A New Leakage Exploitation Framework and Its Application to Authenticated Encryption

Vahid Jahandideh, Léo Weissbart, Bart Mennink, and Lejla Batina

Radboud University, Nijmegen, The Netherlands
{v.jahandideh, l.weissbart, b.mennink, lejla}@cs.ru.nl

Abstract. We target a 32-bit XOR instruction with a secret input and a known random operand and recover the secret with around 10K traces. Based on the leakage of this linear instruction, we propose a framework for power analysis of the unprotected software implementations of finalists of the NIST lightweight cryptography competition. The approach applies to (almost) all finalists and clarifies which details in their design enable successful power analysis attacks. Furthermore, the diversity of the studied ciphers in this work elucidates how mode and primitive design can help to mitigate leakage without demanding a heavy masking countermeasure that conflicts with the lightweight design goals.

1 Introduction

Power analysis attacks are a practical concern for ciphers' real-world security, especially in embedded devices designed for lightweight applications. In these devices, power traces are relatively easy to capture after gaining physical access to the target device. Given enough power consumption traces for an unprotected cipher, various side-channel (SC) attacks can recover hidden secrets effectively [17,8,27,31]. These attacks are classified into *differential* and *simple* power analysis (DPA and SPA, respectively). Normally, DPA relies on (a) divide-and-conquer and needs (b) sufficiently many traces. SPA attacks are used when (a) is not applicable or (b) does not hold. DPA is generally easier to apply and more successful in practice. To withstand SC attacks, specifically DPA attacks, implementation-level protections such as *masking* and *hiding* are commonly used [19]. Masking, at order d, at least in theory, ensures that any collection of power samples with at most $d-1$ members cannot convey usable leakage about the running cryptographic algorithm [16]. This immunity comes at the cost of an $\mathcal{O}(d^2)$ blowup in the computational load of the running code and consequently results in a penalty on performance. Considering the energy and power budget, there may be no room in lightweight battery-powered setups to opt for high values of d (e.g., $d \geq 3$). On the other hand, in practice, low-order masking is successfully attacked with device-dependent *micro-architectural* leakages rooted in the hidden interactions and couplings of the registers [2,28]. Furthermore, advanced SPA, such as *horizontal attacks*, demonstrate that some commonly used masking gadgets, such as the ISW gadget for AND operation [16,25], are insecure irrespective of the deployed masking order d [4].

Addressing these and many other challenges in the implementation-level leakage mitigation, new cryptographic algorithms are highly desired to contribute to SC security at the *mode* level. This mode-level security, starting from the initial proposal, was one of the objectives of the NIST LWC contest in the lightweight cryptography domain for designing authenticated encryption with associated data (AEAD) [21]. The competition is currently over, and ASCON [11] is selected as the winner[1] after several years of being under scrutiny. In this work, we compare the side-channel properties of ASCON with some of its fellow finalists. As our main contribution, we lay a framework to clarify to which extent an AEAD mode is (in)secure in case of DPA key/plaintext recovery attacks, with the primary goal of identifying prominent features in their construction that can help to achieve more SC-aware AEAD designs in future.

1.1 Related works on SC analysis of AEAD ciphers

Tailored for AEAD constructions, Guo et al. [14] and Berti et al. [6] augmented *CCA indistinguishability* and *ciphertext integrity* (CI) definitions with a comprehensive set of assumptions for leakage and nonce reuse/misuse. Later, Bellizia et al. [5] looked into the implications of these leakage-bearing definitions for some NIST LWC candidates. Their work notably clarified which parts of the various AEAD constructions should be protected against DPA. This work got recently expanded to include more ciphers [30]. However, the DPA discussion in these papers is an overview and needs practical/implementation details. On the experimental side, successful applications of SC attacks for some NIST candidates are reported in individual projects [32,26,36,15,24]. From the countermeasures side, with a notable impact on the computational load of masking protection, Pereira et al. [22] proposed to design ciphers such that not all parts demand the same level of protection. This property is called *leveled* implementation. It is also referred to as *uniform* vs. *non-uniform* masking [5]. The idea is that, in a uniform case, the whole cipher needs to be masked, contrasted to the non-uniform case where masking only some sensitive segments is enough. Particularly, ASCON, in the face of key recovery DPA, needs only non-uniform masking. However, for plaintext recovery DPA attacks, costly uniform protection is unavoidable. Recently, following a NIST SC evaluation proposal, many groups/labs took part in hardware/software analysis of the third-round finalists of NIST LWC to evaluate T-test/DPA/masking on them with frequently updated reports.[2]

1.2 Our contributions

We perform improved SC analysis of software implementations of ASCON along with some of the third-round NIST LWC finalists. To do so, we propose a generic DPA framework that applies to almost all of the candidates. This approach clarifies how (in some of the considered ciphers) details, such as parameter choice,

[1] https://csrc.nist.gov/Projects/lightweight-cryptography

[2] https://cryptography.gmu.edu/athena/index.php?id=LWC

simplicity of key schedule, or invertibility of the permutation path, make DPA attacks possible or more accessible. To build the DPA framework as broadly as possible, we stick to the simple linear XOR instruction instead of seeking non-linear operations such as S-box. In practice, linear operations (alone) are not a primary choice for DPA attacks because they are expected to leak less than non-linear operations. Moreover, if some bit-positions are not leaking, higher leakage of other bit-positions cannot compensate. With this in mind, our contributions are as follows.

- We target an isolated XOR operation with one secret input and one known random operand. We demonstrate that, given enough traces, there are various SC techniques for extracting the secret operand. More precisely, we apply and compare *correlation power analysis* (CPA), *linear regression* (LR), *deep learning* (DL), and their combinations.
- We use this XOR observation to argue the NIST candidates' DPA (in)security. The basic assumption is that if w-bit chunks of an n-bit target secret are processed in $\frac{n}{w}$ separate XOR operations, each with a known random operand, the attacker can learn that secret.
- We explore the unprotected optimized code of the AEAD candidates to locate (if existing) a set of XOR operations for key/plaintext recovery.
- For the case of ASCON, we demonstrate the applicability of the proposed model in our experimental setup.
- The most significant part of our contribution is that we make clear which property in the underlying unprotected ciphers is helping the feasibility of a DPA attack. For example, in the case of Xoodyak, we show that a high absorption rate and invertible permutation chain make DPA possible, or in the case of Grain128-AEAD, bit-oriented implementation enables the attack. For GIFT-COFB, we demonstrate that the attack uses the simplicity of the key schedule. Considering the diversity of studied ciphers, our discussions clarify how future designs can be more SC-aware.
- We additionally provide a table summarizing our SC analysis for studied AEAD ciphers.

1.3 Structure of this paper

Section 2 gives our framework for power analysis attacks. Section 3 explains the generic construction of AEAD ciphers. In Section 4, we discuss the power analysis of ASCON and some of its competitors. DPA for plaintext recovery is discussed in Section 5. Finally, in Section 6, we give a summary of the results.

2 Leakage modeling and exploitation

The first step of a power analysis attack is to represent the exploitable leakage with a mathematical model. For software implementations, power leakage is attributed mostly to the bit-flips of the operands (registers) in the *instruction-level* description of the running algorithm. It assumes that power samples in a trace are linear mixes of power consumption of successive instructions of the running assembly code, and the power consumption of each instruction is a noisy function of its operands [28,20]. Details of the employed Instruction Set Architecture (ISA) should be known for conducting a case study. In this work, we use the ARMv7-M architecture. This platform is a common choice for SC investigations. It contains 16 shift registers, each of which holds a 32-bit number. In the listings presented in this paper, the shift registers are labeled as $\{\texttt{r0}, ..., \texttt{r9}, \texttt{sl}, \texttt{fp}, \texttt{ip}, \texttt{sp}, \texttt{lr}, \texttt{pc}\}$. The operations that manipulate these registers for carrying a proper computation are instructions. Typically, each line of an assembly code describes an instruction `INST` that works with a specified set of registers [34].

2.1 Recovering secret parameters with leakage

We demonstrate that leakage of XOR instruction with a known and random operand p and a secret operand k contains a noisy correlation with $k \oplus p$ that can reveal k. Depending on the trace sampling rate and the clock frequency of the target device, the time interval of execution of an XOR instruction contains m samples. Let us denote these samples for the ith execution with $\mathbf{T}_i$. So, $\mathbf{T}_i = \{T_i[1], \ldots, T_i[m]\}$ is a trace. In an experimental setup for a singled-out XOR with N traces, we use SC techniques such as CPA, LR, and DL followed by key-ranking to extract the correct k from $\{p_i, \mathbf{T}_i\}_{i=1}^{N}$ collection.

2.2 DPA attack with XOR

We use the assembly code in Listing 1.1 that helps set apart XOR's leakage by enough leading and trailing `nop` operations. During a `nop` instruction, the processor ideally performs no activity on the registers. With enough `nop` operations, we are sure that the transition effects of the trigger signals will be damped. `eor.w` and `eors` both denote XOR operation.

```
...        ; r3 = k   r9 = p
8000aa8: bf00           nop
8000aaa: ea89 0903      eor.w r9, r9, r3 ;   r9 = r9 ⊕ r3
8000aae: bf00           nop
```

Listing 1.1: Assembly code for XOR leakage.

Attack with CPA. In the Hamming weight (HW) leakage model, samples in $\mathbf{T}$, in specific instances (also known as Points of Interest (PoI)), are assumed to contain HW of the processed intermediates with some additive Gaussian noise. For the case of our experiment, we assume that there is at least one PoI index in $\mathbf{T}$, denoted by random variable l, which for some noise n and constant a can be described as $l = a\mathsf{HW}(p \oplus k) + n$. If $p[j]$ and $k[j]$ for $0 \leq j \leq 3$ represent the corresponding bytes of p_i and k, we can write:

$$l = a \sum_{j=0}^{3} \mathsf{HW}(p[j] \oplus k[j]) + n.$$

Values of $\mathsf{HW}(p_i[j] \oplus k[j])$ for different bytes are independent of each other. Thanks to this property, we conduct a CPA attack with a collection of N traces with 32-bit random p to learn k.

The CPA attack can consider all samples in $\mathbf{T}$ without worrying about the selection of PoIs. For attack, we compute the following m-dimensional vector of empirical correlation coefficients $\mathbf{C}_j[k^*] = \mathsf{cor}(\mathsf{HW}(k^* \oplus p[j]), \mathbf{T})$ for all hypotheses $0 \le k^* \le 255$ [19]:

$$\mathbf{C}_j[k^*] = \frac{\sum_{i=1}^{N}(\mathsf{HW}(k^* \oplus p_i[j]) - 4) \cdot (\mathbf{T}_i - \bar{\mathbf{T}}))}{\sqrt{\sum_{i=1}^{N}(\mathsf{HW}(k^* \oplus p_i[j]) - 4)^2 \cdot \sum_{i=1}^{N}(\mathbf{T}_i - \bar{\mathbf{T}})^2}}, \tag{1}$$

where m-dimensional $\bar{\mathbf{T}}$ is the sample average of the traces. In the computations of $\mathbf{C}_j[k^*]$ for byte j, the value of the other bytes of k are unknown and are treated as noise. The correct $k[j]$ is assumed to be the k^* corresponding to the largest value in $256 \times m$ matrix $\mathbf{C}_j$. The same traces can be used to recover all the key bytes.

The drawback of this approach. Estimating the key in the given byte by byte method lowers the computation load compared to the direct exhaustive search on the 2^{32} possible k candidates. However, the penalty paid is the increase in the number of required traces to compensate for the noise effect of non-targeted key bytes.

Attack with LR. The HW leakage l of a w-bit variable v in terms of its bits is $l = a\mathsf{HW}(v) + n = a\sum_{j=1}^{w} v[j] + n$ where index j runs on bits. In practice, this model can be fine-tuned as $l = \sum_{j=1}^{w} a_j v[j] + n$ by finding a fitting set of coefficients a_j [27]. The method used for approximating a_js is linear regression, and the estimation quality improves as the number of measurements N increases.

In the case of the 32-bit XOR, for leakage of the output variable, ignoring noise n, we can write:

$$\begin{aligned} l = \sum_{j=1}^{32} a_j(k[j] \oplus p[j]) &= \sum_{k[j]=0} a_j p[j] + \sum_{k[j]=1} a_j(1 - p[j]) \\ &= \sum_{k[j]=0} a_j p[j] - \sum_{k[j]=1} a_j p[j] + b, \end{aligned} \tag{2}$$

for some constant b independent of p [13]. In the attack scenario, an adversary unaware of k can approximate a set of coefficients a'_j with linear regression for the following estimation:

$$l_i \approx \sum_{j=1}^{w} a'_j p_i[j] + b',$$

where index i runs over different traces, and l_i is the value of a single sample (PoI) in the traces. The main observation is that sign of a'_j will reveal the value of

$k[j]$: positive (resp. negative) a'_j means that $k[j]$ equals 0 (resp. 1). a_j values are typically positive; however, for more caution, it is better to run the experiment with a known key once and record the signs of a_js. We need to select a PoI in the traces to conduct this attack. Usually, PoI is selected by examining all samples in $\mathbf{T}$ to peak one that minimizes the variance of residual noise after estimation.

Attack with DL. Inspired by [18], for the XOR experiment, as our third attack approach, we use a multilayer perceptron (MLP) deep learning (DL) network to approximate $\mathsf{HW}(k \oplus p)$ with the knowledge of p and $\mathbf{T}$. In our neural network, the input nodes are m (normalized) samples in $\mathbf{T}$ and 32 bits of p with relu as the activation function. The network has one hidden dense layer again with relu. There are 33 output nodes to cover all possible HW values. The activation for the output layer is sigmoid. See [35] for the methodology of neural networks in SC works. We used 100K pairs of random $(k_i, p_i, \mathbf{T}_i)$ for training and testing the network, with the cross-entropy metric as the loss function. The search for the network parameters was done by trial and error. The code for the network is in Python and developed with Keras API.[3]

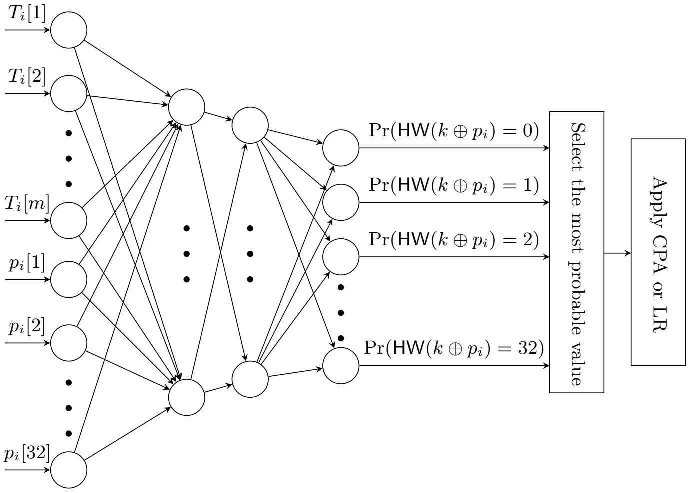

$T_i[1]$
$T_i[2]$
$T_i[m]$
$p_i[1]$
$p_i[2]$
$p_i[32]$
$\Pr(\mathsf{HW}(k \oplus p_i) = 0)$
$\Pr(\mathsf{HW}(k \oplus p_i) = 1)$
$\Pr(\mathsf{HW}(k \oplus p_i) = 2)$
$\Pr(\mathsf{HW}(k \oplus p_i) = 32)$
Select the most probable value
Apply CPA or LR

Fig. 1: MLP schematic used for estimation of $\mathsf{HW}(k \oplus p_i)$ with the knowledge of $\{p_i, \mathbf{T}_i\}$, where the $T_i[j]$s are samples in trace $\mathbf{T}_i$, and the $p_i[j]$s are bits of p_i.

After the training phase, for conducting an attack, the 32-bit search space is big, so we cannot directly use probabilities computed by the neural network model. See [35] for the typical neural network model in the attack phase. To mitigate this challenge, as depicted in Figure 2, we propose to use the model's output combined with CPA and LR. For inputs $\mathbf{T}_i$ and p_i, the most probable

[3] `https://keras.io`

output of the model is interpreted as an estimation of $\mathsf{HW}(k \oplus p_i)$. This estimation will be the input of CPA or LR attack. In this sense, the constructed MLP model mainly tries to denoise the targeted HW value.

2.3 Experimental results

Our SC evaluation setup is a Chipwhisperer CW308 UFO board,[4] with an STM32F405 32-bit microcontroller running at the clock frequency of 7.37 MHz. The trace collection is with Chipwhisperer lite at a synchronized rate of four times the clock frequency.

In this setup, with 10K measurements, we could recover the value of all the key bytes with high certainty for the XOR operation. In these bytes, the correct value had a rank of one or two. See Figure 2 for the results. The constructed DL model effectively decreases the required traces for both CPA and LR.

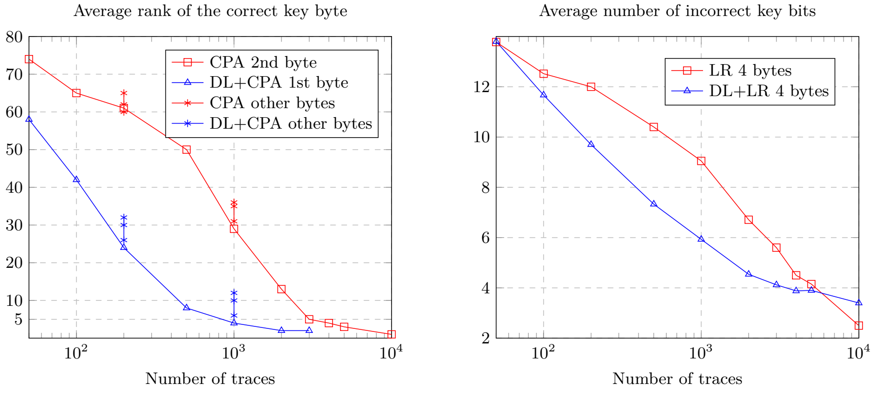


Fig. 2: (Left) CPA with and without DL. The CPA attack for the 2nd byte produces slightly better results. DL+CPA in the used model worked better for the 1st byte. Results for other bytes are also shown. (Right) LR with and without DL.

For the presented byte by byte CPA attack, optimally ranking the key given the ranking of its bytes is not a trivial problem [23]. However, in our proof of concept experiments in this work, this is not a concern. Instead, we assume that the rank of the key is roughly the multiplication of the ranks of its bytes. Whereas, for the results reported in Figure 2, for the case of an LR attack, our metric differs from CPA and is the average number of bit positions in the 32-bit key that are estimated correctly.

[4] `https://github.com/newaetech/chipwhisperer`

2.4 Limitations of our model

The assumption that the attacker can separate the leakage of a single instruction requires complete knowledge of the running assembly code that might not always be available. Overlapping leakages and pipeline effects will likely increase the number of required traces for a successful attack. However, for the particular case of ASCON, we will demonstrate the possibility of this attack model in our experimental setup.

3 Structure of AEAD ciphers

We define a prototype AEAD scheme X and describe its algorithms and input-output parameters as a generic interface such that all candidates are regarded as different instantiations. This approach unifies discussions about the various ciphers and abstracts common explanations in one place.

Notation. A binary string of length $m \in \mathbb{N}$ belongs to $\{0,1\}^m$, and $\{0,1\}^*$ represents arbitrary length strings. The same pattern applies to unary strings (e.g., 0^m). With $0^\dagger$, we mean the concatenation of an adequate number of zeros. Lengths are always in bits, and the length of string B is $|B|$. To partition a string B into blocks of r bits, we write $(B_1, \ldots, B_t) = \mathsf{Split}(B, r)$, where the last block, i.e., B_t, can be of size less than r bits. For a string B of size $\geq m$, $\mathsf{Trunc}(B, m)$ denotes the first m bits of B, starting from the leftmost bit. The concatenation of two strings, B and C, is denoted as $B \,\|\, C$. For a times shift to the left (resp. to the right), we write $B \ll a$ (resp. $B \gg a$). Bitwise NOT and OR operations are shown with $\sim$ and $|$, respectively. With π, we denote a cryptographic permutation $\{0,1\}^n \to \{0,1\}^n$, where n is called the state size. We refer to a rounds of application of π as π^a. Usually, a is a *security parameter*. To clarify instances of π, we append -p at the end of their names (e.g., Keccak-p).

A prototype authenticated encryption scheme. For cipher X, we denote by X-Encryption the corresponding encryption algorithm. Inputs to X-Encryption are a key K, nonce N, message $M \in \{0,1\}^*$, and associated data $A \in \{0,1\}^*$. Outputs of X-Encryption are a ciphertext C with $|C| \geq |M|$ and tag T. It is possible that a separate X-Authentication with K, N, A, and C, generates T. In either case, T is responsible for validating both C and A. During decryption of a received tuple (N, A, C, T), X-Decryption will check the validity of T. If T is found valid, the corresponding message M will be the output. Otherwise, a failure notice will be the output of the decryption.

Attack assumptions. Following [1], we assume the strongest possible attacker: it has access to encryption and decryption devices and knows their running assembly codes. So, it can encrypt any tuple (N, A, M) and record the associated leakage. It can also try to decrypt any tuple (N, A, C, T) and receive the leakage.

This means that an attack may fall in a chosen nonce scenario. Encryption and decryption devices may try to impose nonce uniqueness. Nevertheless, it is much more difficult for decryption devices since old ciphertexts should still be decryptable. Keeping this in mind, for each presented attack, we make clear whether the attacker needs a chosen nonce (including nonce-reuse) or a random nonce. We have not examined the case of non-fixed nonce such as a counter. However, most of the discussed attacks will fail with non-uniform randomness.

4 Power analysis of selected NIST LWC finalists

For the candidates, we study their unprotected official ($\mathsf{GCC}\ -\mathsf{O2}$) optimized ARMv7-M assembly code and try to identify a set of XOR instructions for applying DPA key recovery attack of Section 2.1. A masking countermeasure consideration will follow the discussion for each cipher, with the sole objective of identifying whether the cipher can be protected with non-uniform masking or not. However, we will not consider details of masking protection or their effectiveness in withstanding the proposed attacks.

4.1 Power analysis of ASCON

ASCON is announced as the winner of the NIST LWC contest. It is also one of the finalists of the CAESAR competition and is recommended for lightweight applications.[5] It has two instances that both claim 128-bit security. The instances differ in the choice of internal parameters [11]. This cipher is sponge-based and uses ASCON-p with state size $n = 320$ for its cryptographic permutation. ASCON-p is suitable for optimized bit-sliced implementation. Some other NIST LWC candidates (e.g., ISAP) also rely on this permutation. The ASCON specification states that its design prevents internal state recovery attacks from leading to the disclosure of K or forgery of T. Inside the assembly code of ASCON, we point to several XOR instructions that fulfill the DPA requirements (see Section 2.1). The target secret for all of these DPAs is K.

Composition of ASCON. Key, nonce, and tag are each of size 128 bits. Plaintext and associated data are separately partitioned into r-bit blocks. The plaintext is always padded with one 1 and an adequate number of 0s, even if $|M| = 0$, such that $|(M\|1\|0^{\dagger})|$ is a multiple of r. In this way, we have $(M_1, \ldots, M_t) = \mathsf{Split}(M\|1\|0^{\dagger}, r)$. If associated data is present, i.e., if $|A| > 0$, it will be padded similarly: $(A_1, \ldots, A_s) = \mathsf{Split}(A\|1\|0^{\dagger}, r)$. Ciphertext C is also in blocks of r bits and has the same length as M. ASCON-Encryption, as depicted in Figure 3, is responsible for generating both C and T. For one instance of ASCON, the parameters are $(a, b, r) = (12, 6, 64)$. For the other one, they are $(a, b, r) = (12, 8, 128)$. For both of them, the 64-bit IV is $(128_8 \parallel r_8 \parallel a_8 \parallel b_8 \parallel 0_{32})$, where the subscripts denote the number of bits that are used for the left-MSB

[5] https://competitions.cr.yp.to/caesar-submissions.html

representation. It is relevant to our SC discussion that the permutation π is easy to invert. However, both encryption and decryption need only forward computation of it. We skip detailing ASCON-Decryption since it trivially follows the construction of ASCON-Encryption with an appropriate swap of plaintext and ciphertext blocks.

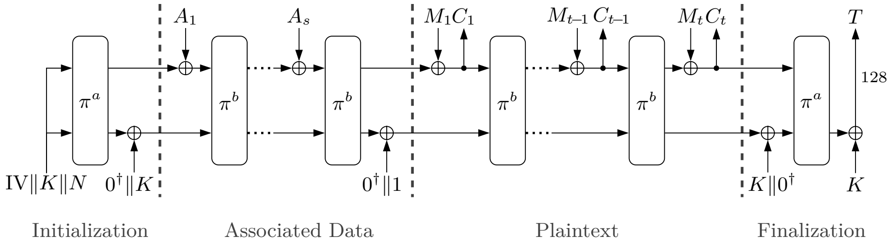


Fig. 3: ASCON-Encryption. a and b denote the number of permutation rounds.

DPA possibilities. We are looking for instructions combining secret intermediates with known random values. Inside the assembly code of ASCON, there are various instances where such instructions show up.

Initialization phase of ASCON-Encryption*.* In this phase, N and K are merged inside permutation π. This early involvement of N makes it possible to find instructions accepting bits of N and bits of K as their operands. Listing 1.2 is part of the computation for the first round of π^a in the initialization.

```
...   ; r0 holds address of the parameters
10b60: e9d0 1205   ldrd  r1, r2, [r0, #20]; r1 = K2, r2 = N1
10b64: e9d0 7601   ldrd  r7, r6, [r0, #4] ; r7 = IV0, r6 = K1
10b68: e9d0 5403   ldrd  r5, r4, [r0, #12]; r5 = K0, r4 = K3
...   ; r1,r2,r4,r5,r6,r7 are not touched
10b7e: e9d0 ec08   ldrd  lr, ip, [r0, #32]; lr = N3, ip = N2
10b82: 69c3        ldr   r3, [r0, #28]    ; r3 = N0
10b84: f8d0 8000   ldr.w r8, [r0]         ; r8 = IV1
10b88: f084 04f0   eor.w r4, r4, #240     ; K3 = K3 ⊕ 0xf0
10b8c: ea86 0904   eor.w r9, r6, r4       ; r9 = K1 ⊕ K3
10b90: ea88 0a0e   eor.w sl, r8, lr       ; sl = IV1 ⊕ N3
10b94: ea82 0b0e   eor.w fp, r2, lr       ; fp = N1 ⊕ N3
10b98: ea62 0e0e   orn   lr, r2, lr       ; lr = N1|(∼ N3)
10b9c: ea8e 0e09   eor.w lr, lr, r9   ; lr = K1 ⊕ K3 ⊕ (N1|(∼ N3))
10ba0: ea82 0206   eor.w r2, r2, r6       ; r2 = N1 ⊕ K1
```

Listing 1.2: Inside the initialization of ASCON-Encryption.

In this listing, we represent 32-bit words of N and K, for $0 \leq i \leq 3$, with `Ni` and `Ki`, respectively. For words of the 64-bit IV, we use `IV0` and `IV1`. At address

0X10ba0 in Listing 1.2, N1 and K1 are operands of one eor (XOR) instruction with relation $\texttt{r2} = \texttt{N1} \oplus \texttt{K1}$. With the assumption that the attacker can choose N1 uniformly random, this instruction fulfills the requirements of DPA to recover K1. After recovering K1, the instruction at address 0X10b9c can be the target (with the same traces) for recovering K3. In the subsequent instructions, there are similar XOR operations for attacking the remaining words of K. Samwel and Daemen [26], with a Hamming leakage model for an experimental setup with a more complex operation (compared to our proof of concept simple XOR instruction), used DPA to recover K from the initialization phase.

Finalization phase of ASCON-Encryption*.* In this phase, K is directly XORed with part of the state, and the result is T, which will be available to the attacker. For non-fixed inputs, T can be considered random. Hence, mixing it with bits of K is catastrophic (from a power analysis security point of view). Listing 1.3 highlights some of the relevant instructions of this phase.

```
...    ; r4 holds address of the state, r5 address of the key
109ae: e9d5 0100    ldrd  r0, r1, [r5]    ; r0 = K0, r1 = K1
109b2: 69a2         ldr   r2, [r4, #24]  ; r2 = S0
109b4: 69e3         ldr   r3, [r4, #28]  ; r3 = S1
109b6: 4050         eors  r0, r2      ;    r0 = K0 ⊕ S0 (T0 = r0)
...    ; r0,r2 are not touched, and r4 has not changed
```

Listing 1.3: Inside the finalization of ASCON-Encryption.

We denote the last 128 bits of the state with S, and we have $\texttt{T} = \texttt{K} \oplus \texttt{S}$ with the corresponding assembly code in Listing 1.3. Words of S are denoted with Si for $0 \leq i \leq 3$. At address 0X109b6, K0 (the first word of K) and S0 are operands of an eors (XOR operation), which with a DPA attack will reveal K0, with the assumption that with non-fixed inputs, Sis are uniform. There are similar eors instructions to recover the rest of Kis.

Data absorption. In our SC model, the attacker can control associated data bits and message bits for its purposes. These bits are XORed with the state bits. Consequently, if the rate r is high, it will recover many state bits. However, we leave the question of whether $r = 64$ or $r = 128$ for $n = 320$ in ASCON can or cannot be used to recover all bits of the state for future research.

State recovery is not revealing *K*. As stated in the specification of ASCON, thanks to XOR of $0^{\dagger} \parallel K$ and $K \parallel 0^{\dagger}$ at the initialization and finalization, even if the attacker recovers the state of any of the permutations in the middle phases of encryption/decryption, it will not be able to recover K or forge T. This property is also beneficial for masking the cipher. An implementation can employ masking only at initialization and finalization to withstand key recovery attacks for encryption and decryption. However, ASCON needs uniform masking for plaintext recovery attacks (if they are to be cared for); see also Section 5.

Experimental validation of the attacks. Figure 4 depicts the results of our experiment for the described attack in the initialization phase of ASCON. Recall that we need to recover bytes of K1, and then with the knowledge of these bytes and using leakage of one instruction before, bytes of K3 are estimated. With 20K traces, the correct values for the bytes have a rank of one/two in our setup. A profiling phase with a known key is required to find the time interval of targeted instructions. As in Figure 4, we are only interested in the peak value of the correlation in the interval of the targeted instruction. Our attack in the finalization phase also proceeds similarly with one exception. The correct values for the key bytes in their targeted instructions correspond to negative peaks. There, we are interested in the leakage of an input operand of the instructions.

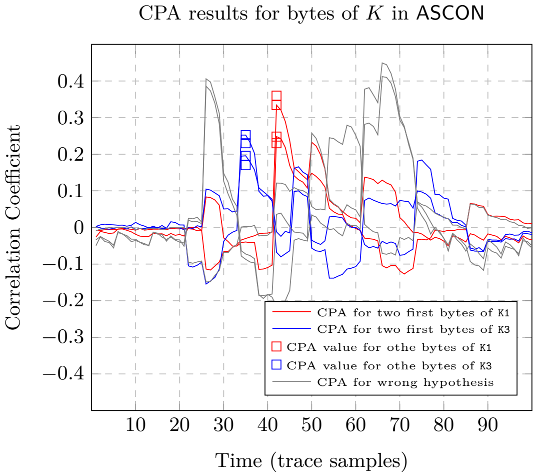


Fig. 4: Attacking the initialization phase of ASCON. Interval for K1 is one instruction after that of K3.

4.2 Power analysis of GIFT-COFB

GIFT-COFB [3] uses the block cipher GIFT in a Combined Output FeedBack (COFB) mode. Neither the mode nor the block cipher claims to have built-in leakage resistance. Thus, it is no surprise that the DPA requirements are met for various instructions. K is only used within the block cipher. Because of this, we skip the mode description and only focus on the block cipher. From the mode, we need to know that for absorbing associated data and for encrypting plaintext, GIFT is called with the same K under some (for the attacker) controllable input or known output data. We will discuss how each call is susceptible to a DPA key recovery attack, implying that every GIFT invocation should be protected for an SC secure implementation. As a result, GIFT-COFB requires uniform masking.

Structure of GIFT. Figure 5 shows that GIFT : $(K, I) \rightarrow O$ has a classical substitution-permutation network (SPN) design, with K and I as the inputs and O as the output, where $|K| = |I| = |O| = 128$. This block cipher is composed of 40 similar rounds. In each round, 64 bits of the state are XORed with a 64-bit round key. Relevant to our SC discussion, the key schedule is simple: the round keys are derived with specific shifts of K, so knowledge of any two consecutive round keys is sufficient to learn K.

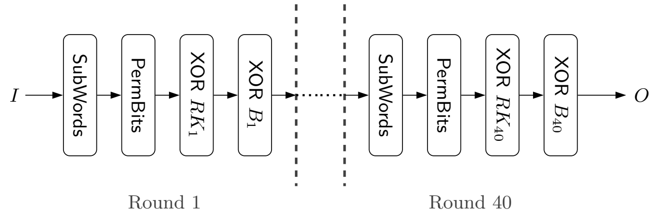


Fig. 5: GIFT block cipher. The RK_is are round key, and the B_is are constant.

DPA possibilities in GIFT-COFB. In the initialization phase, the input to GIFT is used for key recovery, and in the message encryption phase the output of GIFT is used for the attack.

Initialization phase. In this phase, the input I is N. This enables the attacker to exploit leakages of the two initial rounds to recover all bytes of K. In Listing 1.4, we have extracted the relevant XOR instructions of these rounds.

```
...   ;Round1
...   ; t0 = (K12 ≪ 8)|K13, t1 = (K14 ≪ 8)|K15
10cae: 406a        eors   r2, r5; r2 = S1, S1 ⊕= ((t0 ≪ 16)|t1)
...   ; t2 = (K4 ≪ 8)|K5, t3 = (K6 ≪ 8)|K7
10cc6: ea83 0308   eor.w  r3, r3, r8; r3 = S2, S2 ⊕= ((t2 ≪ 16)|t3)
...   ;Round2
...   ; t0 = (K8 ≪ 8)|K9, t1 = (K10 ≪ 8)|K11
10cae: 406a        eors   r2, r5; r2 = S1, S1 ⊕= ((t0 ≪ 16)|t1)
...   ; t2 = (K0 ≪ 8)|K1, t3 = (K2 ≪ 8)|K3
10cc6: ea83 0308   eor.w  r3, r3, r8; r3 = S2, S2 ⊕= ((t2 ≪ 16)|t3)
```

Listing 1.4: Inside the initialization of GIFT-COFB-Encryption.

In Listing 1.4, the XOR instructions (i.e., `eors` and `eor.w`) are 32-bit. `Ki`, for $0 \leq i \leq 15$, denotes bytes of K. `Si`, for $0 \leq i \leq 3$, denotes words of the state of GIFT. Registers `r5` and `r8` contain the round keys. Since N is the input of GIFT, the `Si` values in the first round are known. For the second round, the round key, i.e., $(\texttt{t0} \ll 16)|\texttt{t1}$ and $(\texttt{t2} \ll 16)|\texttt{t3}$, is required. Therefore, a successful DPA in the first round is a prerequisite for attacking the second round.

Plaintext encryption phase. For encryption of each plaintext block, GIFT is called with K and some input I to produce output O. The corresponding ciphertext block is produced as $C = O \oplus M$. Hence, the attacker that knows both 128-bit M and C can compute O. Combined with the leakage information of the last two rounds, in an almost similar way to the initialization phase, the attacker can recover four 32-bit secrets that collectively reveal K. Hou et al. [15] also presented an SC attack targeting K with only knowledge of C without requiring M.

4.3 Power analysis of Grain128-AEAD

Grain128-AEAD uses a stream cipher as its core cryptographic primitive. For this candidate, the specification does not mention resilience to SC attacks. However, due to involvement of many non-linear relations, it seems infeasible to identify sufficient instructions to mount a DPA on key K. Our main observation, from SC point of view, is that the bits of K, in the initialization phase, are processed in separate XORs. However, there is no known randomness associated with these XOR operations. In this cipher, the bits of K only appear in the initialization. Moreover, because of the non-linear and hard-to-invert computations of the core stream cipher, we assume that state recovery after the initialization phase will not reveal anything about K. Based on these two reasons, the DPA discussion will be confined to the initialization phase. We could particularly conclude that only the initialization needs masking.

The initialization of Grain128-AEAD. The cipher comprises one LFSR and one NFSR, both with 128-bit states. In the initialization phase, the LFSR is loaded with a 96-bit N, and the NFSR is loaded with a 128-bit K. During the initialization, in each clock cycle $0 \le t \le 511$, these two states are updated and merged via a non-linear function H that operates on the bits of the two states. More concretely, let $S_t = [s_0^t, s_1^t, \ldots, s_{127}^t]$ be the content of the NFSR after clock t, and $B_t = [b_0^t, b_1^t, \ldots, b_{127}^t]$ be that of the LFSR. The NFSR is preloaded with the key as $b_i^0 = k_i$ for $0 \le i \le 127$, where the k_is are the bits of K. For the LFSR, we have $s_i^0 = n_i$, for $0 \le i \le 95$, $s_i^0 = 1$ for $96 \le i \le 126$, and $s_{127}^0 = 0$, where n_is are bits of N. With a linear feedback function L and a non-linear feedback function F, during $0 \le t \le 319$, the two 128-bit shift registers are updated as

$$\begin{aligned} s_{127}^{t+1} &= \mathsf{L}(S_t) \oplus \mathsf{H}(S_t, B_t), \\ b_{127}^{t+1} &= s_0^t \oplus \mathsf{F}(B_t) \oplus \mathsf{H}(S_t, B_t). \end{aligned} \tag{3}$$

For $320 \le t \le 383$, the states are updated with reintroducing the key bits as

$$\begin{aligned} s_{127}^{t+1} &= \mathsf{L}(S_t) \oplus \mathsf{H}(S_t, B_t) \oplus k_{t-256}, \\ b_{127}^{t+1} &= s_0^t \oplus \mathsf{F}(B_t) \oplus \mathsf{H}(S_t, B_t) \oplus k_{t-320}. \end{aligned} \tag{4}$$

Finally, for $384 \le t \le 511$, state are once again updated, this time with

$$\begin{aligned} s_{127}^{t+1} &= \mathsf{L}(S_t), \\ b_{127}^{t+1} &= s_0^t \oplus \mathsf{F}(B_t). \end{aligned} \tag{5}$$

DPA analysis of the initialization phase. The first place for seeking DPA vulnerable instructions are those governed with relations (3) for $0 \leq t \leq 319$. During these instructions, the attacker can track the mixing of the key bits and randomness bits. However, non-linear relations will come into play such that it seems unlikely that an attacker can locate enough linear combinations to recover all the bits of K. The next place where the bits of K are involved is for $320 \leq t \leq 383$. Listing 1.5 belongs to this segment of the assembly code.

```
10954:  ea88  0606  eor.w  r6,r8,r6  ;r6 = r8 ⊕ r6,r8 = k_{t−256},r6 = L(S_t)
10970:  ea83  0509  eor.w  r5,r3,r9  ;r5 = r3 ⊕ r9,r3 = k_{t−320},r9 = s_0^t
```

Listing 1.5: Inside the initialization, where the key bits are reintroduced.

The XOR instructions in Listing 1.5 are inside a loop called for $320 \leq t \leq 383$. Each bit of K participates in one operation and there is no known randomness. The attacker only knows that `r6` (at the input of XOR) and `r9` are not fixed. Joint leakage of `r6` before and after XOR can leak corresponding K bits. In Listing 1.5, all registers contain only one bit of information. For this cipher, to generate an ARMv7-M compatible assembly, only reference C code was available, which was bit-oriented. For single-bit registers, HW leakages of `r8` and `r3` are enough to recover K completely because, for single-bit variable v, we have $\mathsf{HW}(v) = v$. This discussion is an excellent example of how bit-slicing can improve SC security.

4.4 Power analysis of ISAP

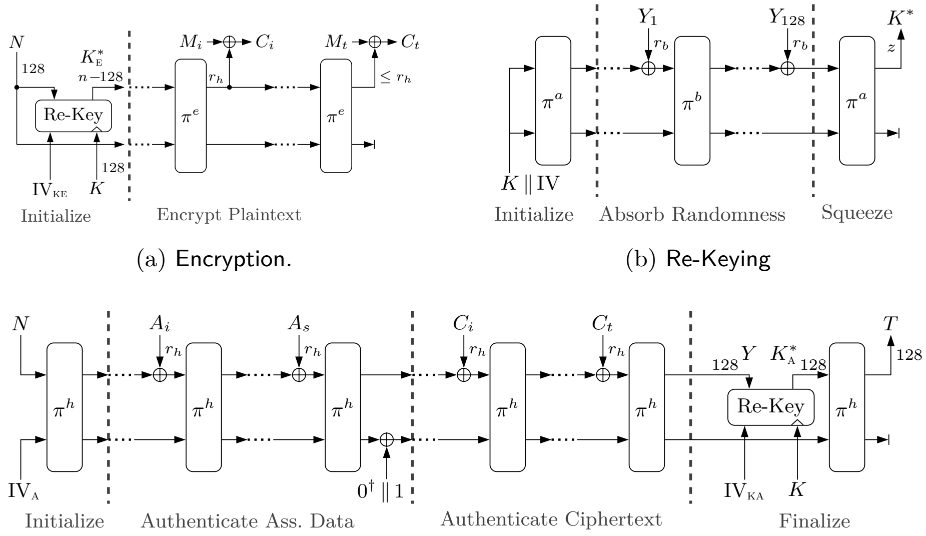
ISAP [10] is designed to offer mode-level protection against various implementation attacks. It is sponge-based and has four instances that differ in the value of some parameters and the internal permutation π, which is either Ascon-p, with state size $n = 320$, or Keccak-p, with $n = 400$.

ISAP-Encryption works as in Figure 6a. It first creates a fresh $(n - 128)$-bit session key by calling the Re-Keying algorithm (see Figure 6b). In this call, the randomness bits Y_is are the 128 bits of N. A separate algorithm ISAP-Authentication, as in Figure 6c, is used for generating a 128-bit tag T. This means that ISAP has an encrypt-then-MAC composition.

The Re-Keying is called twice: once for producing a session key and once during tag generation. The IV, z, and Y_is are different for the two use cases. For a fixed π, $\{\mathrm{IV}_{\mathrm{A}}, \mathrm{IV}_{\mathrm{KE}}, \mathrm{IV}_{\mathrm{KA}}\}$ are constant.

DPA prevention in ISAP. In this mode, DPA attacks are avoided, even in the initialization and finalization phases, without requiring any *leak-free* component. The divide-and-conquer strategy of SC attacks is not feasible, and finding a sufficient number of instructions for a successful DPA attack seems impossible. We look at different parts of the mode to justify DPA prevention.

Inside Re-Keying. In the computation of $\pi^a(K \| IV)$, since no random value is involved, the attacker cannot mount a DPA attack. In the absorbing randomness phase, the Y_is are known and random parameters. The main obstacle



(a) Encryption.

(b) Re-Keying

(c) Authentication

Fig. 6: ISAP working mode.

for the attacker is that K is *well-mixed* before the start of this phase, so it is impossible to find any intermediate that is only based on a fraction of K. The attacker can target Y_i absorb operations. For example, for Y_1, we have a $\mathsf{Trunc}(\pi^a(K \| IV), 1) \oplus Y_1$ operation. However, this combination does not provide enough randomness for a successful DPA attack [29,12]. Since r_b is 1, Y_1 has only two possible values, where $\pi^a(K \| IV)$ is 32-bit or even more. Finally, the squeezing phase is also DPA-secure since no randomness is involved in its computations.

Inside ISAP-Encryption. After the computation of a fresh session key K_E^*, the sponge is squeezed at rate r_h, and the output bits are used to encrypt plaintext blocks. K_E^* is entirely fresh each time. Therefore, the requirement of DPA attacks that the targeted secret remains fixed during multiple executions is not satisfied.

Inside ISAP-Authentication. For a DPA attack, phases before finalization are not helpful since no secret is involved. The finalize itself is composed of Re-Keying with randomness Y and an extra permutation π^h. We have already considered Re-Keying. The permutation is also DPA-secure since it involves no randomness.

Masking considerations. For this candidate, state recovery during encryption/decryption is not leading to key recovery. However, state recovery in each of the permutations inside Re-Keying reveals K. That is because both Ascon-p and Keccak-p are invertible. Hence, Re-Keying should be uniformly protected. We were unable to point to a possible DPA attack. Thus, the discussion about protected implementation is only meaningful considering state recovery SC attacks. Otherwise, no protection is needed.

4.5 Power analysis of Xoodyak

Xoodyak [9] is also a sponge-based AEAD candidate. In this cipher, the authors propose using Taha and Schaumont's method [29] to combat DPA key recovery attacks at the initialization phase, similar to ISAP in Section 4.4. Meanwhile, the source code submitted to the NIST LWC competition does not include this SC protection. We skip the power analysis of the initialization phase and focus on the relatively high absorption rate of associated data.

Construction of Xoodyak. Xoodyak-Encryption works as in Figure 7. The internal permutation π is Xoodoo-p with state size $n = 384$ bits and a constant number of rounds. The permutation is invertible, even though only forward computation is required for the regular operation of the cipher. The tag, nonce, and key are all 128 bits long. The computations and the assembly code are byte-oriented. Rate r_a, by which the associated data is absorbed, satisfies $r_a = (44 \times 8) = n - 32$. Message blocks are $r_b = (24 \times 8) = n/2$ bits long, except for the last block, i.e., M_t, which can have less than r_b bits. The constant byte 0x01 is used to pad the A_is and M_is.

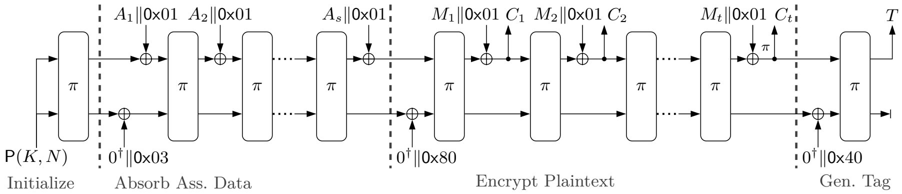


Fig. 7: Xoodyak-Encryption, with P$(K, N) = K\|N\|$0x80$\|$0x01$\|0^\dagger\|$0x02.

DPA possibilities in Xoodyak. In the associated data absorption phase (see Figure 7), r_a bits of the state are mixed with (for the attacker) controllable associated data. State recovery with DPA on the corresponding XOR instructions is possible, provided it is allowed to ask for encryption (or decryption) of messages under the same nonce. By doing so, for absorbing a ($r_a = 8 \times 44$)-bit A_i, there are 44 XOR instructions in the assembly code (one for each byte of A_i) that can be targeted with DPA for recovering the corresponding bytes of the state. Successful application of these attacks will provide 44 out of 48 bytes of the state. The remaining 4 bytes can be found by exhaustive search to learn the state entirely. Recovery of the state directly uncovers K, since Xoodoo-p is easy to invert.

4.6 Power analysis of Elephant

The NIST LWC candidate Elephant [7] has an encrypt-then-MAC construction and includes three instances. The one with the smallest state size, i.e., $n = 160$, is the main recommendation of its designers for the competition. The instances differ in the choice of internal building permutation π, and consequently, in their state size. They also differ in the length of the tag. Elephant has no claim for SC security of its unprotected implementation, and unsurprisingly there are successful key recovery attacks with power analysis for it [32]. For this cipher, we describe the Elephant-Encryption algorithm. Inside the encryption, we point to XOR operations that are enabling a key recovery DPA.

Specification of Elephant. The encryption works as in Algorithm 1. The permutation π has input and output length n. The sequences $R^{a,b}(K)$, for $b \in \{0,1,2\}$ and $a \in \mathbb{N}$, are generated with a byte-oriented LFSR, whose state size n and feedback polynomial depend on the choice of π. The function $\psi : \{0,1\}^n \to \{0,1\}^n$, on each call, produces its output by $n/8$ times clocking the LFSR and concatenating the output bytes.

Algorithm 1 Elephant-Encryption

```
    Input $(K, N, A, M) \in \{0,1\}^{128} \times \{0,1\}^{96} \times \{0,1\}^* \times \{0,1\}^*$
    Output $(C, T) \in \{0,1\}^{|M|} \times \{0,1\}^{|T|}$
$(M_1 \dots M_t) = \mathsf{Split}(M, n)$
for $i = 1$ to $t$ do
    $C_i = M_i \oplus \pi((N \,\|\, 0^{n-96}) \oplus R^{i-1,1}(K)) \oplus R^{i-1,1}(K)$
$C = \mathsf{Trunc}(C_1 \| \dots \| C_t, |M|)$
$(A_1 \dots A_s) = \mathsf{Split}(N \,\|\, A \,\|\, 1, n)$
$(C_1 \dots C_{t'}) = \mathsf{Split}(C \,\|\, 1, n)$
$T' \leftarrow A_1$
for $i = 2$ to $s$ do
    $T' = T' \oplus \pi(A_i \oplus R^{i-1,0}(K)) \oplus R^{i-1,0}(K)$
for $i = 1$ to $t'$ do
    $T' = T' \oplus \pi(C_i \oplus R^{i-1,2}(K)) \oplus R^{i-1,2}(K)$
$T' = \pi(T' \oplus R^{0,0}(K)) \oplus R^{0,0}(K)$
return $(C, \mathsf{Trunc}(T', |T|))$
```

The sequences $R^{a,b}(K)$ are generated as follows, where a number over ψ denotes the number of times it is composed:

$$\begin{aligned}
R^{a,0}(K) &= \psi^{a}(\pi(K \,\|\, 0^{n-128})), \\
R^{a,1}(K) &= \psi^{a+1}(\pi(K \,\|\, 0^{n-128})) \oplus \psi^{a}(\pi(K \,\|\, 0^{n-128})), \\
R^{a,2}(K) &= \psi^{a+2}(\pi(K \,\|\, 0^{n-128})) \oplus \psi^{a}(\pi(K \,\|\, 0^{n-128})).
\end{aligned} \tag{6}$$

DPA on Elephant. Successful application of power analysis exploits two features of Elephant: (a) π is invertible for all the three alternatives and (b) the states of the underlying LFSRs (and so the ψs), as for LFSRs in general, can be reconstructed with access to a certain amount of their output bits. The source code of Elephant is byte-oriented. This means that the intermediates of the corresponding assembly code are 8 bits. Compared to ASCON, the key is not directly used at multiple points. It is only used for creating stream $R^{a,b}(K)$. Therefore, for a key recovery attack, $R^{a,b}(K)$ should be targeted. Internal computations of $R^{a,b}(K)$ involve no randomness. Thus, it is not the best place to mount DPA. However, stream $R^{a,b}(K)$ is XORed with N frequently. These occasions can lead to the recovery of enough bytes of $R^{a,b}(K)$, which quickly results in recovery of K because $R^{a,b}(K)$ for any $b \in \{0, 1, 2\}$ is a linear system of equations with $\pi(K \| 0^{n-128})$ as its unknowns. By finding $\pi(K \| 0^{n-128})$, the attacker can compute K since π is invertible.

Masking concerns of Elephant. The cipher is suitable for parallel computation. This feature is retained in the masked implementation as well. Also, the function ψ has no non-linearity. Thus, it is easy to mask. However, the masking overhead will be high: since the recovery of output bits of each $R^{a,b}(K)$ will lead to recovery of K, all iterations of encryption (for each message block), and all iterations of MAC (for each ciphertext block and associated data block) should be protected.

4.7 Power analysis of TinyJambu

TinyJambu uses a keyed permutation π_K derived from a non-linear stream cipher with ($n = 128$)-bit state. This cipher supports various key sizes. Here, we consider its 128-bit instance. The nonce is 96 bits, and the tag is 64 bits [33]. The cipher has an optimized 32-bit C code. The absorption rate of the nonce is $r = 32$. Our investigation of the corresponding assembly code demonstrates that $r = 32$ is high enough to make the initialization phase vulnerable to a DPA key recovery attack.

The initialization of TinyJambu. This cipher, at its core, utilizes an NFSR with a 128-bit shift register. Let $S = [s_0, s_1, \ldots, s_{127}]$ be the state of this NFSR, and F be the non-linear feedback polynomial of it. Permutation $\pi_K^a : \{0,1\}^{128} \to \{0,1\}^{128}$ is computed by a times clocking the NFSR. If X is the input to π_K^a, X will be the initial state of the NFSR, and the state of the NFSR after a clock will be $\pi_K^a(X)$. In each clock t, the bits in S will be shifted one position to the left, and s_{127} will update as

$$s_{127} = \mathsf{F}(S) \oplus k_{t \bmod n}, \tag{7}$$

where k_is for $0 \le i \le 127$ are the bits of K. The initialization is composed of two steps. First, K is absorbed, then a 96-bit nonce N is merged with the state in blocks of 32 bit. More concretely, the initialization is given in Algorithm 2.

Algorithm 2 TinyJambu-Initialization

```
    Input $(K, N) \in \{0,1\}^{128} \times \{0,1\}^{96}$
    Output 128-bit state $S$ of the NFSR
 1: function $\pi_K^a$(S)
 2:     for $t = 0$ to $a - 1$ do
 3:         $b = \mathsf{F}(S) \oplus k_{t \bmod n}$
 4:         for $j = 0$ to $126$ do
 5:             $s_j = s_{j+1}$
 6:         $s_{127} = b$
 7: return $S$

 8: $S = 0^{128}$
 9: $S = \pi_K^{1024}(S)$                                  ▷ The key setup phase
10: for $i = 0$ to $2$ do                                   ▷ The nonce absorbing phase
11:     $s_{36} = s_{36} \oplus 1$
12:     $S = \pi_K^{640}(S)$
13:     for $j = 0$ to $31$ do
14:         $s_{96+j} = s_{96+j} \oplus N_{32 \cdot i+j}$           ▷ $N_i$s denote bits of the nonce
15: return $S$
```

The pseudocode in Algorithm 2 is bit-oriented. However, the feedback function F and the nonce XORing steps are designed to fit in 32-bit bit-sliced operations nicely.

First order DPA in the Initialization. In Algorithm 2, it is apparent that inside a for loop, bits of N are XORed with bits of the state. In the optimized bit-sliced implementation, the inner for loop (lines 13 and 14 of Algorithm 2) are laid in a single 32-bit XOR. The code at line 0x10636 in Listing 1.6 is exactly this XOR for the first iteration of the outer for loop (at line 10 of Algorithm 2).

```
...   ;First iteration :  i = 0
10636:ea83 0302 eor.w r3, r3, r2   ; r3 = S3, r2 = N0, S3 = S3 ⊕ N0
10632:68c3      ldr   r3, [r0, #12] ; Save the result
...   ;Second iteration :  i = 1
10518:68c4      ldr   r4, [r0, #12] ; Load the result to r4
1051e:046b      lsls  r3, r5, #17   ; r5 = S2, r3 = S2 ≪ 17
10520:ea4f 1a44 mov.w sl, r4, lsl #5      ;sl = S3 ≪ 5
10526:ea43 33d6 orr.w r3, r3, r6, lsr #15;r3 = (S1 ≫ 15)|(S2 ≪ 17)
1052a:ea4a 6ad5 orr.w sl, sl, r5, lsr #27;sl = (S2 ≫ 27)|(S3 ≪ 5)
1053e:ea83 030a eor.w r3, r3, sl   ; r3 = r3 ⊕ sl
10542:407b      eors  r3, r7       ; r7 = K0
```

Listing 1.6: Inside the initialization of TinyJambu-Encryption.

In Listing 1.6, Ni, Si, and Ki for $0 \le i \le 3$ denote words of N, the state, and K, respectively. The first line of the presented Listing is $\mathtt{S3} = \mathtt{S3} \oplus \mathtt{N0}$, which with a random nonce fulfills the requirements of a DPA attack to recover S3.

After successfully recovering S3, the attacker can go for S2 with the leakage of the orr.w instruction at line 0x1052a. In this second DPA attack, S3 is the known randomness, and S2 is the target secret. We consider S3 as random, since in this stage, it is XORed the with first word of the nonce. In the ARMv7-M architecture, orr.w is a bitwise OR operation. With the recovery of S2, the value of the shift register sl at line 0x1052a will be known. The attacker can do almost the same steps to conduct the third DPA attack, this time for recovering r3 with the leakage of the instruction at line 0x1053e. For this attack, sl is the known randomness. However, because of the OR operation, even with a uniform S3, the distribution of sl is not necessarily uniform. As a result, practically, this DPA will have low success. The leakage of the XOR instruction at line 0x10542 with known randomness r3 with the fourth DPA can finally uncover a word of K. The attack, in somewhat similar reasoning, continues to recover all words of K.

Masking considerations for TinyJambu. For absorption of associated data and also for the plaintext encryption phase, the permutation π_K is called immediately after XORing the state with a block of 32-bit (for the attacker) controllable data. These calls have the same structure as the nonce absorption phase. Therefore, they are also vulnerable to the presented DPA attack. Consequently, all of these calls to π_K should be protected for secure implementation. In other words, uniform masking for the entire cipher is required.

5 Plaintext recovery with leakage

SC works primarily target recovery of the key. However, leakage can be exploited for other malicious purposes as well, see [6,5] for tag forging with leakage. In this section, we discuss plaintext recovery attacks in detail. Assume that an attacker wants the plaintext M^* for a received challenge tuple (N^*, A^*, C^*) without knowing a valid tag. It is allowed to ask for the decryption of tuples (N, A, C, T) and receive the leakage. For invalid tags, the decryption responds with a failure symbol. For some candidates, this access enables to run a DPA attack to learn M^*.[6] For example, in ASCON (see Figure 3), an r-bit plaintext block M_i is blinded with part of the current state S as $C_i = M_i \oplus \mathsf{Trunc}(S, r)$. For decryption, the same state S is used to unblind C_i as $M_i = C_i \oplus \mathsf{Trunc}(S, r)$. Starting from the first word of C^* in a chosen nonce setup, the attacker can ask for the decryption of a random C_1 block with the same (N^*, A^*) and learn $\mathsf{Trunc}(S^*, r)$ by DPA over the decryption XOR instructions. Knowledge of $\mathsf{Trunc}(S^*, r)$ is sufficient for unblinding C_1^*. Similar DPA attacks can successively recover all parts of C^*.

However, in an encrypt-then-MAC mode, since there is a separate authentication algorithm, it is impossible to get decryption leakage without knowing a valid tag.

[6] This game is formally defined in [14]. Here, we are interested in the applicability of our DPA with the XOR model.

6 Summary of the results

Table 1 summarizes the results of this paper for the considered AEAD ciphers of the NIST LWC competition. In this table, column (A) is checked if, for the candidate, there is a DPA key recovery attack in the random nonce setting. Column (B) is checked if our DPA key recovery attack requires chosen nonce. For some candidates (specified in column (C)), the requirement of known randomness for a DPA is not satisfied; however, the key is being processed in different XOR operations. As discussed in Section 4.3, these operations can leak the key for the case of $w = 1$. This means that the assembly codes' word size (in column (D)) is relevant for power analysis, see also the discussion in Section 3. For protected implementations, we discussed in the introduction that non-uniform masking is an advantage. With caring only for key recovery attacks, the list of candidates requiring only non-uniform masking is in (E). The ciphers that prevent decryption leakage are highlighted in (F), see the discussion in Section 5. Finally, in (G), for the applicable candidates, we specify that their SC-aware version is not heavier than the primary instance by putting a check mark. A blank space in column (G) means this candidate has no SC-aware instance. In other columns, a blank space means that there is already a more practical attack.

Cipher	(A)	(B)	(C)	(D)	(E)	(F)	(G)
ASCON	✓			32 bit	✓	×	
Elephant	✓			8 bit	×	✓	
GIFT-COFB	✓			32 bit	×	×	
Grain128-AEAD	×	×	✓	1 bit	✓	×	
ISAP	×	×	×	32 bit	✓	✓	✓
TinyJambu	✓			32 bit	×	×	
Xoodyak	×	✓		8 bit	×	×	×

Table 1: Summary of our power analysis results for the studied AEAD ciphers.

7 Conclusion

This work, for the first time, systematically investigates the impact and potential exploitation of DPA attacks on linear operations in lightweight authenticated encryption schemes. We presented a theoretical discussion for each of the seven ciphers investigated in this work, all of which are finalists in the NIST LWC competition. In addition, for ASCON, we presented experimental evidence that the introduced framework applies. However, we stress that these discussions are only for the unprotected versions of the schemes. As future work, it is interesting to investigate the robustness of the protected versions of the ciphers against our framework.

References

1. Azouaoui, M., Bellizia, D., Buhan, I., Debande, N., Duval, S., Giraud, C., Jaulmes, È., Koeune, F., Oswald, E., Standaert, F.X., Whitnall, C.: A Systematic Appraisal of Side Channel Evaluation Strategies. In: van der Merwe, T., Mitchell, C., Mehrnezhad, M. (eds.) Security Standardisation Research. pp. 46–66. Springer International Publishing, Cham (2020)
2. Balasch, J., Gierlichs, B., Grosso, V., Reparaz, O., Standaert, F.X.: On the Cost of Lazy Engineering for Masked Software Implementations. In: Joye, M., Moradi, A. (eds.) Smart Card Research and Advanced Applications. pp. 64–81. Springer International Publishing, Cham (2015)
3. Banik, S., Chakraborti, A., Iwata, T., Minematsu, K., Nandi, M., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: GIFT-COFB (v1.1). The NIST Lightweight Cryptography (LWC) Standardization project (A Round-3 Candidate), 2021. `https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/gift-cofb-spec-final.pdf` (2021)
4. Battistello, A., Coron, J.S., Prouff, E., Zeitoun, R.: Horizontal Side-Channel Attacks and Countermeasures on the ISW Masking Scheme. In: Gierlichs, B., Poschmann, A.Y. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2016. pp. 23–39. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
5. Bellizia, D., Bronchain, O., Cassiers, G., Grosso, V., Guo, C., Momin, C., Pereira, O., Peters, T., Standaert, F.X.: Mode-Level vs. Implementation-Level Physical Security in Symmetric Cryptography. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020. pp. 369–400. Springer International Publishing, Cham (2020)
6. Berti, F., Pereira, O., Peters, T., Standaert, F.X.: On Leakage-Resilient Authenticated Encryption with Decryption Leakages. IACR Transactions on Symmetric Cryptology p. 271–293 (2017). https://doi.org/10.13154/tosc.v2017.i3.271-293, `https://tosc.iacr.org/index.php/ToSC/article/view/774`
7. Beyne, T., Chen, Y.L., Dobraunig, C., Mennink, B.: Elephant (v2). The NIST Lightweight Cryptography (LWC) Standardization project (A Round-3 Candidate), 2021. `https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/elephant-spec-final.pdf` (2021)
8. Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In: Joye, M., Quisquater, J.J. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2004. pp. 16–29. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
9. Daemen, J., Hoffert, S., Mella, S., Peeters, M., Van Assche, G., Van Keer, R.: Xoodyak, a lightweight cryptographic scheme. The NIST Lightweight Cryptography (LWC) Standardization project (A Round-3 Candidate), 2021. `https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/xoodyak-spec-final.pdf` (2021)
10. Dobraunig, C., Eichlseder, M., Mangard, S., Mendel, F., Mennink, B., Primas, R., Unterluggauer, T.: ISAP (v2.0). The NIST Lightweight Cryptography (LWC) Standardization project (A Round-3 Candidate), 2021. `https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/isap-spec-final.pdf` (2021)

11. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon (v1.2). The NIST Lightweight Cryptography (LWC) Standardization project, 2021. https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/ascon-spec-final.pdf (2021)
12. Dobraunig, C., Mennink, B.: Leakage Resilient Value Comparison with Application to Message Authentication. In: Canteaut, A., Standaert, F.X. (eds.) Advances in Cryptology – EUROCRYPT 2021. pp. 377–407. Springer International Publishing, Cham (2021)
13. Fu, S., Wang, Z., Wei, F., Xu, G., Wang, A.: Linear Regression Side Channel Attack Applied on Constant XOR. Cryptology ePrint Archive, Paper 2017/1217 (2017), https://eprint.iacr.org/2017/1217, https://eprint.iacr.org/2017/1217
14. Guo, C., Pereira, O., Peters, T., Standaert, F.X.: Authenticated Encryption with Nonce Misuse and Physical Leakage: Definitions, Separation Results and First Construction. In: Schwabe, P., Thériault, N. (eds.) Progress in Cryptology – LATINCRYPT 2019. pp. 150–172. Springer International Publishing, Cham (2019)
15. Hou, X., Breier, J., Bhasin, S.: DNFA: Differential No-Fault Analysis of Bit Permutation Based Ciphers Assisted by Side-Channel. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 182–187 (2021). https://doi.org/10.23919/DATE51398.2021.9474154
16. Ishai, Y., Sahai, A., Wagner, D.: Private Circuits: Securing Hardware against Probing Attacks. In: Boneh, D. (ed.) Advances in Cryptology - CRYPTO 2003. pp. 463–481. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
17. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) Advances in Cryptology — CRYPTO' 99. pp. 388–397. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
18. Kumar, S., Dasu, V.A., Baksi, A., Sarkar, S., Jap, D., Breier, J., Bhasin, S.: Side Channel Attack On Stream Ciphers: A Three-Step Approach To State/Key Recovery. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 166–191 (2022)
19. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security). Springer-Verlag, Berlin, Heidelberg (2007)
20. Marshall, B., Page, D., Webb, J.: MIRACLE: MIcRo-ArChitectural Leakage Evaluation: A study of micro-architectural power leakage across many devices. IACR Transactions on Cryptographic Hardware and Embedded Systems p. 175–220 (Nov 2021). https://doi.org/10.46586/tches.v2022.i1.175-220, https://tches.iacr.org/index.php/TCHES/article/view/9294
21. NIST: Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process, 2019. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf (2019)
22. Pereira, O., Standaert, F.X., Vivek, S.: Leakage-Resilient Authentication and Encryption from Symmetric Cryptographic Primitives. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. p. 96–108. CCS '15, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2810103.2813626, https://doi.org/10.1145/2810103.2813626
23. Poussier, R., Standaert, F.X., Grosso, V.: Simple Key Enumeration (and Rank Estimation) Using Histograms: An Integrated Approach. In: Gierlichs, B., Poschmann, A.Y. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2016. pp. 61–81. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)

24. Ramezanpour, K., Abdulgadir, A., Diehl, W., Kaps, J.P., Ampadu, P.: Active and Passive Side-Channel Key Recovery Attacks on Ascon. In: Proc. NIST Lightweight Cryptogr. Workshop. pp. 1–27 (2020)
25. Rivain, M., Prouff, E.: Provably Secure Higher-Order Masking of AES. In: Mangard, S., Standaert, F.X. (eds.) Cryptographic Hardware and Embedded Systems, CHES 2010. pp. 413–427. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
26. Samwel, N., Daemen, J.: DPA on Hardware Implementations of Ascon and Keyak. In: Proceedings of the Computing Frontiers Conference. p. 415–424. CF'17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3075564.3079067, https://doi.org/10.1145/3075564.3079067
27. Schindler, W., Lemke, K., Paar, C.: A Stochastic Model for Differential Side Channel Cryptanalysis. In: Rao, J.R., Sunar, B. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2005. pp. 30–46. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
28. Shelton, M.A., Samwel, N., Batina, L., Regazzoni, F., Wagner, M., Yarom, Y.: ROSITA: Towards Automatic Elimination of Power-Analysis Leakage in Ciphers. NDSS (2021)
29. Taha, M., Schaumont, P.: Side-channel countermeasure for SHA-3 at almost-zero area overhead. In: 2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST). pp. 93–96 (2014). https://doi.org/10.1109/HST.2014.6855576
30. Verhamme, C., Cassiers, G., Standaert, F.X.: Analyzing the Leakage Resistance of the NIST's Lightweight Crypto Competition's Finalists. In: CARDIS. Lecture Notes in Computer Science, Springer (2022), https://perso.uclouvain.be/fstandae/PUBLIS/279b.pdf
31. Veyrat-Charvillon, N., Gérard, B., Standaert, F.X.: Soft Analytical Side-Channel Attacks. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology – ASIACRYPT 2014. pp. 282–296. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
32. Vialar, L.: Fast Side-Channel Key-Recovery Attack against Elephant Dumbo. Cryptology ePrint Archive (2022)
33. Wu, H., Huang, T.: TinyJAMBU: A Family of Lightweight Authenticated Encryption Algorithms. The NIST Lightweight Cryptography (LWC) Standardization project (A Round-3 Candidate), 2019. https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/TinyJAMBU-spec-round2.pdf (2019)
34. Yiu, J.: The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors. Newnes (2013)
35. Zhang, J., Zheng, M., Nan, J., Hu, H., Yu, N.: A Novel Evaluation Metric for Deep Learning-Based Side Channel Analysis and Its Extended Application to Imbalanced Data. IACR Transactions on Cryptographic Hardware and Embedded Systems p. 73–96 (Jun 2020). https://doi.org/10.13154/tches.v2020.i3.73-96, https://tches.iacr.org/index.php/TCHES/article/view/8583
36. Zhang, J., Li, L., Li, Q., Zhao, J., Liang, X.: Power Analysis Attack on a Lightweight Block Cipher GIFT. In: Liu, Q., Liu, X., Li, L., Zhou, H., Zhao, H.H. (eds.) Proceedings of the 9th International Conference on Computer Engineering and Networks. pp. 565–574. Springer Singapore, Singapore (2021)