Blind side channel analysis on the Elephant LFSR Extended version

Julien Maillard, Awaleh Houssein Meraneh, Modou Sarry, Christophe Clavier, Hélène Le Bouder, Gaël Thomas

Blind side channel analysis on the Elephant LFSR Extended version

Julien Maillard[2,3], Awaleh Houssein Meraneh[1], Modou Sarry[1], Christophe Clavier[2], Hélène Le Bouder[1], and Gaël Thomas[4]

[1] IMT-Atlantique, OCIF, IRISA, Rennes, France
[2] Université de Limoges, XLIM-CNRS, France
[3] Université de Grenoble Alpes, CEA, LETI MINATEC Campus, F-38054, France
[4] DGA Maîtrise de l'Information, Bruz, France

Abstract. The National Institute of Standards and Technology (NIST) started a competition for lightweight cryptography candidates for authenticated encryption. Elephant is one of the ten finalists. Many physical attacks exist on the different traditional cryptographic algorithms. New standard are a new targets for this domain. In this paper, an improvement of the first theoretical blind side channel attack against the authenticated encryption algorithm Elephant is presented. More precisely, we are targeting the LFSR-based counter used internally. LFSRs are classic functions used in symmetric cryptography. In the case of Elephant, retrieving the initial state of the LFSR is equivalent to recovering the encryption key. This paper is an extension of a previous version. So an optimization of our previous theoretical attack is given. In the previous version, in only half of the cases, the attack succeeds in less than two days. In this extended paper, with optimization, the attack succeeds in three quarters of the cases.

Keywords: Blind Side Channel Analysis, Hamming Weight, Elephant, LFSR, NIST

1 Introduction

Internet of things (IoT) devices become more and more widespread within our day-to-day life. From military grade to general-purpose hardware, the need for strong security raises. The cryptosystems implemented on those devices must ensure both security and low power consumption overhead. In this context, the *National Institute of Standards and Technology* (NIST) started the competition for lightweight cryptography candidates for authenticated encryption [32]. An authenticated encryption algorithm should ensure confidentiality and integrity of the communications.

The security of authenticated encryption schemes can be supported by several strategies. Various approaches have been considered by the lightweight cryptography competition candidates: cryptographic permutations with sponge or duplex construction [16,3,15]; block cipher combined with a mode (e.g. AES combined with Galois/Counter Mode) [21,7]; stream cipher paradigms [19].

When discussing about the security of a cryptographic algorithm, numerous tools allow the cryptographers to prove the security of a cipher. Unfortunately, those tools do not consider the interaction of the computing unit with its physical environment. Physical attacks are a real threat, even for cryptographic algorithms proved secure mathematically. Physical attacks are divided in two families: side-channel analysis (SCA) and the fault injection attacks.

Motivation
Many attacks exist on the different traditional cryptographic algorithms, as detailed in the book [33]. Lightweight cryptography, much younger and used in embedded devices and IoT, has been far less studied. For example, attacks on stream ciphers [34] or sponge functions [35] are less common. That is why we chose to study SCA against new authenticated encryptions. The chosen algorithm is the cryptosystem Elephant [7]. More precisely, this paper focuses on its underlying *Linear Feedback Shift Registers* (LFSR), in a block cipher combined with a mode construction. Some attacks exist yet as in [34,22,10,11,24,23], but this work differs from state-of-the-art attacks by its attacker model. To the best of our knowledge, there is no blind side channel attack on LFSR in the context of authenticated encryption except our previous contribution [20]. This paper is an extension of [20], so motivation is to improve previous results.

Contribution
In this paper, we present a theoretical blind side channel attack targeting the LFSR of the Elephant algorithm. We exploit the usage of intermediate variables that are statistically dependent to the secret (here the secret LFSR initial state) and show that this structure could threaten the security of a cryptosystem's regarding SCA. Also, the study of the influence of the choice of the LFSR is presented. This paper is an extended version of a previous attack [20], so we present a major improvement: an optimization to find the best time, relative to the beginning of encryption, to start the attack is given in section 4.2.

Organization
The paper is organized as follows. In section 2, the context of blind side channel attack and the Elephant are introduced. The theoretical attack is explained in section 3, it is a reminder of the short version of the paper [20]. Details of implementated attack and new improvement are described in section 4. Then, section 5 presents experimental results and discussion about LFSR design. Finally, a conclusion is drawn in section 6.

2 Context

The first section starts by presenting the Elephant cryptosystem. This first part is extracted from the first shorted paper version [20] according the description of the standard [7]. Then, the background contents on blind side channel attacks

is introduced. Eventually, a brief state of the art of SCA attacks against LFSRs is presented.

2.1 Elephant

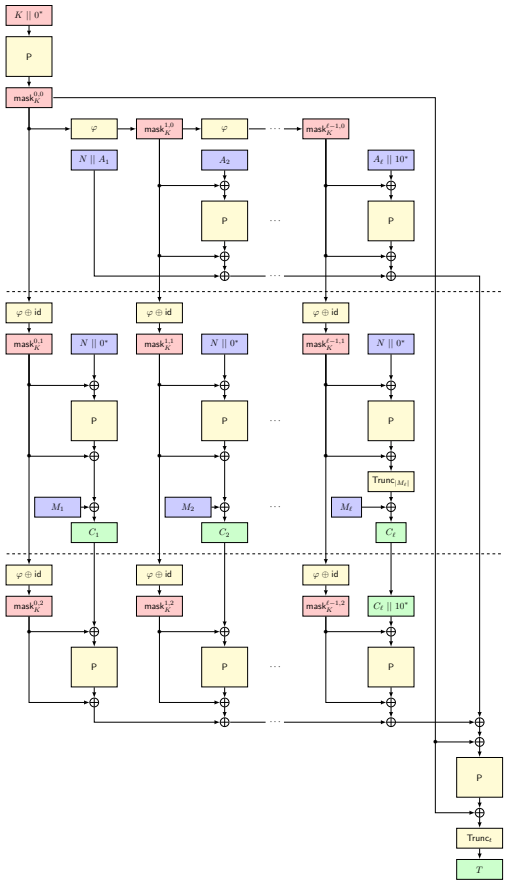


Fig. 1: Elephant associated data authentication (top), plaintext encryption (middle), and ciphertext authentication (bottom). This figure comes from [20] according to the description of Elephant [7].

The purpose of an authenticated encryption algorithm is to ensure both confidentiality and integrity. It takes as input different parameters: a plaintext, data associated to the plaintext, a secret key, and an initialisation vector, also called a nonce. The nonce is public but must be different for each new plaintext. The algorithm ensures confidentiality of the plaintext and integrity of both the plaintext and the associated data.

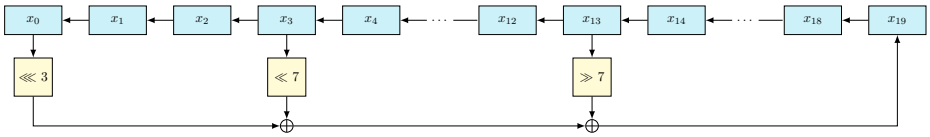


Fig. 2: 160-bit LFSR φ_{Dumbo}. This figure comes from [20] according to the description of Elephant [7].

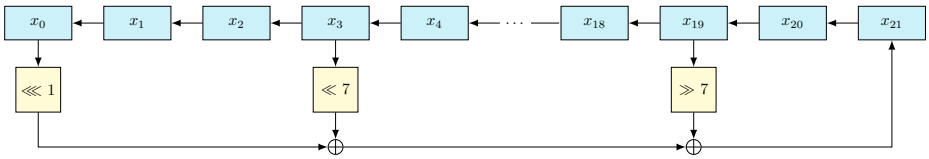


Fig. 3: 176-bit LFSR φ_{Jumbo}. This figure comes from [20] according to the description of Elephant [7].

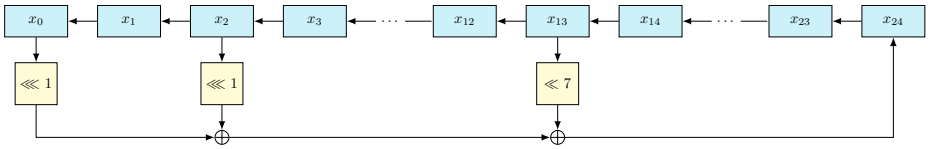


Fig. 4: 200-bit LFSR $\varphi_{\mathsf{Delirium}}$. This figure comes from [20] according to the description of Elephant [7].

$$\varphi_{\mathsf{Dumbo}} : (x_0, \cdots, x_{19}) \mapsto (x_1, \cdots, x_{19}, x_0 \lll 3 \oplus x_3 \ll 7 \oplus x_{13} \gg 7) \tag{1}$$

$$\varphi_{\mathsf{Jumbo}} : (x_0, \cdots, x_{21}) \mapsto (x_1, \cdots, x_{21}, x_0 \lll 1 \oplus x_3 \ll 7 \oplus x_{19} \gg 7) \tag{2}$$

$$\varphi_{\mathsf{Delirium}} : (x_0, \cdots, x_{24}) \mapsto (x_1, \cdots, x_{24}, x_0 \lll 1 \oplus x_2 \lll 1 \oplus x_{13} \ll 7) \tag{3}$$

Elephant [6,7] is a finalist to the NIST lightweight cryptography competition. It is a nonce-based authenticated encryption with associated data (AEAD). Its construction is based on an Encrypt-then-MAC that combines CTR-mode encryption with a variant of the protected counter sum [4,28]. Elephant uses a cryptographic permutation masked with LFSRs in an Even-Mansour-like fashion [17] in place of a blockcipher.

Let P be an n-bit cryptographic permutation, and φ an n-bit LFSR. Let the function $\mathsf{mask} : \{0,1\}^{128} \times \mathbb{N} \times \{0,1,2\} \to \{0,1\}^n$ be defined as follows:

$$\mathsf{mask}_K^{t,b} = (\varphi \oplus \mathsf{id})^b \circ \varphi^t \circ \mathsf{P}(K||0^{n-128}) \tag{4}$$

Let $\mathsf{Split}(X)$ be the function that splits the input X into n-bit blocks, where the last block is zero-padded. Let $\mathsf{Trunc}_\tau(X)$ be the τ left-most bits of X.

Encryption `enc` under Elephant gets as input a 128-bit key K, a 96-bit nonce N, associated data $A \in \{0,1\}^*$, and a plaintext $M \in \{0,1\}^*$. It out-

Algorithm 1 Elephant encryption algorithm `enc`

Require: $(K, N, A, M) \in \{0,1\}^{128} \times \{0,1\}^{96} \times \{0,1\}^* \times \{0,1\}^*$
Ensure: $(C, T) \in \{0,1\}^{|M|} \times \{0,1\}^t$
1: $M_1, \cdots, M_{\ell_M} \leftarrow \mathsf{Split}(M)$
2: **for** $t \leftarrow 1$ to ℓ_M **do**
3: $\quad C_t \leftarrow M_t \oplus \mathsf{P}(N||0^{n-96} \oplus \mathsf{mask}_K^{t-1,1}) \oplus \mathsf{mask}_K^{t-1,1}$
4: **end for**
5: $C \leftarrow \mathsf{Trunc}_{|M|}(C_1||\cdots||C_{\ell_M})$
6: $T \leftarrow 0^n$
7: $A_1, \cdots, A_{\ell_A} \leftarrow \mathsf{Split}(N||A||1)$
8: $C_1, \cdots, C_{\ell_C} \leftarrow \mathsf{Split}(C||1)$
9: $T \leftarrow A_1$
10: **for** $t \leftarrow 2$ to ℓ_A **do**
11: $\quad T \leftarrow T \oplus \mathsf{P}(A_t \oplus \mathsf{mask}_K^{t-1,0}) \oplus \mathsf{mask}_K^{t-1,0}$
12: **end for**
13: **for** $t \leftarrow 1$ to ℓ_C **do**
14: $\quad T \leftarrow T \oplus \mathsf{P}(C_t \oplus \mathsf{mask}_K^{t-1,2}) \oplus \mathsf{mask}_K^{t-1,2}$
15: **end for**
16: $T \leftarrow \mathsf{P}(T \oplus \mathsf{mask}_K^{0,0}) \oplus \mathsf{mask}_K^{0,0}$
17: **return** $(C, \mathsf{Trunc}_\tau(T))$

puts a ciphertext C as large as M, and a t-bit tag T. The description `enc` is given in **Algorithm** 1 and is depicted on Fig. 1.

Decryption `dec` gets as input a 128-bit key K, a 96-bit nonce N, associated data $A \in \{0,1\}^*$, a ciphertext $C \in \{0,1\}^*$, and τ-bit tag T. It outputs a plaintext M as large as C if the tag T is correct, or the symbol $\perp$ otherwise. The description of `dec` is analoguous to the one of `enc`.

Elephant comes in three flavours which differ on the n-bit cryptographic permutation P and the LFSR φ used, as well as the tag size t.

Dumbo uses the 160-bit permutation Spongent-π[160] [8], the LFSR φ_{Dumbo} given by equation (1) and illustrated on Fig. 2, and has tag size $\tau = 64$ bits.

Jumbo uses the 176-bit permutation Spongent-π[176] [8], the LFSR φ_{Jumbo} given by equation (2) and illustrated on Fig. 3, and has tag size $\tau = 64$ bits.

Delirium uses the 200-bit permutation Keccak-f[200] [5,31], the LFSR $\varphi_{\mathsf{Delirium}}$ given by equation (3) and illustrated on Fig. 4, and has tag size $\tau = 128$ bits.

2.2 State-of-the-art

Side channel analysis Even if an algorithm has been proven to be mathematically secure, its implementation can open the gate to physical attacks. SCA are a subcategory of physical attacks. They exploit the fact that some physical states of a device depend on intermediate values of the computation. This is the so-called leakage of information of the circuit. It could be used to retrieve sensitive data, such as secret keys, or to reverse engineer an algorithm. An SCA is often led with a divide-and-conquer approach. Namely, the secret is divided into small pieces that are analysed independently.

Different kinds of leakage sources can be exploited as execution time [18], power consumption [25] or electromagnetic (EM) radiations [36]. In this paper, we consider a power consumption or EM leakage channel. At each instant, the measurement of the intensity of the electric current reflects the activity of the circuit. The power consumption of a device is a combination of the power consumption of each of its logic gates.

Several analysis paradigms have been described in the literature. The *Simple Power Analysis* (SPA) [29] are called simple because they determine directly, from an observation of the power consumption, during a normal execution of an algorithm, information on the calculation performed or the manipulated data. Other attacks like *Correlation Power Analysis* (CPA) [9] use a mathematical model for the leakage. A confrontation between measurement and model is performed. More precisely, a statistic tool called distinguisher gives score to the different targets. Template attacks are statistical categorizations [1] that require no leakage model *a priori*. It is a domain in its own right, as shown different books [30,33].

Blind side channel analysis The *blind side channel analysis* (BSCA) family is new improvement in SCA. Linge *et al.* has presented the concept in [27]. In parallel, Le Bouder *et al.* published an attack in [26]. Then, these works have been improved by Clavier *et al.* in [13], moreover this contribution introduces for the first time, the term of blind side channel. Now it is a new family of SCA [14,2,37,20].

The main idea is to only perform the attack on the leakage measurements *i.e*, without data such as plaintexts or ciphertexts.

In BSCA, the *Hamming weight* (HW) leakage model have often been used and a strong assumption is made: the attacker is supposed to retrieve a noisy HW from the leakage. In this paper, the considered adversary model is that the HW of all manipulated intermediate variables can be recovered by the attacker. Several techniques, such as signal filtering, trace averaging or templates [12], can be used in order to fulfill this prerequisite.

Overview of SCA attacks on LFSRs Linear feedback shift registers (LFSRs) with primitive polynomials are used in many symmetric cryptographic primitives because of their well-defined structure and remarkable properties such as long period, ideal autocorrelation and statistical properties.

The information leakage and the vulnerability of stream ciphers based on Galois LFSRs are studied in [22] and those based on Fibonacci LFSRs are analysed in [10]. In [22], the information leakage of XOR gates is exploited to perform a simple side-channel attack. However, if the leakage from the XOR gates is too low compared to other operations in the cipher, the attack fails. In [11], the attack recovers the initial state of a Galois LFSR by determining the output of the LFSR from the difference in power dissipation values in consecutive clock cycles.

In this paper, a theoretical blind side channel attack targeting the LFSR of the Elephant algorithm is presented. Whereas several attacks on LFSRs have been described in the literature, the specific structure of the Elephant cryptosystem allows us to elaborate a new approach that is depicted in the rest of this paper.

3 Theoretical attack

This section is a reminder of the first shorted paper version [20].

3.1 Goal

LFSRs are used in different lightweight cryptography candidates, and its initial state often depends on both a key and a nonce. As the nonce needs to be changed for each encryption request, attacks on such schemes are limited to the decryption algorithm. In the case of Elephant, the LFSR only depends on the secret key. Consequently, our attack can be applied in an encryption scenario.

The goal of the presented attack is to retrieve the LFSR secret initial state. One has to remark three important points:

- Retrieving the initial state of the LFSR, which is equal to $\mathsf{mask}_K^{0,0}$, is equivalent to retrieving the secret key. Indeed, the initial state is the result of the known permutation P applied to the key.
- As the retroaction polynomial is publicly known, it is possible to shift the LFSR backwards: an attacker who recover enough consecutive bytes of the secret stream is able to reconstruct the initial state.
- The smaller the LFSR is, the more the attack is able to succeed. As a consequence, the Dumbo instance (see Fig. 2) is the most vulnerable one: the following of this paper is focused on Dumbo.

3.2 Leakage in the LFSR

In this attack, it is assumed that the Hamming weight of every byte of the LFSR can be obtained by an attacker. Let x be a byte: it can take any of the 256 values in $[\![0, 255]\!]$. With the HW of x, the attacker reduces the list of possible values, as shown in Table 1.

$HW(x)$	0	1	2	3	4	5	6	7	8
$\#x$	1	8	28	56	70	56	28	8	1

Table 1: Number of possible values per Hamming weight value [20,26].

Since the LFSR generates a single new byte at each iteration, let $(x_j, \cdots, x_{j+19}) = \mathsf{mask}_K^{j,0}$ be the content of the Dumbo LFSR and x_{j+20} the byte generated at iteration j. Precisely, the attacker has the following relation (L1), according to the equation (1).

L1 $x_{j+20} = (x_j \lll 3) \oplus (x_{j+3} \ll 7) \oplus (x_{j+13} \gg 7)$.

The first idea is to use the knowledge of the following Hamming weights: $HW(x_{j+20})$, $HW(x_{j+13})$ and

$$HW(x_j) = HW(x_j \lll 3). \tag{5}$$

So with the two equations (L1) and (5) the attacker has:

$$HW(x_{j+20}) = \begin{cases} HW(x_j) \\ HW(x_j)+1 \\ HW(x_j)-1 \\ HW(x_j)+2 \\ HW(x_j)-2 \end{cases} \tag{6}$$

Looking more precisely at equation (L1), it can be seen that the difference $HW(x_{j+20}) - HW(x_j)$ only depends on four bits. Let $x_j[i]$ denote the i-th least significant bit of byte x_j, these four bits are $\{x_{j+3}[0]; x_{j+13}[7]; x_j[4]; x_j[5]\}$. Table 2 gives the value of observed difference $HW(x_{j+20}) - HW(x_j)$ depending on the values of these four bits. In the worst case, there are only 6 possibilities left, out of 16.

$HW(x_{j+20}) - HW(x_j)$		$(x_{j+3}[0], x_{j+13}[7]) =$			
		$(0,0)$	$(0,1)$	$(1,0)$	$(1,1)$
$(x_j[4], x_j[5]) =$	$(0,0)$	0	$+1$	$+1$	$+2$
	$(1,0)$	0	$+1$	-1	0
	$(0,1)$	0	-1	$+1$	0
	$(1,1)$	0	-1	-1	-2

Table 2: Values of $HW(x_{j+20}) - HW(x_j)$ according to $\{x_{j+3}[0]; x_{j+13}[7]; x_j[4]; x_j[5]\}$ [20] .

3.3 Link between the different masks

The value $\mathsf{mask}_K^{j,1}$ can be expressed in terms of $\mathsf{mask}_K^{*,0}$ as in (7).

$$\begin{aligned} \mathsf{mask}_K^{j,1} &= (\varphi \oplus \mathsf{id}) \left(\mathsf{mask}_K^{j,0} \right) \\ &= \varphi \left(\mathsf{mask}_K^{j,0} \right) \oplus \mathsf{mask}_K^{j,0} \\ &= \mathsf{mask}_K^{j+1,0} \oplus \mathsf{mask}_K^{j,0}. \end{aligned} \tag{7}$$

Likewise, for $\mathsf{mask}_K^{j,2}$, equation (8) holds.

$$\begin{aligned}\mathsf{mask}_K^{j,2} &= (\varphi \oplus \mathsf{id})^2 \left(\mathsf{mask}_K^{j,0}\right) \\ &= (\varphi^2 \oplus \mathsf{id}) \left(\mathsf{mask}_K^{j,0}\right) \\ &= \varphi^2 \left(\mathsf{mask}_K^{j,0}\right) \oplus \mathsf{mask}_K^{j,0} \\ &= \mathsf{mask}_K^{j+2,0} \oplus \mathsf{mask}_K^{j,0}\end{aligned} \tag{8}$$

As in the case of $\mathsf{mask}_K^{j,0}$, let y_j denote either the byte j of $\mathsf{mask}_K^{0,1}$ when $0 \leq j \leq 19$, or the new byte obtained after j iterations of the LFSR initialized with $\mathsf{mask}_K^{0,1}$. Likewise, let z_j denote either the byte j of $\mathsf{mask}_K^{0,2}$ when $0 \leq j \leq 19$, or the new byte obtained after j iterations of the LFSR initialized with $\mathsf{mask}_K^{0,2}$.

Equation (7) then translates to equation (9).

$$y_j = x_j \oplus x_{j+1} \tag{9}$$

Likewise, the equation 8 translates to (10).

$$z_j = x_j \oplus x_{j+2}. \tag{10}$$

The evolution of the LFSR is analogous to (L1):

$$y_{j+20} = (y_j \lll 3) \oplus (y_{j+3} \ll 7) \oplus (y_{j+13} \gg 7). \tag{11}$$

$$z_{j+20} = (z_j \lll 3) \oplus (z_{j+3} \ll 7) \oplus (z_{j+13} \gg 7). \tag{12}$$

The attacker can thus exploit two attack vectors: on the one hand, equations (L1), (11), and (12) coming from iterating the LFSR, and on the other hand, equations (9) and (10) coming from the different masks used for domain separation.

4 Attack strategy

For a byte x_j of the Dumbo LFSR with $j \geq 0$, let x'_j denote a guess of its value by the attacker. Given m successive bytes $(x_j, \cdots, x_{j+m-1})$, let $\mathbf{X}_j^m$ denote the set of guesses $(x'_j, \cdots, x'_{j+m-1})$ satisfying the constraints of the Hamming weights of masks x, y and z depicted in Equations 9 and 10.

4.1 Algorithm of the attack

The whole search space corresponding to the initial state of the LFSR is represented as a rooted tree. The nodes at depth j correspond to all the possible values for the bytes x_0 to x_j of the LFSR. The tested candidates are denoted by $(x'_0, \cdots, x'_{19})$. The nodes in the graph of the search space are labelled as follows:

- the nodes at depth j correspond to all the possible values of $(x'_0, \cdots, x'_j)$;

Algorithm 2 $\mathsf{isvalid}(x'_0, \cdots, x'_j)$

Require: Byte-wise partial candidate $(x'_0, \cdots, x'_j)$ of length $1 \leq j+1 \leq 20$
Assumes $\mathsf{isvalid}(x'_0, \cdots, x'_{j-1})$ is true.
Ensure: **true** if candidate $(x'_0, \cdots, x'_j)$ is compatible with the observations, **false** otherwise
# Hamming weights of the xors
if $HW(x'_j) \neq HW(x_j)$ **then**
  return false
end if
if $HW(x'_j \oplus x'_{j-1}) \neq HW(y_{j-1})$ **then**
  return false
end if
if $HW(x'_j \oplus x'_{j-2}) \neq HW(z_{j-2})$ **then**
  return false
end if
# Hamming weights of the feedbacks
if $|HW(x'_j \lll 3) - HW(x_{j+20})| > 2$ **then**
  return false
end if
if $|HW(x'_{j-3} \lll 3 \oplus x'_j \ll 7) - HW(x_{j+17})| > 1$ **then**
  return false
end if
if $HW(x'_{j-13} \lll 3 \oplus x'_{j-10} \ll 7 \oplus x'_j \gg 7) \neq HW(x_{j+7})$ **then**
  return false
end if
return true

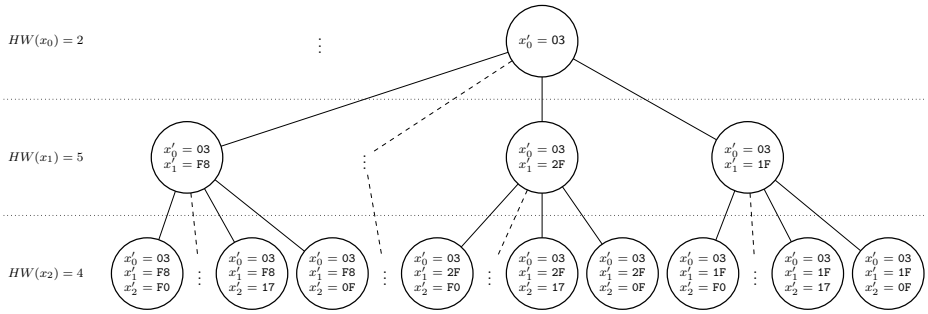


Fig. 5: Example of the tree representation of the LFSR initial state for the Hamming weights given on the left. Only the first three layers of the subtree rooted at $x'_0 = \mathtt{03}$ are shown.

– the children of node $(x'_0, \cdots, x'_j)$, are the nodes labelled: $(x'_0, \cdots, x'_j, x'_{j+1})$ for all values of x'_{j+1}.

Algorithm 3 Attack

Require: Observed Hamming weights $HW(x_0), \cdots, HW(x_{19})$, $HW(y_0), \cdots, HW(y_{18})$, and $HW(z_0), \cdots, HW(z_{17})$. For the sake of clarity, they are seen as global variables.
Ensure: S set of keys compatible with the observed Hamming weights

$(x'_0, \cdots, x'_{19}) \leftarrow (0, \cdots, 0)$
$\ell \leftarrow 0$
$S \leftarrow \{\}$
while true do
  if $j < 19$ **and** isvalid$(x'_0, \cdots, x'_j)$ **then**
    $j \leftarrow j + 1$
    $x'_j \leftarrow 0$
  else
    if $j = 19$ **and** isvalid$(x'_0, \cdots, x'_j)$ **then**
      $S \leftarrow S \cup \{(x'_0, \cdots, x'_j)\}$
    end if
    while $j \geq 0$ **and** $x'_j =$ FF **do**
      $j \leftarrow j - 1$
    end while
    if $j \geq 0$ **then**
      $x'_j \leftarrow x'_j + 1$
    else
      break
    end if
  end if
end while
return S

In practice, to reduce the number of nodes, only the nodes having the correct Hamming weights are considered. In other words, it suffices to consider nodes with $HW(x'_j) = HW(x_j)$. An example of such a tree is given on Fig. 5.

A backtracking algorithm is used. The tree is traversed in a depth-first manner. For each step, the attacker tests whether the current candidate $(x'_0, \cdots, x'_j)$ satisfies the different conditions given by the observed Hamming weights. This test is given by **Algorithm** 2.

If the test succeeds, the algorithm goes down to the next layer to test the values of the byte x'_{j+1}. If it reaches the bottom of the tree, then a good candidate has been found, and can be saved. The algorithm then iterates upon the next untested node.

If, at some point, the Hamming weights conditions do not hold for the current (partial) candidate, then no node in the sub-tree rooted at that node can lead to a good candidate. Thus, it can be pruned from the whole tree, saving the cost of browsing it. Finally, the algorithm ends when the whole tree has been explored. A pseudocode of the attack is given by **Algorithm** 3.

4.2 Optimisation of the attack

Goal of the optimization Recall from **Algorithm** 1 that for each new block of data to encrypt, a different mask is used. Therefore the attacker actually can choose which values of index t at lines 2, 10, and 13 of **Algorithm** 1 they would rather attack.

The question raised at this point is: at which time t does the attacker minimize the complexity of the attack? The attacker wants to establish a metric $E(\cdot)$ such that $E(t)$ allows estimating the complexity of an attack at time t. The attacker constructs E such that the computation of $E(t)$:

1. is fast enough to allow exploring a large number of values of t and
2. is accurate enough so that the attacker can accurately select the value of t with minimal attack complexity.

Principle The general idea for finding the best attack position is to obtain an estimation of the attack complexity at a given time t. For this sake, the LFSR state $(x_t, \cdots, x_{t+19})$ at time t is divided into tuples that can be treated with three different operations. Note that this approach does not exploit the retroaction polynomial of the LFSR. This allows to use it for all the flavours of the Elephant cryptosystem.

1. **Enumerate** This function takes as an input m indices $(i, \cdots, i+m-1)$ and returns all the candidates $\mathbf{X}_i^m$ for $(x'_i, \cdots, x'_{i+m-1})$ satisfying the constraints of masks x, y and z depicted in equations (9) and (10), based on the knowledge of corresponding $3m-3$ Hamming weights. This operation explores all the possible combinations for each x'_i and only retains the candidates matching the conditions imposed by masks x, y and z. Note that the *enumerate* function quickly becomes computationally intense as m grows. In our experiments, we use this function for $m \in \{1, 2, 3\}$.
2. **Merge** When m grows, the attacker proceeds with a divide and conquer strategy. Indeed, for $k < m$, the attacker first computes candidates $\mathbf{X}_i^k$ and $\mathbf{X}_{i+k}^{m-k}$ thanks to the *enumerate* function. Then candidates $\mathbf{X}_i^k$ and X_{i+k}^{m-k} are merged into $\mathbf{X}_i^m$ so that each remaining candidate in $\mathbf{X}_i^m$ satisfies the constraints on masks y and z. This approach allows benefiting from the reduced candidate sets $\mathbf{X}_i^k$ and $\mathbf{X}_{i+k}^{m-k}$, allowing to reduce the size of the search space for $\mathbf{X}_i^m$. For a growing m, storing the $\mathbf{X}_i^m$ set in memory can become impractical, hence the *merge* function comes with the *merge_ count* variant, that only returns the number $\mid \mathbf{X}_i^m \mid$ of candidates. We emphasize that calling *merge_ count* on X_i^m disables calling it for another merge.
3. **Merge estimator** The number of candidates $\mid \mathbf{X}_i^m \mid$ can be assessed by applying a reduction factor r on the product of $\mid \mathbf{X}_i^k \mid$ and $\mid \mathbf{X}_{i+k}^{m-k} \mid$. Namely, it is defined as:
$$r = \frac{\mid \mathbf{X}_i^k \mid \times \mid \mathbf{X}_{i+k}^{m-k} \mid}{\mid \mathbf{X}_i^m \mid} \tag{13}$$
Hence, the merging estimation procedure takes as input two indices tuples $(i, \cdots, i+k-1)$ and $(i+k, \cdots, i+m)$, for $k < m$, and their corresponding

number of candidates $|\ \mathbf{X}_i^k\ |$ and $|\ \mathbf{X}_{i+k}^{m-k}\ |$. The aim of the estimator is to approximate r based on statistical tools without exploring the entire Cartesian product of $\mathbf{X}_i^k$ and $\mathbf{X}_{i+k}^{m-k}$. The construction of the estimator is discussed in the following.

Gathering information The estimator E for the attack complexity can then be defined by applying the previous three functions. The procedure used for the Dumbo LFSR is depicted in Fig. 6.

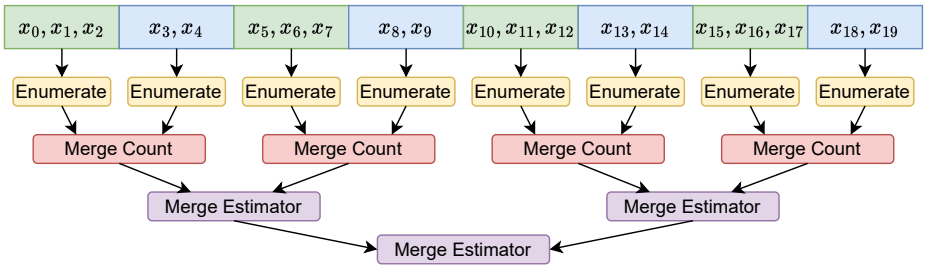


Fig. 6: Gathering information for the Dumbo LFSR.

The 20 LFSR bytes are split into 4 quintuplets. Each quintuplet is itself split into a triplet and a pair of bytes. The attacker gathers the candidates for each triplet and pair with the *enumerate* function, and then call the *merge_ count* function on the union of the triplets and pairs to count the exact number of candidates for the quintuplets. Eventually, the attacker applies the *merge estimator* between the first and second, and the third and fourth quintuplets before calling the estimator a last time on the full state.

At this point, it can be seen that the accuracy of the complexity returned by E highly depends on the quality of the *merge_ estimator*.

Crafting a merge estimator Naive estimator
Equations (9) and (10) allow to derive a *reduction triangle* for the merge of two tuples (see Fig. 7).

This reduction triangle can be exploited to obtain a broad estimation on the information gained by a merge, hence, a estimation of the reduction factor of the Cartesian product of $\mathbf{X}_i^k$ and $\mathbf{X}_{i+k}^{m-k}$. Let $|\ \mathbf{X}_{triangle}\ |$ define the number of candidates for the reduction triangle. This corresponds to the product of the number of bytes values that match each three gathered Hamming weights. Then, r_{naive} is defined by:

$$r_{naive} = \frac{|\ \mathbf{X}_{triangle}\ |}{2^{24}} \tag{14}$$

The estimation error of r_{naive} is defined as follows:

$$err_{naive} = r_{naive} - r \tag{15}$$

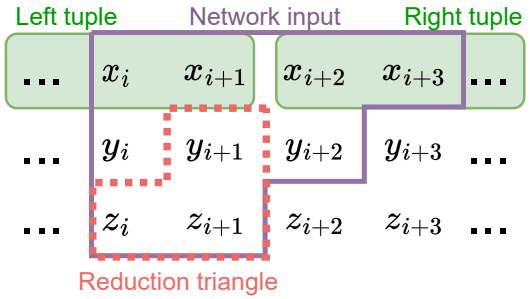
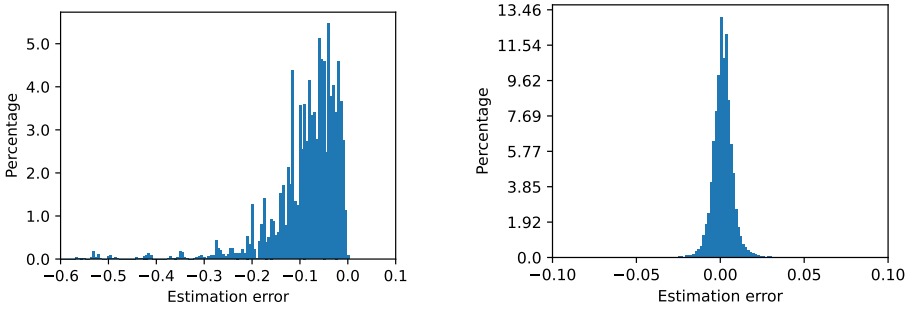


Fig. 7: Necessary data for the naive estimator (reduction triangle) and the neural network based merge estimator.

When $err_{naive} > 0$, reduction factor has been overestimated. This situation raises an issue for the attacker, as the complexity of the attack is falsely underestimated. On the contrary, $err_{naive} < 0$ indicates that the reduction factor has been underestimated. This means that the attacker would probably not consider this step to perform the attack, even if, in reality, the attack complexity would have been much inferior. A distribution of the err_{naive} values is depicted in Fig 8a.



(a) Distribution of err_{naive} for $m = 4$.

(b) Distribution of $err_{network}$ for $m = 4$.

Fig. 8: Estimation error for the naive and the neural network approach.

The attacker stresses that r_{naive}, despite being fast to compute, has the flaw of not considering all the inter-mask dependencies depicted in equations (9) and (10).

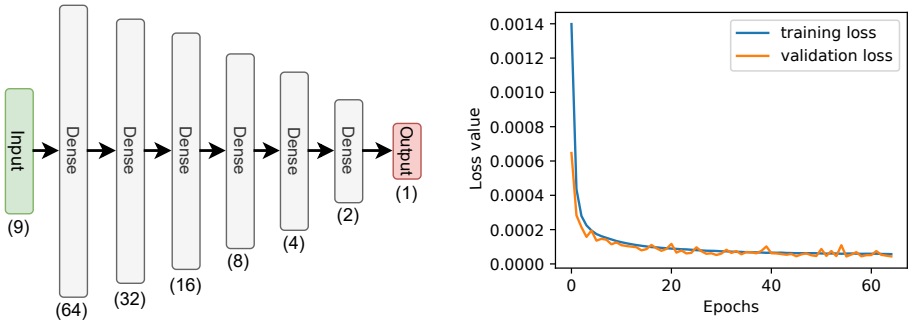
Neural network based estimator

Inter-masks dependencies are difficult to handle as-is for crafting a merge esti-

mator. Luckily, the complex relations that bound these bytes can be exploited thanks to the power of a neural network. Hence, the Hamming weights of the bytes depicted in Fig. 7 can be fed to a neural network in order to predict r.

To do so, we create a dataset by performing 1.3M merges of candidates for two pairs of bytes. Merging pairs allows to quickly compute r, and thus allows building a significant dataset in a reasonable amount of time. We keep the real r values as labels that will be provided to the network in order for it to provide an estimator $r_{network}$.

A neural network composed of 6 fully connected hidden layers is crafted, whose sizes are depicted in Fig 9a. The choice of the neural network architecture has been performed by progressively tuning the parameters in order to reduce the variance and standard deviation of the estimation error for $r_{network}$. Each layer uses the *Relu* activation function, and the model is compiled with the *adam* optimizer and the *Mean-Square-Error* loss function. The model is trained upon 65 epochs upon a 1.287M sample datasets with a 0.01 validation ratio. Training and validation losses are depicted in Fig. 9b.



(a) Architecture of the neural network for the merge estimator E.

(b) Evolution of training and validation losses of the neural network.

Fig. 9: Neural network architecture and training history.

The total training time of the model is approximately 15 minutes on a `Intel Core i7-8565U CPU`. As for r_{naive}, the estimation error of $r_{network}$ is measured as follows:

$$err_{network} = r_{network} - r \tag{16}$$

The distribution of $err_{network}$ on a testing dataset of 13K samples (*i.e.*, a dataset that has not been used to train the network) is displayed in Fig. 8b.

Discussion

Fig. 8a shows that our naive estimator highly underestimates the reduction factor of a merge between two tuples. Some instances even show more than a 50%

delta between r_{naive} and the real reduction factor r. Fig. 8b shows that the neural network provides an estimation of r which is more accurate than the naive estimator, as the mean is closer to 0. Moreover, the error variance and standard deviation of the distribution have been reduced compared to the naive estimator. More importantly, the $err_{network}$ distribution does not show outliers (*i.e*, predictions that are far from the mean) unlike err_{naive}. In our experiment, we hence use $r_{network}$ as the reduction factor estimation for the metric E (see Fig. 6).

Estimating the best position The best position for the attack is assessed by applying estimator E in a sliding window fashion upon the iterations of the LFSR. In terms of performance, the computation time is approximately one second per iteration on a `Intel Core i7-8565U CPU`: this enables estimating the attack complexity on several dozens of thousands of steps in less than a day.

Depending on the attacker's computational power, several strategies can be considered. First, the attacker can set up a threshold upon attack complexity. When targeting an encryption stream, the attacker can apply the metric E on the sliding window until it returns an attack complexity that is below this threshold. Another approach is to fix a limit on the sliding window algorithm. Then, when all the attack complexity estimations have been returned, the attacker chooses the lowest one.

Fig. 10 illustrates a run of the sliding window algorithm on a Dumbo LFSR with initial state bytes sampled from the uniform distribution. The metric $E(t)$ is computed for $t \in \{0, \cdots, 20000\}$. In this example, minimal estimated attack complexity is approximately $2^{39.30}$ for $t = 10909$. Note that each $E(t)$ can be computed independently: the algorithm can easily be transformed into a parallel version.

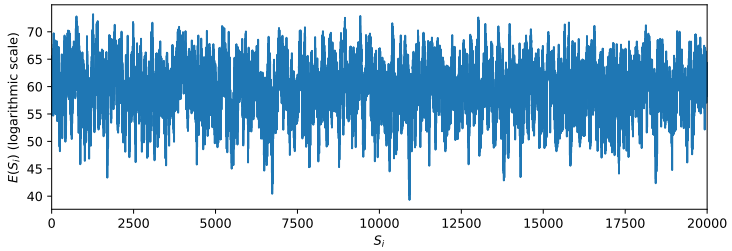


Fig. 10: Example output of the sliding window E estimation algorithm.

5 Results and discussion

5.1 Elephant attack

We have simulated the attack on randomly generated Dumbo keys. We selected the lowest 25% of keys with respect to expected complexity given by the estimator E derived in section 4.2. This gave us $N_{runs} = 1275$ keys to test. For each, the number N_{nodes} of nodes effectively traversed in the tree has been counted. This number roughly corresponds to the time complexity of the attack. Among these nodes, we have specifically counted the number N_{keys} of nodes on the last layer; *i.e.* nodes that correspond to plausible guesses that remain to be brute forced to finish the attack.

Only just above three quarters (77.73%) of the runs have ended after two days. This is an improvement (24.16% more) compared to [20] where only 53.57% had finished in the same amount of time. On average, for the runs that finished after two days, the number of nodes traversed is $N_{nodes} = 2^{41.92}$, and the number of remaining keys is $N_{keys} = 2^{36.52}$.

Figure 11 shows the link between the estimator E and the actual computation time for the differents keys tested. Quartiles for estimator and time distributions are given in Table 3. It should be remembered that data given here only represent the lowest 25% of the estimator distrbution and that time is capped at 48h, and therefore do not represent the full distributions.

	quartile Q_1	median Q_2	quartile Q_3
estimator E (log_2)	51.62	53.64	55.07
time (hours)	1.61	8.88	37.96

Table 3: Quartiles for estimator and time distributions.

5.2 Impact of the generation of masks

The threat brought by this attack upon the Elephant cryptosystem implies a discussion about mitigations. Apart from using generic countermeasures, like *e.g.* Boolean masking, there seem to be two possibilities for improvement. Indeed, the attacker gains information from two sources:

- from equations (7) and (8) used to derive the masks for domain separation;
- from the LFSR state update equation (L1).

Thus, either the mask derivation or the LFSR can be changed, or both. This section studies the former case.

We ran two experiments, similar to section 5.1 except that the attacker does not gain information from every Hamming weight. In the first experiment, they only know the values of the $HW(x_j)$, and the $HW(y_j)$. In other words, compared

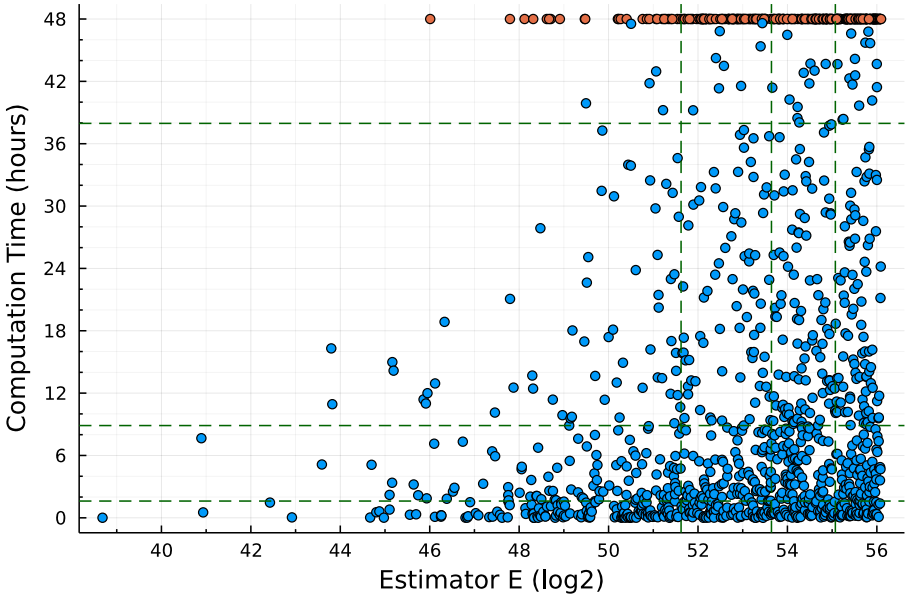


Fig. 11: Estimator E versus actual computation time for finished runs (blue) and unfinished runs (red). Quartiles are represented by dashed lines.

to the experiment in the section 5.1, they lost the knowledge of the $HW(z_j)$. Likewise, in the second experiment, they only know the values of the $HW(x_j)$. In both cases, none of the $N_{runs} = 120$ runs done has terminated after a week.

From these experiments, it seems that the combined knowledge of the $HW(x_j)$, $HW(y_j)$, and $HW(z_j)$ contributed heavily on the success of the attack. It would then seem a good idea to tweak the cryptographic mode of operation by finding another way of generating masks for domain separation.

5.3 Studies on different LFSRs

This section is dedicated to the study of the influence of the choice of the LFSR. We stress that the results shown here are obtained without the optimizations the optimization presented in 4.2: in practice, an attacker can hope lowering the attack complexity by exploiting the optimization.

To keep the spirit of the original Elephant algorithm, only Fibonacci-like LFSRs, at the byte level, are considered. More specifically, LFSRs considered are: LFSRs where a single new byte is computed from a combination of three bytes using byte-wise shifts and rotations. As usual, the associated feedback polynomial must be primitive to ensure only maximum-length sequences can be generated. Among all possible candidates, different behaviours can be triggered.

In this paper, the *type* of a LFSR is defined as the sequence of the number of bits unknown to the attacker at each depth in the tree where a new feedback occurs.

Looking at equation (L1), it can be seen that:

$$|HW(x_{j+20}) - HW(x_j \lll 3)| \leq 2$$

since there are only 2 bits that are modified by:

$$x_{j+3} \ll 7 \oplus x_{j+13} \gg 7.$$

Thus, if other feedback functions are used, with more bits involved, it can be expected to have an impact on the attack.

Later in the attack, when at depth 3 in the tree, the same idea can be applied to check whether:

$$|HW(x_{j+17}) - HW(x_{j-3} \lll 3 \oplus x_j \ll 7)| \leq 1$$

So since now only the single bit $x_{j+10} \gg 7$ is unknown. Consequently, the type of the Dumbo LFSR is $[2, 1]$.

LFSRs with different types can be a first criterion when testing our attack.

A second criterion is the spacing between the feedback bytes. Indeed, the tighter they are, the faster the attacker can use equation (L1) at its full potential.

In the case of Dumbo, the feedback bytes are at indices 0, 3, and 13. We call 13 the *depth*, this is simply the highest index of the feedback.

We chose LFSRs based on these two criteria. Types is defined from $[2, 1]$ to $[8, 8]$. For types $[2, 1]$, and $[5, *]$, we looked at all the possible LFSRs in order to study the influence of their depth.

The state update function of the different LFSR tested are given by equations (L2) to (L21). Their type and depth are given at the second, respectively third, column of Table 4.

L2 $x_{j+20} \leftarrow x_j \lll 3 \oplus x_{j+1} \ll 7 \oplus x_{j+11} \gg 7$
L3 $x_{j+20} \leftarrow x_j \lll 3 \oplus x_{j+14} \gg 3 \oplus x_{j+17} \gg 7$
L4 $x_{j+20} \leftarrow x_j \lll 1 \oplus x_{j+3} \gg 3 \oplus x_{j+13} \gg 7$
L5 $x_{j+20} \leftarrow x_j \lll 1 \oplus x_{j+9} \gg 3 \oplus x_{j+15} \gg 7$
L6 $x_{j+20} \leftarrow x_j \lll 3 \oplus x_{j+9} \ll 4 \oplus x_{j+19} \gg 7$
L7 $x_{j+20} \leftarrow x_j \lll 3 \oplus x_{j+1} \ll 5 \oplus x_{j+3} \gg 6$
L8 $x_{j+20} \leftarrow x_j \lll 1 \oplus x_{j+4} \gg 3 \oplus x_{j+19} \gg 5$
L9 $x_{j+20} \leftarrow x_j \lll 1 \oplus x_{j+7} \gg 3 \oplus x_{j+18} \gg 5$
L10 $x_{j+20} \leftarrow x_j \lll 1 \oplus x_{j+3} \gg 3 \oplus x_{j+9} \gg 5$
L11 $x_{j+20} \leftarrow x_j \lll 3 \oplus x_{j+1} \gg 7 \oplus x_{j+17} \ll 4$
L12 $x_{j+20} \leftarrow x_j \lll 3 \oplus x_{j+5} \gg 7 \oplus x_{j+19} \gg 3$
L13 $x_{j+20} \leftarrow x_j \lll 1 \oplus x_{j+5} \ll 7 \oplus x_{j+16} \ll 3$
L14 $x_{j+20} \leftarrow x_j \lll 1 \oplus x_{j+1} \gg 7 \oplus x_{j+9} \gg 3$
L15 $x_{j+20} \leftarrow x_j \lll 1 \oplus x_{j+13} \ll 5 \oplus x_{j+19} \ll 3$
L16 $x_{j+20} \leftarrow x_j \lll 3 \oplus x_{j+14} \gg 7 \oplus x_{j+17} \gg 3$

L17 $x_{j+20} \leftarrow x_j \lll 3 \oplus x_{j+4} \ll 1 \oplus x_{j+5} \gg 6$
L18 $x_{j+20} \leftarrow x_j \lll 1 \oplus x_{j+3} \gg 1 \oplus x_{j+9} \ll 1$
L19 $x_{j+20} \leftarrow x_j \lll 1 \oplus x_{j+4} \gg 1 \oplus x_{j+5} \ll 1$
L20 $x_{j+20} \leftarrow x_j \lll 3 \oplus x_{j+1} \gg 1 \oplus x_{j+8} \lll 7$
L21 $x_{j+20} \leftarrow x_j \lll 3 \oplus x_{j+3} \ll 5 \oplus x_{j+4} \lll 5$

We ran the same experience as in section 5.1 for every considered LFSR with $N_{tests} = 120$. For each LFSR, we noted the proportion of runs finished after two days of computations, the average number of nodes effectively traversed in the tree, and the average number of remaining keys. Results are summarized in Table 4.

LFSR	type	depth	finished	N_{nodes}	N_{keys}
(L1)	$[2,1]$	13	53.57%	$2^{41.82}$	$2^{36.59}$
(L2)	$[2,1]$	11	82.5%	$2^{41.23}$	$2^{36.39}$
(L3)	$[5,1]$	17	0.83%	$2^{42.89}$	$2^{34.68}$
(L4)	$[5,1]$	13	94.17%	$2^{39.68}$	$2^{33.68}$
(L5)	$[5,1]$	15	28.33%	$2^{42.13}$	$2^{35.25}$
(L6)	$[5,1]$	19	11.67%	$2^{42.38}$	$2^{36.77}$
(L7)	$[5,2]$	3	100.0%	$2^{30.93}$	$2^{24.93}$
(L8)	$[5,3]$	19	0.83%	$2^{43.99}$	$2^{37.59}$
(L9)	$[5,3]$	18	0.0%	–	–
(L10)	$[5,3]$	9	95.83%	$2^{40.32}$	$2^{34.0}$
(L11)	$[5,4]$	17	0.83%	$2^{43.58}$	$2^{35.6}$
(L12)	$[5,5]$	19	0.0%	–	–
(L13)	$[5,5]$	16	0.0%	–	–
(L14)	$[5,5]$	9	82.5%	$2^{41.43}$	$2^{34.95}$
(L15)	$[5,5]$	19	0.0%	–	–
(L16)	$[5,5]$	17	0.0%	–	–
(L17)	$[8,2]$	5	100.0%	$2^{35.53}$	$2^{29.17}$
(L18)	$[8,7]$	9	78.75%	$2^{41.56}$	$2^{34.79}$
(L19)	$[8,7]$	5	100.0%	$2^{35.41}$	$2^{29.42}$
(L20)	$[8,8]$	8	79.17%	$2^{41.59}$	$2^{35.78}$
(L21)	$[8,8]$	4	100.0%	$2^{34.76}$	$2^{29.29}$

Table 4: Type, depth, proportion of runs finished after two days of computations, the average number of nodes traversed, and the number of remaining keys for Dumbo (L1), and LFSRs (L2) to (L21) [20].

From these experiments, it seems that the depth has a much more relevant impact than the type. Yet, this seems to be quite tailored to our particular attack. Changing the generation of the different masks is generally more impactful, since it can cut down in three the amount of information given to the attacker.

6 Conclusion

In this paper and its previous version, theoretical and simulated practical blind side-channel attack targeting the LFSR of the Elephant algorithm have been presented. Elephant is a pertinent target. First, Elephant is a finalist for the (NIST) competition for lightweight cryptography candidates for authenticated encryption. Moreover, Elephant is an interesting target because the internal LFSR only depends on the secret key. In other words, in the use case of Elephant, retrieving the initial state of the LFSR is equivalent to recovering the encryption key.

Different tweaking options have been considered. Going from the most impactful to the least, they are changing the mask derivation for domain separation, and modifying the LFSR, looking at the importance of depth and type.

Our attack is based on the fact that an attacker can retrieve the Hamming weights of the different bytes in the LFSR. The Elephant design, where there exist relations between the different masks of the LFSR, is an added vulnerability to our attack. In the previous version, in half the cases, the key is retrieved in less than two days. In this paper, an important improvement is made. The result is that in three quarters of the cases, the key is retrieved in less than two days. To have this new result the main idea has been that the attacker can wait and decide when to begin the attack. In other words the attacker has to find the best attack position in the LFSR computation progress.

Future works may consider the inclusion of noise in the simulations. To succeed we need a new tool able to treat errors resulting noisy measurement Hamming weight. An idea is to use a belief propagation as in [26]. Ultimate future work can be performing the attack on an actual implementation.

Acknowledgments This research is part of the chair CyberCNI.fr with support of the FEDER development fund of the Brittany region and with APCIL project fund of the Brittany region too.

References

1. Archambeau, C., Peeters, E., Standaert, F., Quisquater, J.: Template attacks in principal subspaces. In: Cryptographic Hardware and Embedded Systems-CHES. Springer (2006)
2. Azouaoui, M., Papagiannopoulos, K., Zürner, D.: Blind side-channel sifa. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE
3. Beierle, C., Biryukov, A., dos Santos, L.C., Großschädl, J., Perrin, L., Udovenko, A., Velichkov, V., Wang, Q., Biryukov, A.: Schwaemm and esch: lightweight authenticated encryption and hashing using the sparkle permutation family. NIST round **2** (2019)
4. Bernstein, D.J.: How to Stretch Random Functions: The Security of Protected Counter Sums. J. Cryptol. (1999)

5. Bertoni, G., Daemen, J., Peeters, M., van Assche, G.: The Keccak Reference (2011)
6. Beyne, T., Chen, Y.L., Dobraunig, C., Mennink, B.: Dumbo, Jumbo, and Delirium: Parallel Authenticated Encryption for the Lightweight Circus. IACR Transactions on Symmetric Cryptology (2020)
7. Beyne, T., Chen, Y.L., Dobraunig, C., Mennink, B.: Elephant v2. NIST lightweight competition (2021)
8. Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., Verbauwhede, I.: Spongent: a Lightweight Hash Function. In: CCryptographic Hardware and Embedded Systems-CHES. Springer (2011)
9. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Cryptographic Hardware and Embedded Systems-CHES. Springer (2004)
10. Burman, S., Mukhopadhyay, D., Veezhinathan, K.: LFSR based stream ciphers are vulnerable to power attacks. In: INDOCRYPT. Lecture Notes in Computer Science, vol. 4859, pp. 384–392. Springer (2007)
11. Chakraborty, A., Mazumdar, B., Mukhopadhyay, D.: Fibonacci LFSR vs. galois LFSR: which is more vulnerable to power attacks? In: SPACE. Lecture Notes in Computer Science, vol. 8804, pp. 14–27. Springer (2014)
12. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Cryptographic Hardware and Embedded Systems-CHES. Springer (2002)
13. Clavier, C., Reynaud, L.: Improved blind side-channel analysis by exploitation of joint distributions of leakages. In: International Conference on Cryptographic Hardware and Embedded Systems. pp. 24–44. Springer (2017)
14. Clavier, C., Reynaud, L., Wurcker, A.: Quadrivariate improved blind side-channel analysis on boolean masked aes. In: International Workshop on Constructive Side-Channel Analysis and Secure Design. pp. 153–167. Springer (2018)
15. Daemen, J., Hoffert, S., Peeters, M., Assche, G.V., Keer, R.V.: Xoodyak, a lightweight cryptographic scheme (2020)
16. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon. Submission to the CAESAR competition (2014)
17. Granger, R., Jovanovic, P., Mennink, B., Neves, S.: Improved Masking for Tweakable Blockciphers with Applications to Authenticated Encryption. In: EUROCRYPT. Springer (2016)
18. Handschuh, H., Heys, H.M.: A timing attack on rc5. In: International Workshop on Selected Areas in Cryptography. Springer (1998)
19. Hell, M., Johansson, T., Maximov, A., Meier, W., Yoshida, H.: Grain-128aead, round 3 tweak and motivation (2021)
20. Houssein Meraneh, A., Clavier, C., Le Bouder, H., Maillard, J., Thomas, G.: Blind side channel on the elephant lfsr. In: SECRYPT (2022)
21. Iwata, T., Khairallah, M., Minematsu, K., Peyrin, T.: Duel of the titans: the romulus and remus families of lightweight aead algorithms. IACR Transactions on Symmetric Cryptology (2020)
22. Joux, A., Delaunay, P.: Galois LFSR, Embedded Devices and Side Channel Weaknesses. In: INDOCRYPT (2006)
23. Jurecek, M., Bucek, J., Lórencz, R.: Side-channel attack on the a5/1 stream cipher. In: Euromicro Conference on Digital System Design (DSD). IEEE (2019)
24. Kazmi, A.R., Afzal, M., Amjad, M.F., Abbas, H., Yang, X.: Algebraic side channel attack on trivium and grain ciphers. IEEE Access (2017)
25. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Advances in Cryptology - CRYPTO. Springer (1999)

26. Le Bouder, H., Lashermes, R., Linge, Y., Thomas, G., Zie, J.: A Multi-round Side Channel Attack on AES Using Belief Propagation. In: Foundations and Practice of Security. Springer (2016)
27. Linge, Y., Dumas, C., Lambert-Lacroix, S.: Using the joint distributions of a cryptographic function in side channel analysis. In: International Workshop on Constructive Side-Channel Analysis and Secure Design. Springer (2014)
28. Luykx, A., Preneel, B., Tischhauser, E., Yasuda, K.: A MAC Mode for Lightweight Block Ciphers. In: Peyrin, T. (ed.) Fast Software Encryption FSE. Springer (2016)
29. Mangard, S.: A simple power-analysis (spa) attack on implementations of the aes key expansion. In: ICISC. Springer (2002)
30. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks: Revealing the secrets of smart cards, vol. 31. Springer (2008)
31. NIST: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. FIPS 202 (2015)
32. NIST: Lightweight Cryptography Standardization Process (2018)
33. Ouladj, M., Guilley, S.: Side-Channel Analysis of Embedded Systems. Springer (2021)
34. Rechberger, C., Oswald, E.: Stream ciphers and side-channel analysis. In: In ECRYPT Workshop, SASC-The State of the Art of Stream Ciphers. Citeseer (2004)
35. Samwel, N., Daemen, J.: DPA on hardware implementations of ascon and keyak. In: Proceedings of the Computing Frontiers Conference. ACM (2017)
36. Standaert, F.X.: Introduction to side-channel attacks. In: Secure integrated circuits and systems. Springer (2010)
37. Yli-Mäyry, V., Ueno, R., Miura, N., Nagata, M., Bhasin, S., Mathieu, Y., Graba, T., Danger, J.L., Homma, N.: Diffusional side-channel leakage from unrolled lightweight block ciphers: A case study of power analysis on prince. IEEE Transactions on Information Forensics and Security (2020)