y} & = A_{x,y}. \\
\chi : A_{x,y,z} & = A_{x,y,z} + ((\neg A_{x+1,y} + 1) \wedge A_{x+2,y}). \\
t : A_{0,0} & = A_{0,0} + RC.
\end{align*}
\]

1. The lane is denoted by \( A_{i,j} \) with \( i \) for the column index and \( j \) for the row index, where \( i \) and \( j \) are in the set \([0,1,2,3,4]\) and they are working modulo 5 without other specification.

2.3. The Xoodoo permutation

Daemen et al. [4] introduced a 384-bit permutation Xoodoo at ToSC 2019, which is similar to Keccak-p. We use the three-dimensional matrix \( A[4][5][32] \) as the state. The round function of Xoodoo includes five operations, denoted as \( R = \rho_{west} \circ \chi \circ \circ \circ \circ \circ \circ \circ \circ \theta \), which are listed in the following.

\[
\begin{align*}
\theta & : A_{x,y,z} = A_{x,y,z} + \sum_{j=0}^{l} (A_{x-1,j,z} + A_{x-1,j,z-4}). \\
\rho_{west} & = A_{x,1,z} = A_{x-1,1,z}, A_{x,2,z} = A_{x,2,z-11}. \\
t & = A_{0,0} = A_{0,0} + RC. \\
\chi & = A_{x,y,z} = A_{x,y,z} + (A_{x,y,z} + 1) \wedge A_{x,y,z} + 22z. \\
\rho_{east} & = A_{x,1,z} = A_{x,1,z-1}, A_{x,2,z} = A_{x-2,2,z}. \\
\end{align*}
\]

Xoodoo could be applied as an AE scheme in Ketje style, which is shown in [7].

2.4. Elephant

ELEPHANT is a family of lightweight AE schemes, which has been submitted to the NIST lightweight cryptography project [1], and it has reached the second round. As a lightweight cryptographic algorithm, it has many advantages in practical applications. The underlying mode is permutation-based and inverse-free with a small state size, and allows for a high degree of parallelism. The authors provide three instantiations Dumbo, Jumbo and Delirium, which use different round functions. But in this paper, we only discuss the ELEPHANT-Delirium encryption scheme which uses a 18-round Keccak-f[200] permutation.
TABLE 1. Summary of Key-recovery Attacks

<table>
<thead>
<tr>
<th>Target</th>
<th>Rounds</th>
<th>(Top.)</th>
<th>M(bit)</th>
<th>D(block)</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELEPHANT-DELIRIUM</td>
<td>8/18</td>
<td>$2^{93.3}$</td>
<td>$2^{10}$</td>
<td>$2^{10}$</td>
<td>5.1</td>
</tr>
<tr>
<td>KRAVATTE 6644</td>
<td>4/4</td>
<td>$2^{17.2}$</td>
<td>$2^{27.2}$</td>
<td>$2^{7.3}$</td>
<td>[5]</td>
</tr>
<tr>
<td>KRAVATTE ACHOUFFE</td>
<td>4/6</td>
<td>$2^{106.2}$</td>
<td>272</td>
<td>$2^{78.3}$</td>
<td>5.2</td>
</tr>
<tr>
<td>XOOFF</td>
<td>4/6</td>
<td>$2^{90.4}$</td>
<td>$2^{68}$</td>
<td>$2^{74.2}$</td>
<td>5.3.2</td>
</tr>
<tr>
<td></td>
<td>6/6</td>
<td>$2^{20.6}$</td>
<td>$2^{68}$</td>
<td>$2^{74.2}$</td>
<td>5.3.3</td>
</tr>
</tbody>
</table>

1: KRAVATTE 6644 uses linear rolling function.

FIGURE 2. Encryption of ELEPHANT [2]

The generic ELEPHANT encryption mode is presented in Fig. 2. For ELEPHANT-DELIRIUM, the state size $n$ is 200 bits, the size of the master key is 128 bits and the size of the nonce $N$ is 96 bits. And the mask$_K$ is generated by the function mask. Moreover, the mask$_K$ is independent of nonce value, and hence does not affect the dimension of the input space.

2.5. Kravatte and Xoooff

KRAVATTE and XOOFF are both Farfalle pseudo-random constructions. We give an introduction about Farfalle in the following.

First of all, Farfalle is permutation-based with variable input and output length. Although the input and output lengths are tunable, inside the construction, strings of bits are processed in chunks of $b$ bits, where $b$ is the size of the underlying permutation. The Farfalle construction can be used to build a pseudo-random function from parallel applications of fixed permutations, and returns a string of arbitrary blocks of output.

The Farfalle construction is divided into three parts: mask derivation, compression layer and expansion layer. More specifically, it includes four cryptographic permutations (possibly identical or related), and each of them operates on a $b$-bit block and they are used as follows:

- $p_b$ derive the initial mask from the master key,
- $p_c$ used in the compression layer,
- $p_d$ used between the compression and expansion layer,
- $p_e$ used in the expansion layer.

Besides these four permutation functions, its instantiation requires the definition of two so-called rolling functions, represented by $\odot$ in Fig. 3, operating on a $b$-bit block. They are denoted by $roll_c$, $roll_e$ and applied as follows:

- $roll_c$ for generating masks added to the input blocks in the compression layer,
- $roll_e$ to update the internal state during the expansion layer.

$roll_i(k)$ represents the result after applying the rolling function $i$ times. In particular, $roll_0(k)$ means you do not apply the rolling function on it.

The Farfalle construction takes a master key $K$ and a message $M$ as input. The details are listed as below:

Mask derivation This layer takes the padded master key $K$ as input and generates $k^m$ by $p_b$, $k^n = p_b(K || 10^s)$. Then $roll_c$ updates $k^m$ every time to get the masks $k_{0}^{m-1}$. Specially, $k_{0}^{m-1} = k^n$. Besides, $k^{out} = roll_c^1(k^n)$.

Compression layer This layer takes the padded message as input. Firstly, $M || 10^s$ is padded into $t_i$ blocks $m_i$. Then the permutation $p_c$ takes $m_i + k_{i}^m$ as input. Finally, XOR all the results after $p_c$ together and get the $a$-bit block accumulator value: $Acc(M) = \sum_{i=0}^{l-1} p_c(m_i + k_{i}^m)$.

Expansion layer This layer takes the result of compression layer as input. Firstly, apply the permutation $p_d$ on the accumulator result to get $y = p_d(Acc(M))$. Secondly, apply $roll_e$ on $y$ and get the $b$-bit block accumulator output: $z^j = p_e(roll_e^j(y)) + k^{out}$ for $j = 0, ..., l_o - 1$.

2.5.1. Kravatte

In this part, we introduce the Farfalle original instantiation KRAVATTE ACHOUFFE which is based on the permutation Keccak-p[1600, 6]. The Farfalle and Kravatte are both designed by Bertoni et al. [3].

**Definition 2.1.** (Kravatte ACHOUFFE [3]) Kravatte ACHOUFFE is Farfalle[$p_b, p_c, p_d, p_e, roll_c, roll_e$] with the following parameters:

- $p_b = p_c = p_d = p_e = Keccak-p[1600, n_r = 6]$,
- $roll_c$ as specified below,
- $roll_e$ as specified below.

The rolling function $roll_c$ applies a linear transfor-
The Farfalle construction [3]

The input mask rolling function roll\textsubscript{c} updates a state A in the following way:

\begin{align*}
B_{3,0} & \leftarrow A_{0,0} + (A_{0,0} \ll 13) + (A_{0,1} \ll 3), \\
B_{x,0} & \leftarrow A_{x+1,0} \quad \forall x \neq 3, \\
A_{0} & \leftarrow A_{1}, \\
A_{1} & \leftarrow A_{2}, \\
A_{2} & \leftarrow B_{0}.
\end{align*}

Note that \( B_{0} \) is an auxiliary variable that has the shape of a plane, and \( B_{x,0} \) is the lane indexed by \((x,0,*)\) of plane \( B_{0} \).

The state rolling function roll\textsubscript{Xooff} updates a state A in the following way:

\begin{align*}
B_{3,0} & \leftarrow A_{0,0} \cdot A_{0,2} + (A_{0,0} \ll 5) + (A_{0,1} \ll 13) + 0x00000007, \\
B_{x,0} & \leftarrow A_{x+1,0} \quad \forall x \neq 3, \\
A_{0} & \leftarrow A_{1}, \\
A_{1} & \leftarrow A_{2}, \\
A_{2} & \leftarrow B_{0}.
\end{align*}

3. RELATED WORK

3.1. Interpolation Attacks

The interpolation attack was first introduced by Jakobsen and Knudsen on block ciphers with low algebraic degree in 1997 [8], which is related to high-order differential cryptanalysis proposed by Lai [9].

The interpolation attack considers the intermediate target bit \( a \), whose ANF can be represented by the ciphertext \( C \) and the secret key \( K \), i.e. \( a = F_{K}(C) \), as shown in Equation (1).

\[
F_{K}(C) = F_{K}(c_{1},...,c_{n}) = \sum_{u=(u_{1},...,u_{n}) \in GF(2^{n})} \alpha_{u}M_{u},
\]

where \( \alpha_{u} \in \{0, 1\} \) is the coefficient of monomial \( M_{u} =
\[ \prod_{i=1}^{n} c_i^{u_i}. \] Denote the number of non-zero \( \alpha_u \) as \( N_{\alpha_u} \). Then, we want to recover the coefficients \( \alpha_u \), which only depend on the secret key bits. Suppose the algebraic degree of \( F_K(C) \) is less than \( d \). To deduce the coefficients of \( F_K(C) \), we regard the coefficients as the variables, and recover them by solving a linear system of equations. We use the chosen plaintext interpolation attack over \( GF(2) \). Since \( \deg(F_K(C)) \leq d \), the sum of \( a \) over a \((d+1)\)-dimension plaintext subspace \( S_i \) is zero. That is to say, the sum of the values of the polynomial \( F_K(C) \) over ciphertexts \( C_S = \{C_1, ..., C_{2^{d+1}}\} \) (obtained by encrypting the plaintext subspace \( S_i \)) is zero. As the secret key is an unknown constant, we express this result as Equation (2).

\[
\sum_{i=1}^{2^{d+1}} F_K(C_i) = \sum_{u=(u_1, ..., u_n)\in GF(2^n)} \alpha_u \times \left( \sum_{C \in C_{S_i}} M_u \right) = 0
\]

(2)

This is a linear equation with the coefficients \( \alpha_u \), and \( \alpha_u \) are functions of secret key bits. The subspace \( S_i \) provides one linear equation. We can construct more such subspaces to get \( N_{\alpha_u} \) linear independent equations to recover the secret keys. That needs about \( N_{\alpha_u} \times N_{\alpha_u} \times 2^{d+1} \) XOR operations.

Moreover, we know from the high-order difference cryptanalysis that we can get the value of \( \sum_{C \in C_{S_i}} M_u \) in Equation (2) without any loss if we had the expression of \( M_u \) as a function of the plaintext, \( F(P) \).

### 3.2. Moebius Transform

Firstly, we give a brief definition of Moebius Transform. For more details, please refer to [11].

**Definition 3.1.** The Moebius Transform is a classic algorithm that transforms the truth table of function \( F \) to its ANF efficiently.

We give a simple example to show the Moebius Transform. The input variables of the function \( F: GF(2^3) \rightarrow GF(2) \) are \( x_1, x_2, x_3 \). The truth table of \( F \) is shown in the Table 2.

<table>
<thead>
<tr>
<th>Monomial</th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( x_1 )</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( x_1x_2 )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( x_3 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( x_2x_3 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( x_3x_5 )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( x_1x_2x_3 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 2. Truth Table**

\[ \sum_{i=1}^{2^{d+1}} F_K(C_i) = \sum_{u=(u_1, ..., u_n)\in GF(2^n)} \alpha_u \times \left( \sum_{C \in C_{S_i}} M_u \right) = 0 \]

\[ F(x_1, x_2, x_3) = 1 + x_1 + x_2 + x_3 + x_1x_3 \]

It is obvious that the time complexity of the Moebius Transform is \( \frac{1}{2} \log_2 n \) XOR operations, where \( n \) is the length of the column Output.

Moreover, we use the Moebius Transform to get \( M_u \)’s plaintext expression: \( F(P) \).

For an arbitrary \( M_u = \prod_{i=1}^{n} c_i^{u_i} \), each \( c_i \) can be expressed in plaintext bits \( p \). If we get each \( c_i \)'s ANF and calculate the result, we can get the \( F(P) \) of \( M_u \), i.e. \( M_u = F(P) = \sum_{v=(v_1, ..., v_n)\in GF(2^n)} \beta_v X_v \), where \( \beta_v \in \{0, 1\} \) is the coefficient of the monomial \( X_v = \prod_{i=1}^{n} p_i^{v_i} \).

However, ciphers’ iteration are usually too complicated to calculate the expression. But it is easy to get the values of the truth table of \( M_u \) because \( M_u = \prod_{i=1}^{n} c_i^{u_i} \), and the values of \( c_i \) are known. Thus, the Moebius Transform can be used to get the \( F(P) \) of \( M_u \).

### 3.3. Optimized Interpolation Attack

In this part, we introduce the idea of the optimized interpolation attack which was first introduced by Dinur et al. [10]. It can reduce the time complexity using the Moebius Transform.

The Equation (2) shows that one \((d+1)\)-dimension subspace gives one equation on \( \alpha_u \). Summing over \( N_{\alpha_u} \) such subspaces will get \( N_{\alpha_u} \) linear independent equations and recover the secret keys. This can be done efficiently using the Moebius Transform. As shown in [10], we can view the input as evaluating a \((d+1+c)\)-variable polynomial over \( GF(2) \), and the summation over a \((d+1)\)-dimension subspace is
equal to the coefficient of its corresponding \((d + 1)\)-degree monomial which is called maxterm. Thus, the optimized interpolation attack uses the MOEBIUS Transform to build the equation system and the complexity of this process is about \((d + 1 + e) \cdot 2^{d+e}\) XOR operations.

### 3.4. Construction of Affine Spaces in the Accumulator

In this paper, we mainly focus on the algorithms with the Farfalle construction, which has \(l_i\) input blocks and \(l_o\) output blocks. In order to apply the MOEBIUS Transform to such structures, we need to construct an affine space of dimension \(n\) in the accumulator block. We use a property of Farfalle, which is already identified in [3].

**Property 1.** Given an \(n\)-block padded message \(M = (m_0, ..., m_{n-1})\), let the associated accumulator value be \(\text{Acc}(M) = \sum_{i=0}^{n-1} p_e(m_i + k_i^{(n)})\). Let \(M^0 = (m_0^0, ..., m_{n-1}^0)\) and \(M^1 = (m_0^1, ..., m_{n-1}^1)\) denote an arbitrary pair of padded messages such that \(m_i^0 \neq m_i^1\) for all \(i\). These messages can build the structure of \(2^n\) \(n\)-block input messages \(S = \{(m_0^0, ..., m_{n-1}^0), (\epsilon_0, ..., \epsilon_{n-1}) \in \{0, 1\}^n\}\). Denote the one-block difference by \(\delta_i = p_e(m_i^0 + k_i^{(n)}) + p_e(m_i^1 + k_i^{(n)})\). If \(n \ll b\) (\(b\) is the state size), the \(\delta_i\) are linearly independent with overwhelming probability. Then \(\text{Acc}(S)\) is the \(n\)-dimensional affine subspace \(\text{Acc}(M_0) + (\delta_0, ..., \delta_{n-1})\).

Based on Property 1, we can easily build structures of \(2^n\) \(n\)-block messages that are transformed by the compression layer into an affine space of one-block accumulator values of dimension \(n\). Moreover, there are only two possible values \(m_i^0\) or \(m_i^1\) for each input block. Thus, we give a new definition.

**Definition 3.2.** For \(2^n\) \(n\)-block input messages \(S = \{(m_0^0, ..., m_{n-1}^0), (\epsilon_0, ..., \epsilon_{n-1}) \in \{0, 1\}^n\}\), we use a bit 0 (resp. 1) to represent the value of the block \(m_i^0\) (resp. \(m_i^1\)). So we define the vector \(P_S = (\epsilon_0, ..., \epsilon_{n-1}) \in \{0, 1\}^n\) to represent the input structure \(S\).

It is obvious that Property 1 does not depend on the number of rounds in \(p_c\). Hence, we can regard the compression layer as a affine transformation and the dimension of the affine space \(\text{Acc}(P_S)\) is equal to \(P_S\). It helps us to apply the MOEBIUS Transform on the Farfalle constructions.

### 3.5. Properties of Keccak-\(p\)

In [5], Chaigneau et al. introduce a property of Keccak-\(p^{-1}\) to reduce the variables of its output. It suits for that each input variable of Keccak-\(p^{-1}\) is \(A \oplus k^\text{out}\). It should be pointed out that the structure of Keccak-\(p^{-1}\) is public, so we don’t need a decryption oracle. Suppose the state size is \(b\) and the notations are as shown in Fig. 5.

**FIGURE 5.** Some Notations of Keccak-\(p^{-1}\)

The inverse Sbox in \(\chi^{-1}\) layer has algebraic degree three, and the input of the \(\chi^{-1}\) is \(B = A + k^\text{out}\), where ‘+’ stands for ‘\(\oplus\)’ in the following. The output bit of the inverse Sbox can be expressed as below:

\[
C_x = B_{x+1}B_{x+2}B_{x+4} + B_{x+1}B_{x+2}B_x = (k_{x+1}^\text{out} + A_{x+1} + 1)(k_{x+3}^\text{out} + A_{x+3} + 1)(k_{x+4}^\text{out} + A_{x+4} + 1) + (k_{x+1}^\text{out} + A_{x+1} + 1)(k_{x+2}^\text{out} + A_{x+2} + k_{x+2}^\text{out} + A_x)
\]

By introducing the new variables \(w_x = k_{x+1}^\text{out}k_{x+3}^\text{out} + k_{x+1}^\text{out}k_{x+2}^\text{out} + k_{x+2}^\text{out} + u_x\), \(u_x = k_{x+4}^\text{out}k_{x+3}^\text{out} + k_{x+3}^\text{out}k_{x+2}^\text{out}\), and \(v_x = k_{x+4}^\text{out}k_{x+2}^\text{out} + C_x\) can be rewritten as \(w_x + P_x(A)\), where \(P_x(A)\) is an affine combination of \(u_x\), \(u_{x+1}\), \(v_x\), \(k_{x+1}^\text{out}\), \(k_{x+2}^\text{out}\), with coefficients determined by \(A\).

**Property 2.** (Property of \(\chi^{-1}\) in Keccak-\(p^{-1}\))

For each Sbox in \(\chi^{-1}\) with input \((k_{x}, ..., k_{x+4}) \oplus (A_x, ..., A_{x+4})\), there are 15 new variables in key bits for the output expressions: 10 variables \(u, v\) of algebraic degree two and 5 variables \(w\) of algebraic degree three.

The Sbox in \(\chi^{-1}\) generates new variables and the inverse affine layer spreads them to almost every bit. First of all, we use \(\sigma(x, y, z)\) to denote the state bit moved to \((x, y, z)\) by the permutation \((x \circ \rho)^{-1}\), i.e. \(D_{x,y,z} = C_{\sigma(x,y,z)}\). Moreover, the high diffusion layer \(\theta^{-1}\) has the following property.

For each column \(D_{x,y,z}\) of the state, there is a set of bit positions \(S_{x,y,z}\) such that each bit after \(\theta^{-1}\) is given in the following:

\[
E_{x,y,z} = D_{x,y,z} + \sum_{(x',y',z') \in S_{x,y,z}} D_{x',y',z'} = C_{\sigma(x,y,z)} + \sum_{(x',y',z') \in S_{x,y,z}} C_{\sigma(x',y',z')}
\]

\[
= w_{x,y,z} + P_{\sigma(x,y,z)}(A) + \sum_{(x',y',z') \in S_{x,y,z}} (w_{x',y',z'} + P_{\sigma(x',y',z')}(A))
\]

\[
= w_{x,y,z}' + P_{\sigma(x,y,z)}(A) + Q_{x,z}(A)
\]

where \(w_{x,y,z}'\) is a new variable defined as the linear combination of all the \(w\) variables involved in the expression of \(E_{x,y,z}, Q_{x,z}(A)\) is the sum of the \(P\) over position set \(S_{x,y,z}\). Because there are 6 variables for \(P_{\sigma(x,y,z)}(A)\), we deduce that the output of \(\theta^{-1}\) can be expressed as

\[
E_{x,y,z} = P_{x,y,z}(A) + Q_{x,z}(A),
\]
and \( P'_{x,y,z}(A) \) contains 7 potentially nonzero variables at most, where \( P'_{x,y,z}(A) = w_{x,y,z} + P_{x,y,z}(A) \).

In a conclusion, we get the following property:

**Property 3. (Property of Keccak-p⁻¹)**

Let the input of Keccak⁻¹ be \( A + k_{out} \), then the output can be expressed as \( E_{x,y,z} = P'_{x,y,z}(A) + Q_{x,z}(A) \). Moreover \( P'_{x,y,z} \) has 7 potentially nonzero variables at most; \( Q_{x,z} \) is considered as a linear function of all the 3b variables \( (u, v, \text{ and } k_{out}) \).

### 3.6 Properties of Xoodoo

For Xoodoo⁻¹, we give some properties which are suitable for the situations when the input is \( B = A + k_{out} \). The notations are shown in Fig. 6.

**FIGURE 6. Some Notations of Xoodoo⁻¹**

Similarly, we investigate the Sbox in \( \chi⁻¹ \) layer. The algebraic degree of the Sbox is two, and the input of \( \chi⁻¹ \) is \( B_x = A_x + k_{out} \). The output bit of the Sbox can be expressed as below:

\[
C_x = B_{x+1}B_{x+2} + B_x = (k_{out} + A_{x+1} + 1)(k_{out} + A_{x+2}) + (k_{out} + A_x).
\]

Here we introduce the new variables \( w_x = k_{x+1}k_{out} + k_{out}^2 \), so that \( C_x \) can be written as \( C_x = w_x + k_{out}A_{x+1} + k_{out}^2(A_{x+1} + 1) + (A_{x+1} + 1)(A_{x+2} + A_x) \).

It is a linear equation in \( w_x, k_{out}, \text{ and } k_{out}^2 \).

**Property 4. (Property of \( \chi⁻¹ \) in Xoodoo⁻¹)**

For each Sbox in \( \chi⁻¹ \) with input \( (k_x, k_{x+1}, k_{x+2}) \oplus (A_x, A_{x+1}, A_{x+2}) \), there are 3 new variables in key bits in the corresponding output expression.

The state size of Xoodoo is 384, and there are potentially 384 \( w \) and 384 \( k_{out} \) variables after \( \chi⁻¹ \). The analysis of the affine layer is similar to that of Keccak⁻¹, so we get the following property.

**Property 5. (Property of Xoodoo⁻¹)**

Let the input of Xoodoo⁻¹ be \( A + k_{out} \), then the output of \( \theta⁻¹ \) can be expressed as \( E_{x,y,z} = w_{x,y,z} + Q_{x,z}(A) \), where \( w_{x,y,z} \) is considered as a new variable containing all the \( w \) involved in \( E_{x,y,z} \) and \( Q_{x,z}(A) \) is linear in all the 384 \( k_{out} \) bits. Obviously, \( w_{x,y,z} \) is independent on the state \( A \). In total, \( E_{x,y,z} \) contains 385 nonzero variables.

Notice that there is a rotation layer \( \rho_{est}⁻¹ \) after \( \theta⁻¹ \), i.e. \( F = \rho_{est}⁻¹(E) \). It changes the properties of the \( \theta⁻¹ \) but does not generate new variables.

### 4. The Interpolation Attack

In this part, we introduce how to extend the optimized interpolation attack to the Farfalle constructions.

At first, we give a method to reduce the size of the equation system. We reconstruct \( F_K(C) \) as \( F'_{K}(C) \) basing on Chaigneau et al.'s work [5] to reduce the variables of \( F_K(C) \).

We give an example to explain it. For the equation \( F_K(C) = (k_0 + k_{12})c_0 + (k_0 + k_{12})c_0c_2, \) we rebuild it as \( F'_K(C) = k'_0 \), where \( k'_0 = k_0 + k_{12} \) and \( c'_0 = c_0 + c_0c_2 \). It is obvious that \( N_{eq} < N_{eq} \). Hence it needs fewer equations to get the unknown key bits.

We have rebuilt the expression as Equation (3)

\[
a = F'_{K}(C) = \sum_{t=1}^{N_{eq}} k'_t c'_t \tag{3}
\]

where \( k'_t \in G(2) \) is the equivalent key, \( c'_t \in G(2) \) is the sum of some \( M_u \) which are multiplied by \( k'_t \), and \( N_{eq} \) is the quantity of \( k'_t \). \( c'_t \) is not any monomial, and we call it sum-monomial in the following. \( F_K(C) \) and \( F'_K(C) \) are essentially the same polynomial, and Equation (2) still holds for \( F'_K(C) \).

We choose an example to explain it. For the equation \( F_K(C) = (k_0 + k_{12})c_0 + (k_0 + k_{12})c_0c_2, \) we rebuild it as \( F'_K(C) = k'_0 \), where \( k'_0 = k_0 + k_{12} \) and \( c'_0 = c_0 + c_0c_2 \). It is obvious that \( N_{eq} < N_{eq} \). Hence it needs fewer equations to get the unknown key bits.

We mainly utilize some ideas of Chaigneau et al. in [5] above. Then we combine them with the ideas in the optimised interpolation attack to improve the results.

In order to recover the secret key bit \( k_e \), we need \( N_{eq} \) linear independent equations. So we choose \( N_{eq} \) different \((d + 1)\)-dimension input subspaces \( \{S_i\} \) from a \((d + 1 + e)\)-dimension input super structure. By Definition 3.2, we use \( P_S = (e_0, ..., e_{n-1}) \in \{0, 1\}^n \) to denote the \( 2^n \) \( n \)-block input structure \( S \). If the dimension of the input structure \( P_S \) is \( n \), we will get a \( n \)-dimension space after \( \text{Acc} \).

We mainly utilize some ideas of Chaigneau et al. in [5] above. Then we combine them with the ideas in the optimised interpolation attack to improve the results.

In order to recover the secret key bit \( k_e \), we need \( N_{eq} \) linear independent equations. So we choose \( N_{eq} \) different \((d + 1)\)-dimension input subspaces \( \{S_i\} \) from a \((d + 1 + e)\)-dimension input super structure. By Definition 3.2, \( P_S \) represents the \((d + 1 + e)\)-dimension input super structure and \( C_S \) denotes the corresponding ciphertext space. Each ciphertext subspace \( C_{S_i} \) gives one equation as follows

\[
\sum_{C \in C_{S_i}} F'^\prime_K(C) = \sum_{t=1}^{N_{eq}} k'_t \left( \sum_{C \in C_{S_i}} c'_t \right) = 0, \tag{4}
\]

so the \( \epsilon \) has to satisfy \( \frac{d + 1 + e}{d + 1} \geq N_{eq} \).

Thus, the steps of setting up equation system are similar with the optimized interpolation attack as follows.

1. For a \((d + 1 + e)\)-dimension plaintext super structure \( P_S \), query the encryption oracle to get the corresponding ciphertexts \( C_{S_i} \). Initialize a
$N_{eq} \times N_{eq}$ matrix, in which each row represents a linear equation.

2. Update the matrix column by column. For each $c'_i$ of the equation (3):
   - Evaluate $c'_i$ for all the $2^{d+1+e}$ ciphertexts and get a bit vector.
   - Apply the Moebius Transform on this bit vector to get the $F(P)$ of $c'_i$. Extract the coefficients of master terms of the input subspaces $\{S'_i\}$ from $F(P)$ and copy them to a column of the matrix.

3. The matrix obtained above represents the coefficients of $k'_i$ in the $N_{eq}$ linear independent equations. Solve the system of linear equations to get all the $k'_i$ and derive the key bits.

We need $2^{d+1+e}$ input messages and encryptions in Step 1. Step 2 needs $N_{eq} \times 2^{d+e} \times (d+1+e)$ XOR operations and $2^{d+1+e}$ bits memory to store the bit vector. It also needs $N_{eq}^2$ bits memory to store the matrix, but it is usually smaller. The time complexity of Step 3 is about $N_{eq}^3 / log N_{eq}$ XOR operations [13].

In Sect. 3.1, the traditional interpolation attack needs $N_{eq} \times N_{eq} \times 2^{d+1}$ XOR operations, but for our method, it needs about $N_{eq} \times 2^{d+e} \times (d+1+e)$ XOR operations. If we set $N_{eq} = N_{eq}$, $2^{e} \times (d+1+e)$ is smaller than $N_{eq} \times 2$ in our attack, where $e$ is usually a small number.

5. APPLICATIONS AGAINST ELEPHANT, KRAVATTE AND XOOFFF

5.1. Interpolation Attack against Elephant

In this section, based on the optimized interpolation attack [10], we give an attack against the ELEPHANT-Delirium in nonce-respecting setting.

The ELEPHANT-Delirium uses the Keccak sponge permutation. The state size is 200-bit and the algebraic degree of one round is 2. Moreover, the nonce occupies 96 of 200 bits and the degree after 6 rounds is not more than 64.

To analyze 8-round ELEPHANT-Delirium by interpolation attack, the main work is to linearize the last two rounds and get the ANF of the intermediate target bit $a$ which is also the output of the 6 rounds Keccak-K-p. Furthermore, we have to minimize the number of variables in the ANF of $a$. Obviously, fewer variables require fewer linear equations and lead to lower complexity.

5.1.1. Linearize the Keccak-p-2

We use $A$ to denote $M@C$ in the ELEPHANT-Delirium seen in Fig. 2, and there are 40 Sboxes in the $\chi^{-1}$. Prop. 3 shows that the output of Keccak-p-1 can be expressed as $E_{x,y,z} = P'_{x,y,z}(A) + Q_{x,y,z}(A)$, where $Q_{x,y,z}(A)$ potentially contains $3 \times 200 = 600$ variables ($u, v$ and $k^{out}$) in this place.

As for the second $\chi^{-1}$ layer, we denote its output as $F = \chi^{-1}(E)$. The $F_{x,z}$ omitting index $z$ can be expressed as:

$$F_{0,*} = (P'_{x,z} + Q_1)(P'_{z,*} + Q_3)(P'_{x,*} + Q_4) + (P'_{x,*} + Q_1)(P'_{z,*} + Q_2) + (P'_{x,*} + Q_0)$$

It is easy to see that the item $Q_1Q_2Q_4$ generates most of the variables. And $F_{0,0} + F_{0,1}$ will cancel it out as well as the items $Q_1Q_2$ and $Q_4$. Specially, if we cancel out the $Q_1Q_2Q_4$, there will not be cubic items.

$$F_{0,0} + F_{0,1} = (P'_{1,0} + P'_{1,1})Q_0Q_4 + (P'_{3,0} + P'_{3,1})Q_1Q_4 + (P'_{4,0} + P'_{4,1})Q_1Q_3 + (P'_{1,0}P^*_{4,0} + P'_{1,1}P^*_{3,1})Q_4 + (P'_{3,0}P^*_{4,0} + P'_{3,1}P^*_{4,1})Q_3 + (P'_{1,0} + P'_{1,1})Q_2 + (P'_{2,0} + P'_{3,0}P^*_{2,0} + P'_{2,1} + P'_{3,1}P^*_{2,1})Q_1 + P'_{0,0} + P'_{0,1}P^*_{0,0} + P'_{0,1}P^*_{0,0} + P'_{0,1} + P'_{1,0}P^*_{1,0} + P'_{1,1}P^*_{3,1,4,1}.$$ 

All the $Q$ polynomials are affine combinations of the same set of the 600 variables, and each $P'$ polynomial is an affine combination of 7 variables. Taking into account the constant coefficients of these polynomials, the number of variables required to linearize the expression of $F_{0,0} + F_{0,1}$ is:

$$3 \times 2 \times 8 \times \left(\frac{601}{2}\right) + (3 \times 2 \times 8^2 + 2 \times 2 \times 8) \times 601 + 2 \times (8 + 8^2 + 8^3).$$

That gives approximately $2^{23.1}$ variables (i.e. $k'_i c'_i$).

Moreover, we have run experiments to support the conclusion $N_{eq} < N_{eq}$ in Sect. 4. For this application, the traditional method gives approximately $2^{25.1}$ variables which is bigger than $2^{23.1}$. The test code is given in:

https://github.com/alicebobb/testcode.

5.1.2. The Complexity of the Attack

Since we have got all the variables of the expression, we set up the linear equation system and solve it to get the key.

Firstly, construct the input messages structure. The expression has $2^{23.1}$ variables. Thus setting up $2^{23.1}$ linear independent equations can recover the $k'_i$. One 65-dimension subspace gives one linear independent equation, and $(65+5) > 2^{23.1}$, so we choose a 70-dimension input structure $P_5$. Query the encryption oracle to get the corresponding ciphertext space.

Secondly, construct the linear equation system and solve it. We have shown the process in Sect. 4, and in this attack, the number of linear equation is $N_{eq} = 2^{23.1}$ and the dimension of input structure is 70. So we can get that the data complexity is $2^{70}$ blocks, the memory complexity, which is mainly used to store the Moebius Transform bit vector, is $2^{70}$ and the time complexity is about $2^{98.3}$ XOR operations.

The Computer Journal, Vol. ??, No. ??, ????
5.2. Interpolation Attack against Kravatte Achouffe

In this section, we show how to perform the interpolation attack on a Farfalle cipher Kravatte Achouffe like shown in Sect. 4.

The properties of Farfalle in Sect. 3.4 show that no matter how many rounds \( p \) may have, we can get the \( n \)-dimension affine space after \( \text{Acc} \) by appropriately choosing messages. Thus, the influence of compression layer can be omitted in the attack, i.e. \text{KRAVATTE-}\{n_d, n_e\} represents \text{KRAVATTE-}\{n_c, n_d, n_e\}.

Firstly, we analyze the functions after the \( \text{Acc} \), i.e. \( p_d, \text{roll}_e \) and \( p_c \). Among them, \( p_d \) and \( p_c \) are both Keccak-p permutations. \( \text{roll}_e \) is a non-linear rolling function, but we only use the first limb (i.e. \( z_0 \) in Fig. 3) as output, which is not rolled and the effect of \( \text{roll}_e \) can be omitted. The function after the \( \text{Acc} \) can be seen as an Keccak-p permutation.

 Secondly, the function before the \( \text{Acc} \) is the compression layer. Using the method in Sect. 4, we can construct a \( n \)-dimension space after \( \text{Acc} \).

5.2.1. Analysis of the Attack against Kravatte-\{6,4,4\}

For \text{KRAVATTE-}\{6,4,4\}, we build a \( n \)-dimension space after \( \text{Acc} \), and the items in this space are regarded as inputs of 8 rounds Keccak-p permutation. Then, we analyze it as Sect. 4.

The first step is to linearize and minimize the number of variables for the Keccak-p\(^{-2}(k^{out}+c)\). Chaigneau et al. have given the algebraic expression of Keccak-p\(^{-2}(k^{out}+c)\) of Kravatte in their paper [5], and there are \( 2^{29} \) variables in the expression \( F_{0,0} + F_{0,1} \).

The second step is to build the super structure \( P_S \). As the expression has \( 2^{29} \) variables, we need \( 2^{29} \) linear independent equations to solve the equations system. In [5], they used different output limbs to get more than one equation from one subspace. However, in the Kravatte Achouffe, they have updated the rolling function \( \text{roll}_{X_e} \), which is a non-linear function now. Thus, the algebraic degree of output limbs are different from each other. We can only use the first limb to set up the equation because it is not rolled by \( \text{roll}_e \). As we have introduced before, each subspace of dimension 65 gives one equation on key bits. The super structure must include \( 2^{29} \) different subspaces of dimension 65. That is to say, \( \binom{65+e}{65} > 2^{29} \) and \( e \) is at least 7, thus the data complexity is \( 2^{72} (72 + 1) = 2^{78.3} \) blocks.

By Definition 3.2, we construct the input messages super structure \( P_S = \{(0, ..., e_1, ..., e_7)\} \in \{0, 1\}^{72} \). Then query the encryption oracle with all of the \( 2^{72} \) plaintexts of \( P_S \) to get the ciphertext \( C_S \) and choose \( 2^{29} \) subspaces \( \{S_i\} \) of dimension 65 from \( P_S \). The process is just like the method shown in Sect. 4.

Initialize a \( 2^{29} \times 2^{29} \) matrix. For each of the \( 2^{29} k'_i c'_i \) in \( F_{0,0} + F_{0,1} \):
- Evaluate the \( c'_i \) for each ciphertext of the first limb to get a bit vector.
- (\( M = 2^{72}, T = 2^{29} \cdot 2^{72} \))
- Apply the MOEBS Transform to this bit vector. After that, extract the \( 2^{29} \) bits that correspond to the coefficients of \( n_{maxterms} \) of subspaces \( \{S_i\} \) and copy them to a column of the matrix.
- (\( M = 2^{29}, T = 2^{29} \cdot 2^{31} \cdot \log_2 2^{72} \))

Solve the equations system and derive the key bits. In total, the process of the attack needs \( 2^{78.3} \) blocks data, \( 2^{72} \) bits memory and \( 2^{106.2} \) XOR operations.

5.3. Interpolation Attack against Xoofff

In this section, we consider another Farfalle construction algorithm Xoofff. The Xoofff is a deck function obtained by applying the Farfalle construction on 6-round Xoodoo. The process of analyzing Xoofff is similar to the attack against Kravatte Achouffe above.

5.3.1. Linearize the Xoodoo\(^{-2}\)

The first step is still linearizing the polynomial expression to get the equation (3). We have shown in Sect. 3.6, that the \( E_{x,y,z} \) of Xoodoo\(^{-1}\) potentially has 385 nonzero variables. The \( \rho^{-1}_{cast} \) destroys the property of \( \theta^{-1} \), but it dose not generate new variables. There are still 385 nonzero variables for each bit expression of the state \( F \).

We use \( G \) to denote the state of output of the second \( \chi^{-1} \), i.e. \( G = \chi^{-1}(F) \). The expression of \( G_{x,y,z} \) is as follows:

\[
G_{x,y,z} = F_{x,y+1,z}F_{x,y+2,z} + F_{x,y,z} \\
= w_{p_{x,y+1,z}} w_{p_{x,y+2,z}} + w_{p_{x,y,z}} Q_{p_{x,y+1,z}}(A) + Q_{p_{x,y+2,z}}(A) + Q_{p_{x,y,z}}(A)
\]

In the expression, \( \rho_{x,y,z} \) denotes the state bit moved to \( (x, y, z) \) by the permutation \( \rho^{-1}_{cast} \). As shown in Property 5, all \( Q \) polynomials are considered to involve all the 384 \( k^{out} \) and each \( w \) is considered as one variable. Thus, taking into account the constants, the number of potential variables in the expression \( G_{x,y,z} \) is

\[
1 + 385 + 385 + \binom{385}{2} + 1 + 385, \\
\]

which is approximately \( 2^{16.2} \).

5.3.2. The Analysis of the Attack against Xoofff-\{6,4,4\}

For Xoofff-\{6,4,4\}, we use the first limb as output. The affect of the non-linear rotation layer can be omitted. We only focus on the 8 rounds Xoodoo
after the Acc. We linearize the last two rounds to
get the expression of $G_{x,y,z}$ and its algebraic degree is
not more than 64. Since there are $2^{16.2}$ variables in
$G_{x,y,z}$ and one 65-dimension input messages subspace
gives one linear equation about key bits, we need $2^{16.2}$
such subspaces in the super structure. That is to say, $(65 + e) > 2^{16.2}$
and $e$ is at least 4, thus we construct a 69-
dimension input super structure. The data complexity
set up the equations system.

C plaintexts to get the ciphertexts $P_k$
recover
As we have got all the coefficients of the unknowns
Xoodoo linearize the last two rounds of
blocks data, $2$
complexity is $2$
Xoofff expressions of the same bit position of all the limbs are
this way, we carry out the attack on a bit from any
which does not increase the degree by the
avoid the diffusion. Then perform the attack on a bit
Xoofff has a good diffusion and amplifies the impact. However,
algebraic degree. After that, the
Observing the structure of
5.3.3. Analysis of the Attack against Xoofff-[6,6,2]
Observing the structure of Xoofff, We notice that the
non-linear rotation layer increases only a few bits’
algebraic degree. After that, the Xoodoo permutation
has a good diffusion and amplifies the impact. However,
if we perform the attack to the Xoofff-[6,6,2], we can
linearize the last two rounds of Xoodoo till $roll_{X_e}$
to avoid the diffusion. Then perform the attack on a bit
which does not increase the degree by the $roll_{X_e}$.
In this way, we carry out the attack on a bit from any
limb as long as its degree is not changed.

Although the output has different limbs, the
expressions of the same bit position of all the limbs are
the same for Xoodoo $-2$ and the expression has $2^{16.2}$
variables. We can choose two output limbs to construct
equations because $2 \times (65) > 2^{16.2}$. That is to say,
the input message is 68 blocks and the output message
is 2 blocks, and using this structure, we can get more
than $2^{16.2}$ independent linear equations. The total data
complexity is $2^{68}(68 + 2) = 2^{74.2}$ blocks.

The attack process is similar to the attack on
Xoofff-[6,4,4] version.

Firstly, construct the input messages super structure
$P_S = (e_0, ..., e_i, ..., e_{67}) \in \{0,1\}^{68}$ and query the
encryption oracle to get the ciphertext space $C_S$. Then
choose $2^{15.2}$ subspaces $\{S_i\}$ of dimension 65 from $P_S$.
Initialize a $2^{16.2} \times 2^{16.2}$ matrix. For each of the $2^{16.2}$
k’c’ in $G_{x,y,z}$, update one column by using two limbs.
Do the following steps for each of the two limbs.

- Evaluate $c_i'$ for each of the $2^{68}$ ciphertexts of this
  limb to get a $2^{68}$ bit vector.
  $(M = 2^{68}, T = 2 \times 2^{16.2}, 2^{68})$
- Apply the MOEBIUS TRANSFORM to this bit vector.
  Extract the $2^{15.2}$ bits that correspond to the
  coefficients of the maxterm of subspaces $\{S_i\}$
  and fill in the matrix as a column.
  $(M = 2^{16.2} \times 2, T = 2^{16.2} \times 2^{68}, 2^{68})$

As we have got all the coefficients of the unknowns $k'_i$
(i.e. the $2^{16.2} \times 2^{16.2}$ matrix), we can solve the system to
recover $k'_i$ and derive key bits. $(T = (2^{16.2})^3/\log_2 2^{16.2})$

Thus, in total, the process of the attack needs $2^{75.2}$
blocks data, $2^{68}$ bits memory and $2^{90.4}$ XOR operations.

6. CONCLUSION

In this paper, we extend the optimized interpolation
attack and apply it to the cryptanalysis of the 2 ciphers
Kravatte, Achouffe and Xoofff with Farfalle construction
and to Elephant-Delirium, which is a round 2 submission of the ongoing NIST lightweight
cryptography project. All the results turn out to be
the most efficient method for these ciphers thus far.

More importantly, compared with the optimized
method [10], the improved interpolation attack does not
need particularly stringent conditions on the algorithm
structure, and it is taken as a general method to analyse
some other ciphers.

REFERENCES

nist.gov/projects/lightweight-cryptography.
[2] Beyne, T., Chen, Y., Dobraunig, C. and Mennink,
B. (2019) ELEPHANT v1. In NIST Lightweight
be/cosic/elephant/
[3] Bertoni, G., Daemen, J., Hoffert, S., Peeters, M.,
Assche, G. V. and Keer, R. V. (2018) Farfalle:
Parallel Permutation-based Cryptography. In FSE
2018, Bruges, Belgium, March 5-7, pp. 1-38. IACR
Transactions on Symmetric Cryptology.
Transactions on Symmetric Cryptology.
[5] Chaigneau, C., Gilbert, H., Guo, J., Jean, J., Reinhard,
and Song, L. (2018) Key-Recovery Attacks on Full
Kravatte. In FSE 2018, Bruges, Belgium, March 5-7,
vol. 1 pp. 5-28. IACR Transactions on Symmetric
Cryptography.
Keccak Sponge Function Family Main Document.
Submission to NIST (Round 2).


