Blind Side Channel Analysis against AEAD with a Belief Propagation Approach

Hélène Le Bouder, Modou Sarry, Eïd Maalouf, Gaël Thomas

▶ **To cite this version:**

Hélène Le Bouder, Modou Sarry, Eïd Maalouf, Gaël Thomas. Blind Side Channel Analysis against AEAD with a Belief Propagation Approach. CARDIS (Smart Card Research and Advanced Application Conference), Nov 2023, Amsteram, Netherlands. hal-04293086

HAL Id: hal-04293086
https://hal.science/hal-04293086

Submitted on 18 Nov 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Blind Side Channel Analysis against AEAD with a Belief Propagation Approach

Modou Sarry[1], Hélène Le Bouder[1], Eïd Maaloouf[1], and Gaël Thomas[2]

[1] IMT-Atlantique, OCIF, IRISA, Rennes, France
[2] DGA Maîtrise de l'Information, Bruz, France

Keywords: authenticated encryption with associated data (AEAD), side channel analysis (SCA), blind side channel analysis (BSCA), belief propagation (BP), Sparkle, Alzette, LFSR, Elephant

Abstract. This paper present two new attacks on two lightweight authenticated encryption with associated data (AEAD): Sparkle and Elephant. These attacks are blind side channel analysis (BSCA). The leakage is considered as an Hamming weight (HW) with a Gaussian noise. In both attacks, a belief propagation (BP) algorithm is used to link the different leaks. Another objective is to present BSCA as a new tool for evaluating the robustness of a symmetric cryptographic primitive subfunctions.

1 Introduction

Today, the Internet of things (IoT) interconnects every aspect of our lives, leading to a proliferation of embedded systems. These devices play an essential role in many sectors such as industry, intelligent transport, smart cities and healthcare, to name but a few. However, due to the sensitive nature of the data they process, it is imperative to ensure the security of these systems. Protecting confidential information and preventing cyber-attacks are major concerns when deploying these circuits. Securing these systems is therefore of crucial importance.

In this context, the National Institute of Standards and Technology (NIST) started a competition [1] for the standardization of a lightweight authenticated encryption with associated data (AEAD).

Physical attacks rely on the interaction of the computing unit with the physical environment. They are mainly divided in two families: side channel analysis (SCA) and fault injection attacks. SCA [2] are based on observations of the circuit behaviour during the computation. They exploit the fact that some physical values of a circuit depend on intermediary values of the computation. This is the so-called leakage of information of the circuit. Blind side channel analysis (BSCA) is a new family of SCA, where no inputs nor outputs are used to recover a cryptographic key, only the traces and the knowledge of the algorithm is used.

Motivation Many attacks already exist on older symmetric cryptography algorithms like the advanced encryption standard (AES) [3]. Since many AEAD algorithms are relatively new, there are far fewer attacks against them [4,5,6,7]. Moreover, these new algorithms are often designed to be more resistant to physical attacks than AES. Therefore, one motivation is to test the resistance of these new algorithms against SCA.

Another main motivation is to build an attack which uses only traces, no text and no profiling. We are in the case of an attacker who can just observe a leakage but has no access to the device's input/output, hence the name of blind side channel analysis (BSCA). Once has to remark that retrieving the key without the ciphertext in BSCA may seem absurd in the context of confidentiality. However, in the context of integrity, an attacker could forge a tag for their message after obtaining the secret key.

The last motivation is to show that the BSCA context can be seen as a new tool to evaluate cryptosystems. This method is a good way to select the subfunction of an algorithm at a mathematical level. Thinking about the security of the cryptographic algorithm is an important approach.

Contribution In this paper, we evaluate the security of two AEADs: Elephant [8] and Sparkle [9] against BSCA. The attack on Elephant is a major improvement of a first BSCA [10,11]. The attack on Sparkle is a new attack. The results are based on a belief propagation (BP) algorithm [12].

Organization Targeted algorithms Sparkle and Elephant are described in section 2. State of the art of BSCA attacks are summarised in section 3. Our attack methodology is described in section 4. The application of this attack against the Elephant algorithm and the corresponding results are presented in section 5. Similarly, the application of this attack against the Sparkle algorithm is discussed in section 6. Finally, the conclusion is drawn in section 7.

2 Authenticated encryption with associated data (AEAD)

2.1 NIST Competition

Authenticated encryption with associated data (AEAD), illustrated in Figure 1, should ensure confidentiality and integrity. It takes as input different parameters: a plaintext denoted M, the associated data denoted A, a secret key K, and an initialisation vector N also called a nonce. The nonce is public but must be different for each new plaintext. The algorithms ensure confidentiality of the plaintext M and integrity of both the plaintext M and the associated data A. It returns a ciphertext C and a tag T.

In this context, the NIST started the competition [1] for lightweight cryptography candidates for AEAD. In August 2019, NIST received 57 submissions to be considered for standardization. In March 2021, only 10 finalists remained

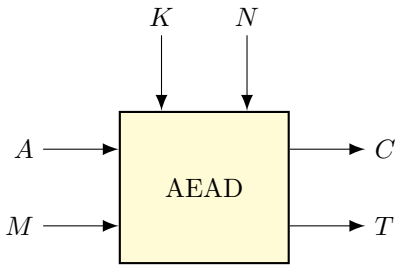

Fig. 1. Overview of AEAD.

of which Elephant [8] and SPARKLE [9] which are targeted in this paper. Finally, in February 2023, NIST decided to standardize ASCON [13].

2.2 Elephant

Elephant [8,14] was a finalist to the NIST lightweight cryptography competition. It is a nonce-based AEAD. Its construction is based on an Encrypt-then-MAC that combines counter (CTR) mode encryption with a variant of the protected counter sum [15,16]. Elephant uses a cryptographic permutation masked with linear feedback shift registers (LFSRs) in an Even-Mansour-like fashion [17] in place of a block cipher.

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

$$\mathsf{mask}_K^{j,i} = (\varphi \oplus \mathsf{id})^i \circ \varphi^j \circ \mathsf{P}(K||0^{n-128}); \tag{1}$$

where id is the identity function. One important thing to note is that the values of $\mathsf{mask}_K^{j,i}$ depend only on the secret key K and not on any other input of Elephant.

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 outputs a ciphertext C as large as M, and a τ-bit tag T. The algorithm is depicted on Figure 2.

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

In this paper, we focus on the smallest and main instance Dumbo. It uses the 160-bit permutation Spongent-π[160] [18], the 160-bit LFSR φ_{Dumbo}, and has tag size $\tau = 64$ bits.

Let $\lll$, $\ll$, and $\gg$ denote the left rotation, left shift, and right shift operators respectively. The update equation of LFSR φ_{Dumbo} is given at the byte level by equation (2) and illustrated on Figure 3.

$$\varphi_{\mathsf{Dumbo}} : (x_0, \cdots, x_{19}) \mapsto (x_1, \cdots, x_{19}, x_0 \lll 3 \oplus x_3 \ll 7 \oplus x_{13} \gg 7) \tag{2}$$

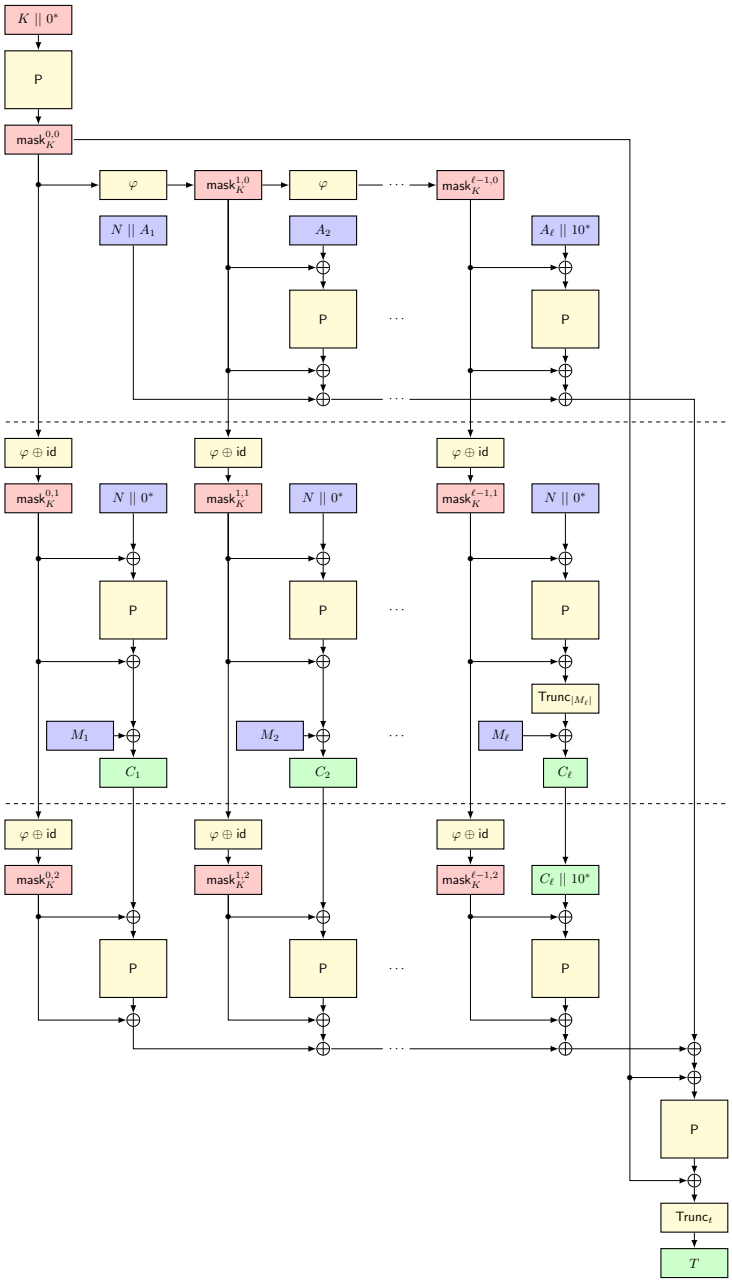

Fig. 2. Elephant associated data authentication (left), plaintext encryption (middle), and ciphertext authentication (right). This figure comes from [10,11] according to the description of Elephant [8].

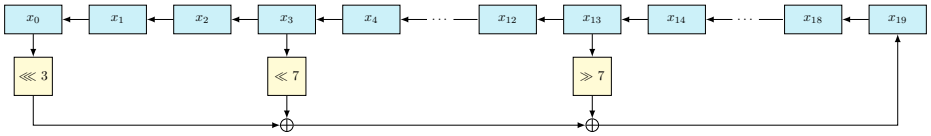

Fig. 3. The 160-bit LFSR φ_{Dumbo}. This figure comes from [10,11] according to the description of Elephant [8].

2.3 Sparkle

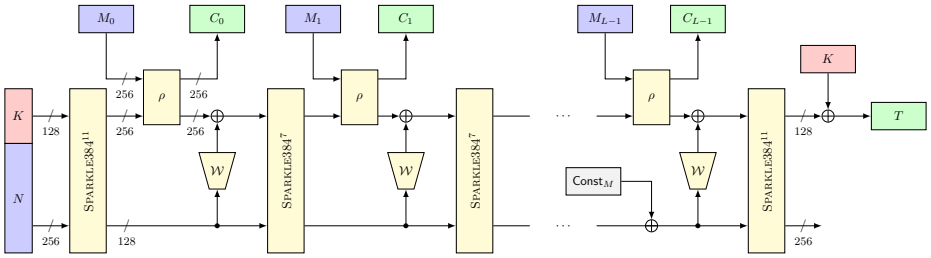

Fig. 4. Encryption using SCHWAEMM256-128, where SPARKLE384^R is the SPARKLE384 permutation with R rounds, ρ is a linear application, $\mathcal{W}$ the linear whitening layer, and the constant Const_M indicating whether the plaintext M was padded or not.

SCHWAEMM is the AEAD of the SPARKLE [9] submission to the NIST lightweight cryptography competition. It is a permutation-based AEAD that builds upon the Beetle [19] variant of the Duplex [20] construction by adding an extra linear whitening layer $\mathcal{W}$, from the inner state into the outer state. The underlying cryptographic permutation is named SPARKLE which gives the whole submission its name.

Several instances of both algorithms with various performance and security trade-offs are proposed.

The main instance is SCHWAEMM256-128. The rest of this paper then focuses on this particular instance. It is illustrated in Figure 4.

SCHWAEMM256-128 takes a 256-bit nonce N, a 128-bit key K, and produces a 128-bit authentication tag T. The corresponding SPARKLE permutation is the 384-bit-wide variant, denoted SPARKLE384.

Zoom on Sparkle The SPARKLE permuations are based on a substitution permuation network (SPN) structure. The linear layer is a Feistel-type transformation operating on 64-bit branches. It is based on a linear function $\mathcal{M}$ whose description is not necessary for the understanding of this paper. The non-linear

layer is a 64-bit S-box named Alzette $\mathcal{A}_i$, applied in parallel to every branch of the Feistel, where i denotes the index of the branch. More details on Alzette are given in the paragraph 2.3.

The number of rounds depends on the particular instance, and whether it is used in the initialisation, encryption or tag generation phase of Schwaemm. In the case of Schwaemm256-128, the number of rounds is $R = 11$ for the initialisation and tag generation, and only $R = 7$ for encryption. The first round of Sparkle384 of the initialisation phase of Schwaemm256-128 is depicted on Figure 5. One important thing to note is that the state is initialised with the concatenation of the nonce N and the secret key K.

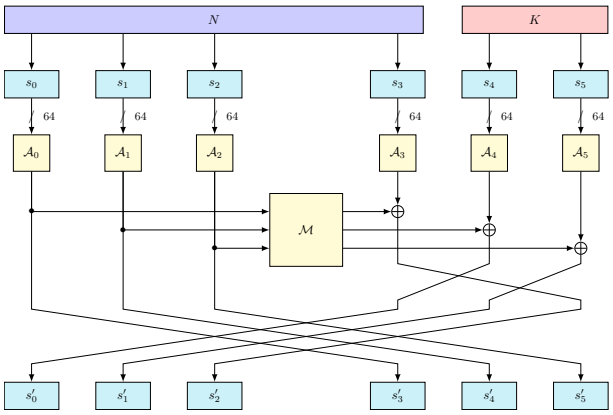

Fig. 5. The first round of the Sparkle384 permutation during the initialisation of Schwaemm256-128.

Alzette The Alzette S-box $\mathcal{A}_i$ is a 64-bit permutation parametrised by a 32-bit known value α_i that depends only on the index i of branch inside the Sparkle permutation. It is built as a 4-round addition-rotation-xor (ARX) Feistel cipher with all round keys equal to α_i. As the name ARX suggests, it uses a combination of 32-bit modular additions, rotations, and xors. The whole process is depicted on Figure 6.

3 BSCA Context

While it has been proven that an algorithm is mathematically secure, its implementation can still expose vulnerabilities known as physical attacks. Physical attacks, such as SCA, constitute a specific subcategory of these vulnerabilities. They exploit the fact that certain physical characteristics of a device depend on intermediate values of the calculations performed, resulting in an information

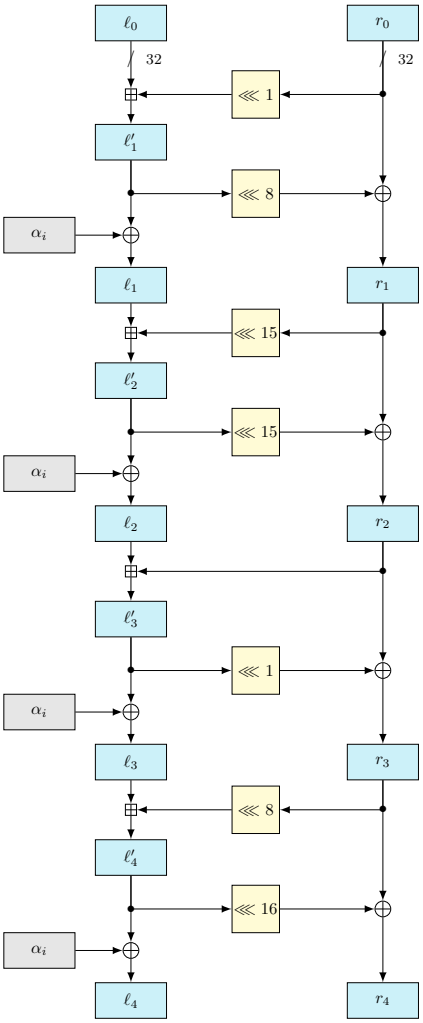

Fig. 6. The Alzette S-box $\mathcal{A}_i$ used in SPARKLE.

leakage within the circuit. This information leakage could be utilized to retrieve secrets, such as a secret key.

Many SCA belong to the family of correlation power analysis (CPA) [21,22,23]. To succeed these attacks need data like plaintext or ciphertext. Then these attacks use a mathematical model for the leakage. A confrontation is performed between measurements and predictions built with the model and known data. More precisely, a statistic tool called distinguisher gives score to the different targets.

In the case of sponge and duplex building, to the best of our knowledge, there are very few CPA attacks [24]. In a classical block cipher, the secret key is directly used in each round. In a duplex or sponge case, the secret key is used only once or twice, at the start and at the end of the whole algorithm. The first difficulty is caused by this fact. Moreover, in the specific use case SPARKLE, there is no small S-box. On the contrary, Alzette is a very big S-box. That is one other reason, we have not chosen to study CPA attacks on SPARKLE, but blind SCA.

BSCA is a type of SCA where some information that is usually known is unknown. The main concept is to perform the attack solely based on leakage measurements. The attacker model is different from that in a classic CPA. They have access to less information. Often BSCA is based on a strong assumption: the attacker is supposed to retrieve a noisy HW from the leakage.

Linge *et. al* [25] have presented the first BSCA. The goal is to attack the block cipher AES without using data such as plaintext or ciphertext. At the same time, Le Bouder *et. al* [26] published a first attack. Then, these works have been improved by Clavier *et al.* [27]. Moreover, their contribution introduces for the first time, the name of blind side channel analysis (BSCA). Now it is a new family of SCA, and different symmetric cryptography algorithms are targeted by these attacks such as AES [25,26,27,28], Elephant [10], and PRINCE [29].

4 Description of the attacks

4.1 Simulated leakage Model

The power consumption or electromagnetic leakages are very correlated to the Hamming weight (HW) of the data. It is a classical model used in the domain of SCA.

One important advantage of HW is that it rapidly reduces guesses. For example, let x be a byte, so x can take 256 values in $[\![0, 255]\!]$. With the HW of x, the attacker reduces the list of possible values, as shown in Table 1.

Table 1. Number of possible values for a byte x according its HW. This table comes from [26]

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

Another model for the leakage is a HW with an additive Gaussian noise. For a given discrete random variable byte x, and its HW(x) $\in [\![0, 8]\!]$; $\tilde{\text{HW}}(x)$ denotes the continuous random variable representing the simulated Hamming weight so called **noisy**; defined in $\mathbb{R}$ as:

$$\tilde{\text{HW}}(x) = \text{HW}(x) + \sigma_{x,t} \quad ; \tag{3}$$

with $\sigma_{x,t}$ an event of the Gaussian random variable $\mathcal{N}\left(0, \sigma^2\right)$ at a time t. The probability density function $\mathbf{F}$ associated to $\mathcal{N}\left(0, \sigma^2\right)$ is given by:

$$\mathbf{F}_\sigma(x) = \frac{1}{\sigma \cdot \sqrt{2\pi}} \cdot \exp\left(-\frac{1}{2} \cdot \left(\frac{x}{\sigma}\right)^2\right) \quad . \tag{4}$$

In this paper, BSCA are studied, so the attacker uses only simulated leakage. The simulation of the power consumption or electromagnetic leakage is generated with Gaussian noise added to the HW obtained from the intermediate computations at the byte level. To simplify the simulation, it is assumed that the noise level remains the same throughout the system. The simulations, which were performed using the Python programming language (v3.9), represent a typical unprotected implementation on an 8-bit processor.

4.2 Belief propagation (BP)

In our attack approach, relations between the different values for which a nosy HW is simulated, and exploited. For that belief propagation (BP) is used. BP was first used by Gallager [30,31] for decoding low-density parity-check (LDPC) codes [32]. It was then rediscovered by Tanner [33] and formalized by Pearl [34]. The first time that BP was used in SCA on symmetric encryption, in the attack of Veyrat-Charvillon *et al.* [35], then it is studied in [36,26].

Tanner graph A BP algorithm relies on a bipartite graph called a factor graph (or Tanner graph). To each node in the factor graph is associated some information. The nodes of a factor graph as are of two kinds:

- variable nodes V representing the variables handled by the algorithm under attack;
- factor nodes, representing the equations E between these variables.

An edge links a variable node V with a factor node E, when the equation represented by the factor node E involves the variable node V. An example of a factor graph is illustrated in Figure 7.

N denotes the set of neighbours for a node. Thus the set $\mathsf{N}(E)$ is made of the variable nodes V involved in equation E, and the set $\mathsf{N}(V)$ is made of factor nodes E that depend on V.

BP algorithm The belief propagation (BP) algorithm inputs are

- the Tanner graph,
- prior probabilities $\mathbb{P}_A(V = v)$ on the different variable nodes V.

The BP algorithm returns a better belief, from the initial values $\mathbb{P}_A(V = v)$. More precisely, BP computes probabilities a posteriori $\mathbb{P}_P(V = v)$ on the different variable nodes according prior probabilities.

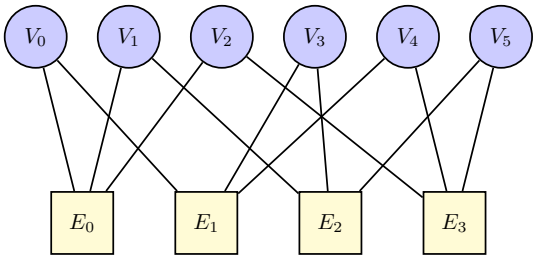

Fig. 7. Example of a Tanner factor graph part. Circles are variable nodes and squares are factor nodes.

The values $\mathbb{P}_P(V = v)$ are computed according to the input prior probability $\mathbb{P}_A(V = v)$ and to the probabilities $\mathbb{P}(V = v|E)$ conditional on factor nodes $E \in \mathsf{N}(V)$ to be satisfied using the following equation:

$$\mathbb{P}_P(V = v) \propto \mathbb{P}_A(V = v) \times \prod_{\mathcal{E} \in \mathsf{N}(V)} \mathbb{P}(V = v|E) \tag{5}$$

In practice, nodes in the factor graph exchange information messages with their neighbours. More precisely, since the graph is bipartite, two types of messages are exchanged:

- factor to variable messages between a factor node E and a variable node V, denoted $\mu_{E \to V}$:

$$\mu_{E \to V}(v) = \sum_{v_1, v_2, \cdots, v_{\#\mathsf{N}(E)-1}} E(v, v_1, \cdots, v_{\#\mathsf{N}(E)-1}) \times \prod_{V_i \in \mathsf{N}(E) \setminus \{V\}} \mu_{V_i \to E}(v_i) \tag{6}$$

  This equation comes from the law of total probability. Roughly speaking, the quantity $\mu_{E \to V}(v)$ represents the belief from the perspective of factor node E that variable node V take the value v. The information is gathered from the other variables nodes in $\mathsf{N}(E)$.
- variable to factor messages between a variable node V and a factor node E, denoted $\mu_{E \to V}$.

$$\mu_{V \to E}(v) \propto \mathbb{P}_A(V = v) \times \prod_{E' \in \mathsf{N}(V) \setminus \{E\}} \mu_{E' \to V}(v) \tag{7}$$

  This equation is essentially Bayes' rule. Here variable node V updates its belief using informations from the factor nodes in $\mathsf{N}(V)$. This equation is meant to be used in alternance with Equation (6). So the contribution from node E is deliberately excluded to avoid self-persuasion.

Both equations make independence assumptions to make computations feasible in practice at the expanse of the risk to converge to an incorrect value.

To complete the description of the BP algorithm, an initialization step is done before applying the above equations. The variable to factor messages $\mu_{V \to E}(v)$

are initialized with the prior probabilities $\mathbb{P}_A(V = v)$. In summary, after an initialization phase, BP works by alternatively applying equations (6) then (7) for every edge (V, E) in the Tanner graph. At the end of the execution, the returned value $\mathbb{P}_P(V = v)$ is computed using equation (5) where the probabilities $\mathbb{P}(V = v|E)$ are replaced by their approximations $\mu_{E\to V}(v)$. The number of iterations depends of the rapidity of convergence, which in turn depends on the Tanner graph.

5 Attack on Elephant

5.1 Attack path

The attack path is based on a similar approach as presented in the theoretical work by Meraneh *et al.* [10,11]. However, their attack focuses on obtaining HW information and is limited to scenarios without noise. It is a theoretical and mathematical assumption that is very useful for testing ideas, but it never happens in the real world. In this paper, we improve this attack to consider noisy HW more close than real measurements and propose the use of BP as a solution to address this challenge.

Linear feedback shift registers (LFSRs) find applications in various lightweight cryptography candidates, where their initial state is typically determined by both a key and a nonce. Since the nonce must be altered for each encryption request, attacks against such schemes are confined to the decryption algorithm. However, in the case of Elephant, the LFSR solely relies on the secret key. As a result, our attack can be employed in an encryption scenario as well.

The objective of the described attack is to recover the secret initial state of the LFSR. Three crucial points should be noted as following.

- 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 recovers 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 Figure 3) is the most vulnerable one: the following of this section is focused on Dumbo.

Since the LFSR generates a single new byte at each iteration, let the content of the Dumbo LFSR be denoted as follows:

$$(x_j, \cdots, x_{j+19}) = \mathsf{mask}_K^{j,0} \tag{8}$$

and let x_{j+20} be the byte generated at iteration j. Similarly, let $(y_j, \cdots, y_{j+19}) = \mathsf{mask}_K^{j,1}$ and $(z_j, \cdots, z_{j+19}) = \mathsf{mask}_K^{j,2}$. By definition of mask, the following hold

for all $j \geq 0$:

$$y_j = x_j \oplus x_{j+1} \tag{9}$$
$$z_j = x_j \oplus x_{j+2}. \tag{10}$$

The attacker can thus exploit two attack vectors: on the one hand, equations (2) coming from iterating the LFSR, and on the other hand, equations (9) and (10) coming from the different masks used for domain separation.

In this use case, the attacker obtain a leakage, simulated by noisy HW on the different bytes of the Dumbo LFSRs.

Once has to remark that a classic CPA is impossible on the $\mathsf{mask}_K^{0,0}$, because the attacker knows zero data in this function.

5.2 Tanner graph

In the case of Elephant, the Tanner graph is defined as follows. Variable nodes are the bytes x_j, y_j and z_j of the LFSRs. There are four types of factor nodes:

- $E_{y_0}, E_{y_1}, \ldots, E_{y_{20}}$ represent the equation (9).
- $E_{z_0}, E_{z_1}, \ldots, E_{z_{19}}$ represent the equation (10).
- We have divided the feedback equation (2) into two sub-equations. This helps reduce the amount of computation in BP. Indeed, the number of terms in equation (6) is exponential in the number of neighbours in $\mathsf{N}(E)$. The factor nodes corresponding to equation (2) are given in Table 2.

The Elephant Tanner graph is illustrated in Figure 8.

Table 2. Intermediate and final feedback equation

intermediate feedback equation	final feedback equation
$E_{tx_{20}} : tx_{20} = (x_0 \lll 3) \oplus (x_3 \ll 7)$	$E_{ux_{20}} : x_{20} = tx_{20} \oplus (x_{13} \gg 7)$
$E_{tx_{21}} : tx_{21} = (x_1 \lll 3) \oplus (x_4 \ll 7)$	$E_{ux_{21}} : x_{21} = tx_{21} \oplus (x_{14} \gg 7)$
$E_{ty_{20}} : ty_{20} = (y_0 \lll 3) \oplus (y_3 \ll 7)$	$E_{uy_{20}} : y_{20} = ty_{20} \oplus (y_{13} \gg 7)$

5.3 Results

Table 3 presents the average and median rank of the correct key byte for various noise levels σ. The statistics are done for all 20 bytes and repeated with 1000 different randomly generated keys.

Results indicate that when the noise level is low, BP helps pushing up the ranks of the correct key bytes. An attacker can then leverage the posterior probabilities returned by BP using smart key enumeration techniques to try and find the whole key. Note that since the LFSR computation only depend on the key, it is possible to reduce the noise level by averaging several traces.

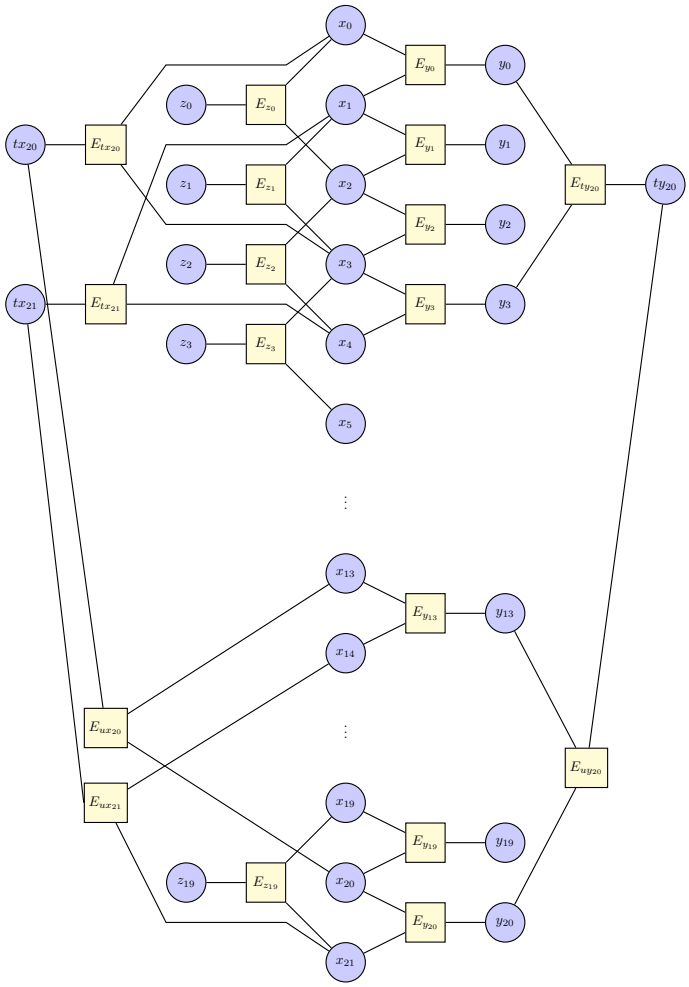

Fig. 8. Tanner factor graph on LFSR Dumbo of Elephant.

Another important thing to note is that, finding the correct value of *any* $\mathsf{mask}_K^{j,i}$, not just the first one, counts as a successful attack. Depending on the actual value of $\mathsf{mask}_K^{j,i}$, some may be easier to attack than others, as was already noticed by Meraneh *et al.* [10] in the noise-free case.

6 Attack on Sparkle

6.1 Attack path

During the initialisation phase of Schwaemm256-128, the key K is loaded into the last 128 bits of the Sparkle384 state, see Figure 4. Let split $K = K_1||K_2$

Table 3. Rank of all key bytes on 1000 different keys of 20 bytes for different noise levels σ. More precisely, BP algorithm returns probabilities a posteriorifor each possible value of each byte. These probabilities can be sorted from most likely to least likely. This table calculates the rank of the correct value for each byte.

σ	Mean	Standard Deviation	Min	Quartile Q1	Median	Quartile Q3	Max
0.1	3.54	5.97	0	0	3	3	27
0.15	3.54	5.97	0	0	3	3	27
0.2	3.67	6.11	0	0	3	3	31
0.25	4.95	9.31	0	0	3	3	97
0.3	6.00	10.76	0	0	3	3	97
0.35	7.46	13.10	0	0	3	8	97
0.4	9.59	15.72	0	0	3	8	97
0.5	15.97	23.03	0	3	8	31	153
0.6	23.65	31.41	0	3	8	31	157
0.7	33.73	40.11	0	3	27	36	213
1	70.68	61.42	0	31	36	92	246

into two 64-bits halves. Then, in the first round, K_1 is the input of Alzette $\mathcal{A}_4$, and K_2 of Alzette $\mathcal{A}_5$.

The rest of this section only describes the attack for a single Alzette since they work exactly the same but for the constant α.

As stated in subsection 2.3, Alzette is a 4-round ARX Feistel cipher, illustrated in see Figure 6. In this use case, the attacker can obtain simulated leakage on the different internal values ℓ_i, ℓ'_i, and r_i of Alzette. Here again, we assume a leakage model as a noisy HW of the bytes, as described in equations (3) and (4). The leakage observed in the first round of Alzette is depicted in Figure 9.

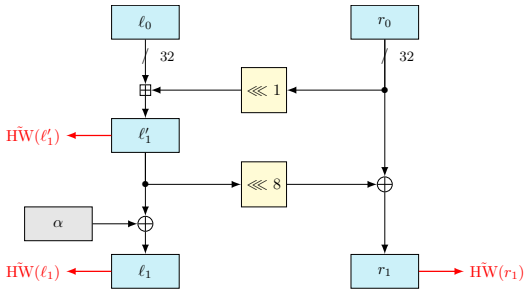

Fig. 9. Measurement leakage in the first round of Alzette described in section 2.3 and Figure 6.

The different internal values related to each other thanks to the following Alzette equations:

$$\begin{array}{lll} \ell'_1 = \ell_0 \boxplus (r_0 \lll 1) & , \quad \ell_1 = \ell'_1 \oplus \alpha & , \quad r_1 = (\ell'_1 \lll 8) \oplus r_0 \\ \ell'_2 = \ell_1 \boxplus (r_1 \lll 15) & , \quad \ell_2 = \ell'_2 \oplus \alpha & , \quad r_2 = (\ell'_2 \lll 15) \oplus r_1 \\ \ell'_3 = \ell_2 \boxplus r_2 & , \quad \ell_3 = \ell'_3 \oplus \alpha & , \quad r_3 = (\ell'_3 \lll 1) \oplus r_2 \\ \ell'_4 = \ell_3 \boxplus (r_3 \lll 8) & , \quad \ell_4 = \ell'_4 \oplus \alpha & , \quad r_4 = (\ell'_4 \lll 16) \oplus r_3 \end{array} \tag{11}$$

To the best of our knowledge, this attack is the first BSCA against Sparkle.

6.2 Tanner graph

In the case of Sparkle, the variable node in the Tanner graph are defined at the bit level. They are the bits of the different ℓ_i, ℓ'_i, and r_i. for $0 \leq i \leq 4$. Since modular additions are involved, a 31-bit carry variable c_i is also added at each round.

As for the equations, they are of two kinds:

- the bitwise translation of equations (11) which describe Alzette,
- the HW equations (sum of bits gives the HW).

In this case, the HW are represented in the graph because they induce new relations between the different bits. The Figure 10 shows a part of the Tanner graph of the first round of Alzette.

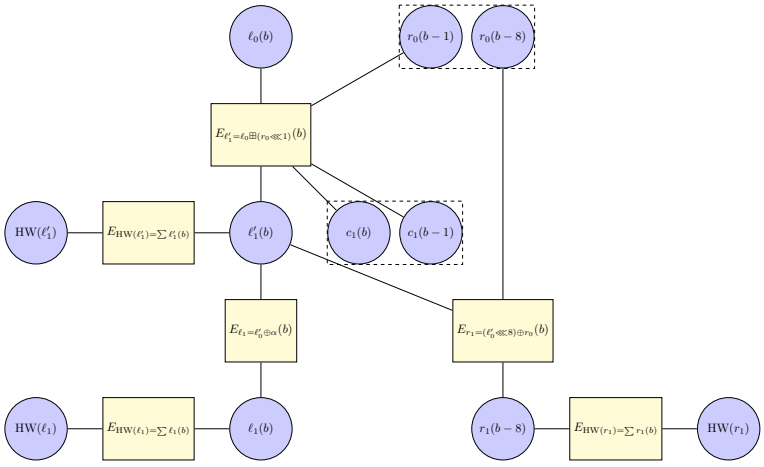

Fig. 10. Tanner graph of the first round of Alzette. Only a single bit with index $b \in [\![0, 31]\!]$ of each 32-bit register is shown. Relative indices are given $\mod 32$. The carries in the modular operation are denoted $c_1(b)$ and $c_1(b-1)$. Bits belonging to the same 32-bit register are enclosed in dashed lines.

6.3 Results

We ran our attack on both Alzettes in a parallel manner, the only difference being the α_4 constant used by $\mathcal{A}_4$ and α_5 used by $\mathcal{A}_5$.

For the different values of the noise σ and with a single simulated measurement, the number of correct decisions on the input bits, or the number of recovered bits was calculated. The experiment was repeated on 1000 uniformly drawn random keys.

Table 4 presents the number of recovered bits of the 64-bit K_1 input of Alzette $\mathcal{A}_4$. Table 5 does the same for K_2 and $\mathcal{A}_5$. Finally, Table 6 presents the number of bits recovered for the whole 128-bit master key K of SPARKLE.

Table 4. Number of bits recovered for the 64-bit key K_1 of Alzette $\mathcal{A}_4$.

σ	Mean	Standard Deviantion	Min	Quartile Q1	Median	Quartile Q3	Max
0.1	57.08	3.02	36	56	57	59	63
0.15	57.16	2.66	38	56	57	59	63
0.2	57.20	2.57	40	56	57	59	64
0.25	57.15	2.76	38	56	57	59	64
0.3	57.07	2.74	40	56	57	59	64
0.35	56.77	2.94	36	55	57	58	64
0.4	56.63	2.81	39	55	57	58	64
0.45	56.12	3.01	35	55	56	58	64
0.5	55.81	2.81	39	54	56	57	64
0.6	54.83	2.94	36	53	55	56	64
0.7	54.19	2.68	36	53	54	56	62
1	52.86	3.47	35	52	54	55	59

Table 5. Number of bits recovered for the 64-bit key K_2 of Alzette $\mathcal{A}_5$.

σ	Mean	Standard Deviantion	Min	Quartile Q1	Median	Quartile Q3	Max
0.1	56.98	2.78	36	56	57	59	63
0.15	57.05	2.40	39	56	57	59	63
0.2	57.05	2.43	39	56	57	59	63
0.25	56.96	2.55	39	55	57	59	63
0.3	56.82	2.53	40	55	57	58	63
0.35	56.59	2.64	39	55	57	58	64
0.4	56.29	2.67	40	55	56	58	64
0.45	55.92	2.75	37	54	56	58	64
0.5	55.53	2.66	38	54	55	57	64
0.6	54.64	2.70	36	53	55	56	63
0.7	53.86	2.79	36	53	54	55	61
1	52.78	3.29	33	52	54	54	59

Table 6. Number of bits recovered for the 128-bit master key K.

σ	Mean	Standard Deviation	Min	Quartile Q1	Median	Quartile Q3	Max
0.1	114.06	4.07	89	112	114	117	123
0.15	114.22	3.59	92	112	114	116	123
0.2	114.26	3.51	97	112	114	117	124
0.25	114.11	3.76	92	112	114	116	124
0.3	113.89	3.67	97	112	114	116	125
0.35	113.36	3.94	90	111	113	116	124
0.4	112.92	3.82	95	111	113	115	124
0.45	112.05	4.09	89	110	112	115	123
0.5	111.35	3.85	92	109	111	114	123
0.6	109.48	3.97	89	108	110	112	121
0.7	108.06	3.77	87	106	108	110	120
1	105.65	4.65	85	105	107	108	115

First, note that as in the case of the attack on Elephant, it is possible to reduce the noise level by averaging several traces, since the variables handled only depend on the key.

Second, it is important to consider that the correct key is often not the first attempt, as there may be a few unknown bits for the attacker. Therefore, allowing the attacker to perform an exhaustive search can be relevant to uncover the remaining bits. That is why we grant the attacker a certain computational power.

Let n be the number of bits the attacker can flip on 64 bits. This means that they can try up to $\mathcal{P}$ keys where $\mathcal{P}$ is given by:

$$\mathcal{P} = \binom{64}{0} + \binom{64}{1} + \binom{64}{2} + \cdots + \binom{64}{n}.$$

Since they must do this for both half-keys, the total number of keys they can guess is $\mathcal{P}^2$. Table 7 indicates the number of keys found out of 1000 with an attacker possessing a computing power of $\mathcal{P}^2$ as a function of the number of bits n they can flip in a reasonnable amount of time.

It is essential to point out that this exhaustive search does not exploit the a posteriori probabilities returned by BP. Therefore, the use of smarter key enumeration techniques, exploiting these probabilities, will certainly lead to improved results without the need for a specific computing power to perform an exhaustive search. Future projects include the use of key enumeration.

7 Conclusion

In this paper, two attacks on two lightweight AEAD algorithms have been successfully conducted and for each attack we used a single simulated measurement.

Firstly, an improvement on the attack against Elephant [10] incorporating a noisy Hamming weight model is presented. This enhanced approach allows for

Table 7. Number of keys, out of 1000, found for both Alzettes by an attacker with computational power $\mathcal{P}^2$, that can exhaustively flip up to n bits in each 64-bit subkey.

	$n=4, \mathcal{P}^2=2^{39}$		$n=5, \mathcal{P}^2=2^{46}$		$n=6, \mathcal{P}^2=2^{53}$		$n=7, \mathcal{P}^2=2^{59}$		$n=8, \mathcal{P}^2=2^{65}$	
σ	$\mathcal{A}_4$	$\mathcal{A}_5$	$\mathcal{A}_4$	$\mathcal{A}_5$	$\mathcal{A}_4$	$\mathcal{A}_5$	$\mathcal{A}_4$	$\mathcal{A}_5$	$\mathcal{A}_4$	$\mathcal{A}_5$
0.1	175	169	274	283	494	422	610	586	831	787
0.15	174	168	273	281	492	420	611	585	831	786
0.2	175	172	284	285	485	421	611	586	818	776
0.25	171	170	284	280	470	417	612	566	801	740
0.3	177	133	284	246	447	389	582	554	751	716
0.35	155	136	248	229	368	365	540	512	703	664
0.4	137	105	243	187	368	306	523	468	652	618
0.45	110	84	180	167	285	265	431	407	584	559
0.5	88	53	154	112	246	216	380	342	527	495
0.6	34	23	86	51	132	116	225	213	360	326
0.7	9	7	25	18	65	43	150	100	266	211
1	0	0	2	1	15	6	47	35	119	88

intelligent backtracking of the key, increasing the chances of finding the correct value.

Next, the first BSCA on Sparkle has been described. By exploiting the attacker computational power, we have shown that the attack can indeed recover a significant portion of the secret keys when the noise level is moderate. Using backtracking on the key in an intelligent way that could improve results without doing an exhaustive search for the few remaining bits not predicted by our attack.

The power of the BP algorithm was also highlighted in our research, prompting us to develop a generic tool for it in the future. In future work, we can speed up the computation time of BP by using Walsh-Hadamard transform [37].

Our future research has to focus on transitioning these attacks into practical implementations and to develop a key backtracking algorithm that is intelligent with posterior probabilities. Additionally, we plan to explore the security of Ascon, the NIST competition winner, as a potential target for our future investigations.

Acknowledgments This research is part of the APCIL project found by the Brittany region. The authors would like to thank Laurent Toutain.

References

1. NIST. *Lightweight Cryptography Standardization Process*, 2018.
2. Maamar Ouladj and Sylvain Guilley. *Side-Channel Analysis of Embedded Systems*. Springer, 2021.
3. NIST. Specification for the Advanced Encryption Standard. *FIPS PUB 197*, 2001.

4. Francesco Berti, Shivam Bhasin, Jakub Breier, Xiaolu Hou, Romain Poussier, François-Xavier Standaert, and Balasz Udvarhelyi. A Finer-Grain Analysis of the Leakage (Non) Resilience of OCB. *IACR T CHES*, 2022.
5. Siang Meng Sim, Dirmanto Jap, and Shivam Bhasin. Differential analysis aided power attack on (non-) linear feedback shift registers. *IACR TCHES*, 2021.
6. Alexandre Adomnicai, Laurent Masson, and Jacques JA Fournier. Practical algebraic side-channel attacks against acorn. In *Inscrypt*. Springer, 2019.
7. Valentina Banciu, Elisabeth Oswald, and Carolyn Whitnall. Exploring the resilience of some lightweight ciphers against profiled single trace attacks. In *COSADE*. Springer, 2015.
8. Tim Beyne, Yu Long Chen, Christoph Dobraunig, and Bart Mennink. Elephant v2. *NIST lightweight competition*, 2021.
9. Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, Qingju Wang, and Alex Biryukov. Schwaemm and esch: lightweight authenticated encryption and hashing using the sparkle permutation family. *NIST round*, 2, 2019.
10. Awaleh Houssein Meraneh, Christophe Clavier, Hélène Le Bouder, Julien Maillard, and Gaël Thomas. Blind Side Channel On The Elephant LFSR. 2022.
11. Julien Maillard, Awaleh Houssein Meraneh, Modou Sarry, Christophe Clavier, Hélène Le Bouder, and Gaël Thomas. Blind side channel analysis on the Elephant LFSR Extended version. *SECRYPT BOOK*, 2023.
12. David Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 04-2011 edition, 2011.
13. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon. *Submission to the CAESAR competition*, 2014.
14. Tim Beyne, Yu Long Chen, Christoph Dobraunig, and Bart Mennink. Dumbo, Jumbo, and Delirium: Parallel Authenticated Encryption for the Lightweight Circus. *IACR Transactions on Symmetric Cryptology*, 2020.
15. Daniel J. Bernstein. How to Stretch Random Functions: The Security of Protected Counter Sums. *J. Cryptol.*, 1999.
16. Atul Luykx, Bart Preneel, Elmar Tischhauser, and Kan Yasuda. A MAC Mode for Lightweight Block Ciphers. In Thomas Peyrin, editor, *FSE*. Springer, 2016.
17. Robert Granger, Philipp Jovanovic, Bart Mennink, and Samuel Neves. Improved Masking for Tweakable Blockciphers with Applications to Authenticated Encryption. In *EUROCRYPT*. Springer, 2016.
18. Andrey Bogdanov, Miroslav Knezevic, Gregor Leander, Deniz Toz, Kerem Varici, and Ingrid Verbauwhede. Spongent: a Lightweight Hash Function. In *CHES*. Springer, 2011.
19. Avik Chakraborti, Nilanjan Datta, Mridul Nandi, and Kan Yasuda. Beetle Family of Lightweight and Secure Authenticated Encryption Ciphers. *IACR TCHES.*, 2018.
20. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In *SAC*. Springer, 2011.
21. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO*. Springer, 1999.
22. Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *CHES*. Springer, 2004.
23. Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In *CHES*. Springer, 2008.

24. Niels Samwel and Joan Daemen. DPA on hardware implementations of ascon and keyak. In *Computing Frontiers Conference*. ACM, 2017.
25. Yanis Linge, Cécile Dumas, and Sophie Lambert-Lacroix. Using the joint distributions of a cryptographic function in side channel analysis. In *COSADE*. Springer, 2014.
26. Hélène Le Bouder, Ronan Lashermes, Yanis Linge, Gaël Thomas, and Jean-Yves Zie. A Multi-round Side Channel Attack on AES Using Belief Propagation. In *FPS*. Springer, 2016.
27. Christophe Clavier and Léo Reynaud. Improved blind side-channel analysis by exploitation of joint distributions of leakages. In *CHES*. Springer, 2017.
28. Christophe Clavier, Léo Reynaud, and Antoine Wurcker. Quadrivariate improved blind side-channel analysis on boolean masked aes. In *COSADE*. Springer, 2018.
29. Ville Yli-Mäyry, Rei Ueno, Noriyuki Miura, Makoto Nagata, Shivam Bhasin, Yves Mathieu, Tarik Graba, Jean-Luc Danger, and Naofumi Homma. 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.
30. Robert G. Gallager. Low-density parity-check codes. *IRE Trans. on Information Theory*, 1962.
31. Robert G. Gallager. *Low Density Parity check codes*. PhD thesis, MIT, Cambridge, MA, 1963.
32. Sae-Young Chung, G. David Forney Jr., Thomas J. Richardson, and Rüdiger L. Urbanke. On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit. *IEEE Communications Letters*, 2001.
33. Robert M. Tanner. A recursive approach to low complexity codes. *IEEE Trans. on Information Theory*, 1981.
34. Judea Pearl. Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach. In *National Conference on Artificial Intelligence*. AAAI Press, 1982.
35. Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft Analytical Side-Channel Attacks. In *ASIACRYPT 2014*.
36. Vincent Grosso and François-Xavier Standaert. ASCA, SASCA and DPA with Enumeration: Which One Beats the Other and When? In *ASIACRYPT 2015*, pages 291–312. Springer.
37. Wanli Ouyang and Wai-Kuen Cham. Fast algorithm for walsh hadamard transform on sliding windows. *Transactions on pattern analysis and machine intelligence*, 2009.