Correlation Power Analysis on Ascon with Multi-bit Selection Function

Viet Sang Nguyen[a] , Vincent Grosso[b] , Pierre-Louis Cayrel[c]
Université Jean Monnet Saint-Etienne, CNRS, Institut d Optique Graduate School, Laboratoire Hubert Curien UMR 5516, F-42023 Saint-Etienne, France
{*viet.sang.nguyen, vincent.grosso, pierre.louis.cayrel*}*@univ-st-etienne.fr*

Keywords: Correlation Power Analysis, Selection Function, Ascon.

Abstract: Ascon has recently been selected by NIST as the new standard for lightweight cryptography. This highlights the need to evaluate its resilience against implementation attacks such as Correlation Power Analysis (CPA). Traditional CPA on Ascon uses a *1-bit selection function*, modeling power consumption based on a single bit of an machine word. However, actual power leakage depends on the entire word. Therefore, the hypothesized power consumption aligns better with the measured values when more bits of the word are involved in the selection function. This paper investigates the use of *multi-bit selection functions* in CPA on Ascon. We show that the bitsliced-oriented design of Ascon leads the multi-bit selection functions to produce a group of key candidates with high correlations, rather than a single candidate as typically expected in CPA. Through theoretical analysis and experimental validation, we examine this behavior in detail. Based on these insights, we propose an efficient key recovery algorithm tailored for the multi-bit selection functions. Our results demonstrate that this approach significantly reduces the number of CPA runs required for full key recovery.

1 INTRODUCTION

Nowadays, small computing devices like RFID tags, sensors, and smart cards are becoming increasingly widespread. Although the Advanced Encryption Standard (AES, 2001) is a highly reliable and secure cipher, it is often too resource-intensive for deployment in such constrained environments. In this context, NIST initiated a competition to look for a new *lightweight* cryptography standard. In February 2023, NIST announced that Ascon was selected to be standardized. Before that, Ascon had also been included in the final portfolio of the CAESAR competition. The rigorous evaluations conducted during both selection processes have solidified confidence in Ascon's security under the traditional *black-box model*, where the adversary can only access the inputs and outputs.

However, such a model does not always capture the security for physical implementations. When deployed in embedded devices, cryptographic algorithms can be vulnerable to *power analysis attacks*, a potent category of side-channel attacks that exploit the *power consumption* of the devices during algorithm execution. Since the introduction of Differential Power Attack (DPA) (Kocher et al., 1999), power analysis attacks have emerged as a prominent research area. DPA distinguishes the correct key candidate from the incorrect ones by the difference of means. Correlation Power Analysis (CPA) (Brier et al., 2004), a variant of DPA, leverages a specific power consumption model, such as Hamming weight, and uses Pearson correlation to identify the correct key candidate. CPA is a versatile and powerful attack, as it requires only the observation of power consumption leakages during the execution of the cryptographic algorithm. Detailed knowledge of the device is not needed. Knowing the algorithm that is executed by the device is usually sufficient. The goal of CPA is to recover the key through a statistical analysis of key-dependent power leakages.

So far, CPA attacks for Ascon have received relatively little attention. The first successful CPA attack was presented in (Samwel and Daemen, 2017) targeting a noisy hardware implementation. A notable contribution of (Samwel and Daemen, 2017) is the development of an effective *selection function* for computing the *intermediate variable* targeted by the attack. Unlike the popular choice of the S-box output in CPA attacks on AES, selecting a suitable function for Ascon is more complex due to the bitsliced-oriented design. In fact, directly using the S-box

[a] https://orcid.org/0009-0004-2939-8478
[b] https://orcid.org/0000-0002-3874-7527
[c] https://orcid.org/0000-0002-6708-868X

output as the selection function in Ascon can result in a failed CPA attack, as evidenced in (Ramezanpour et al., 2020). Samwel and Daemen constructed their selection function through a careful analysis of how information leaks during computation. Using the same selection function, the CPA attacks in (Roussel et al., 2023; Weissbart and Picek, 2023) also successfully recovered the key. These attacks targeted a hybrid CMOS/MRAM hardware implementation and an ARMv7m software implementation, respectively.

In the design of Ascon, the state is represented by 64-bit variables. Depending on the device architecture, a 64-bit variable can be implemented using multiple smaller machine words, such as 8-bit or 32-bit words. While effective, the selection function proposed by Samwel and Daemen is limited to operate on a single bit (referred to as the *1-bit selection function* hereafter). In other words, the power consumption hypotheses are modeled based solely on the value of a single bit within a machine word. However, in practice, the leaked information depends on the value of the entire machine word (Brier et al., 2004; Tunstall et al., 2007). As a result, the 1-bit selection function may not fully exploit all available leakage in the power consumption traces.

Contributions. In this paper, we explore the feasibility of employing *multi-bit selection functions* to CPA attacks on Ascon. We begin by extending the 1-bit selection function proposed by Samwel and Daemen to operate on multiple bits. Our findings reveal that, due to the bitsliced-oriented design of the Sbox implementation, using multi-bit selections function results in a small group of key candidates with high correlations, rather than a single candidate as typically expected in CPA. Through both theoretical analysis and experimental validation, we provide a comprehensive examination of the underlying causes of this behavior.

Second, based on the insights from our analysis, we propose an efficient algorithm to identify the correct key candidate when employing multi-bit selection functions. We show that the multi-bit approach significantly reduces the number of CPA runs required for full key recovery. Specifically, the number of CPA runs is reduced to 26 and 19 for 2-bit and 3-bit selection functions, respectively, compared to 48 for 1-bit selection function. Additionally, this also facilitates the application of second-order success in CPA, *i.e.*, choosing the top two candidates with the highest correlations.

For the sake of reproducibility, we publish the source code of the experiments as well as the traces at: `https://github.com/nvietsang/multibitcpa-ascon`.

Outline. This paper is organized as follows. Section 2 provides the background knowledge. Section 3 presents the extension from 1-bit to multi-bit selection function. Section 4 provides an efficient key recovery algorithm for CPA using a multi-bit selection. Section 5 discusses countermeasures and future work, and concludes our work.

2 PRELIMINARIES

In this section, we begin by briefly outlining the principle of the Correlation Power Analysis (CPA) attack. We then provide the overview of the Ascon cipher. Next, we recall the 1-bit selection function proposed by Samwel and Daemen, which is the foundation for our extension to a multi-bit selection function. Lastly, we describe the experimental setup and the device used in our experiments.

2.1 CPA attack

In a CPA attack, the attacker analyzes the dependence between the power consumption at specific points in time and the data being processed. The attack involves the following five steps:

- **Choose an intermediate variable as the attack point**. This variable should be a function $f(d,k)$, referred to as the *selection function*, which depends on part of the key k and the known non-constant data d (*e.g.*, plaintext).
- **Measure the power consumption**. Let the device execute the algorithm ℓ times with different inputs. For each execution, the attacker records the data value d involved in the selection function and a power trace of s samples. This process results in a vector of data values $\mathbf{d} = (d_1, \ldots, d_\ell)$, and a power trace matrix $\mathbf{T}$ of size $\ell \times s$.
- **Calculate hypothetical intermediate values**. Let $\mathbf{k} = (k_1, \ldots, k_p)$ represent the p possible key candidates. Using the selection function $f(d,k)$, the attacker calculates the hypothetical intermediate values for each combination of $\mathbf{d}$ and $\mathbf{k}$. This produces a matrix $\mathbf{V}$ of size $\ell \times p$.
- **Derive hypothetical power consumption values**. The attacker uses a leakage model to map each value in $\mathbf{V}$ to a hypothetical power consumption value. In this work, we choose the Hamming weight model. This step produces a hypothetical power consumption matrix $\mathbf{H}$ of size $\ell \times p$.

- **Compare hypothetical and measured power values**. The attacker uses the Pearson's correlation coefficient to compare the hypothetical power consumption values with the measured power traces. Specifically, he calculates the correlation between each column $\mathbf{h}_i$ of $\mathbf{H}$ and each column $\mathbf{t}_j$ of $\mathbf{T}$. The correlation is expressed as:

$$r_{i,j} = \frac{\sum_{u=1}^{\ell} \left(h_{u,i} - \overline{h}_i\right)\left(t_{u,j} - \overline{t}_j\right)}{\sqrt{\sum_{u=1}^{\ell} \left(h_{u,i} - \overline{h}_i\right)^2} \sqrt{\sum_{u=1}^{\ell} \left(t_{u,j} - \overline{t}_j\right)^2}}.$$

  Here, $h_{u,i}$ and $t_{u,j}$ (and their respective means $\overline{h}_i$ and $\overline{t}_j$) denote the u-th elements of the columns $\mathbf{h}_i$ and $\mathbf{t}_j$. The resulting matrix $\mathbf{R}$, of size $p \times s$, contains the correlation coefficient $r_{i,j}$ for each key candidate and trace sample.

The key can be recovered based on the fact that the higher value of $r_{i,j}$ indicates the better match between the columns $\mathbf{h}_i$ and $\mathbf{t}_j$. Let $\mathtt{ck}$ denote the index of the correct key $k_{\mathtt{ck}}$ in the vector $\mathbf{k}$, and $\mathtt{ct}$ denote the index of the power consumption values $\mathbf{t}_{\mathtt{ct}}$, which depend on the intermediate values $\mathbf{v}_{\mathtt{ck}}$. The columns $\mathbf{h}_{\mathtt{ck}}$ and $\mathbf{t}_{\mathtt{ct}}$ should have a strong correlation. Consequently, the highest value $r_{\mathtt{ck},\mathtt{ct}}$ in the matrix $\mathbf{R}$ reveals the indexes of the correct key $\mathtt{ck}$ and the corresponding location $\mathtt{ct}$.

2.2 Ascon

Ascon (Dobraunig et al., 2021) is a suite of Authenticated Encryption with Associated Data (AEAD) and hashing algorithms based on the duplex sponge construction (Bertoni et al., 2012b). This paper focuses on the recommended authenticated cipher variant, Ascon-128 (referred to simply as Ascon hereafter). Figure 1 illustrates the encryption process. Its inputs include a key K of 128 bits, a nonce N of 128 bits, an initialization vector IV, associated data $A_1, \ldots, A_s$, each of 64 bits, and plaintexts $P_1, \ldots, P_t$, each of 64 bits. It produces as output a tag T of 128 bits and ciphertexts $C_1, \ldots, C_t$, each of 64 bits. The tag T is used during the decryption process to verify the authenticity of the ciphertexts.

The permutations, denoted by p^a and p^b, form the core of the Ascon construction. These permutations consist of $a = 12$ rounds and $b = 6$ rounds, respectively. Each round is composed of three steps operating on a 320-bit state: (1) addition of constants, (2) substitution layer (S-box), and (3) linear diffusion layer. The three steps are depicted in Figure 2. The 320-bit state is divided into five 64-bit variables, which can be stored in one or more smaller-sized words (or registers in hardware). This design facilitates the transition from the mathematical description to practical and efficient implementations.[1]

Let $x_0, \ldots, x_4$ represent the five 64-bit variables of the round input. In the first step, a round constant is added to the rightmost eight bits of x_2. Since the constant addition step is not relevant to our attack, we simplify the notation by continuing to denote the output of the first step as $x_0, \ldots, x_4$. The second step involves a non-linear transformation applied to on five bits, with one bit taken from each variable of the first step's output $x_0, \ldots, x_4$. Let $y_0, \ldots, y_4$ represent the output state of the S-box, and let $\mathbf{1}$ (in bold) denote a variable filled with 64 bit 1s. The algebraic normal form (ANF) of the S-box, with all operations performed on the full 64-bit variables (in bitsliced form) can be expressed as:

$$\begin{aligned}
y_0 &= x_4x_1 \oplus x_3 \oplus x_2x_1 \oplus x_2 \oplus x_1x_0 \oplus x_1 \oplus x_0,\\
y_1 &= x_4 \oplus x_3x_2 \oplus x_3x_1 \oplus x_3 \oplus x_2x_1 \oplus x_2 \oplus x_1 \oplus x_0,\\
y_2 &= x_4x_3 \oplus x_4 \oplus x_2 \oplus x_1 \oplus \mathbf{1},\\
y_3 &= x_4x_0 \oplus x_4 \oplus x_3x_0 \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0,\\
y_4 &= x_4x_1 \oplus x_4 \oplus x_3 \oplus x_1x_0 \oplus x_1.
\end{aligned} \tag{1}$$

At the beginning of the initialization phase (Figure 1), the 64-bit initialization vector $\mathtt{IV}$ is stored in x_0. The two 64-bit halves of the key, $(k_0, k_1) = K$, are stored in x_1 and x_2. The two 64-bit halves of the nonce, $(n_0, n_1) = N$, are stored in x_3 and x_4. The S-box computation during the first round of the initialization phase, where our attack focuses on, can thus be written as follows (with the constant addition step omitted for simplicity):

$$\begin{aligned}
y_0 &= n_1k_0 \oplus n_0 \oplus k_1k_0 \oplus k_1 \oplus k_0\mathtt{IV} \oplus k_0 \oplus \mathtt{IV},\\
y_1 &= n_1 \oplus n_0k_1 \oplus n_0k_0 \oplus n_0 \oplus k_1k_0 \oplus k_1 \oplus k_0 \oplus \mathtt{IV},\\
y_2 &= n_1n_0 \oplus n_1 \oplus k_1 \oplus k_0 \oplus \mathbf{1},\\
y_3 &= n_1\mathtt{IV} \oplus n_1 \oplus n_0\mathtt{IV} \oplus n_0 \oplus k_1 \oplus k_0 \oplus \mathtt{IV},\\
y_4 &= n_1k_0 \oplus n_1 \oplus n_0 \oplus k_0\mathtt{IV} \oplus k_0.
\end{aligned} \tag{2}$$

The third step, linear diffusion layer, applies a rotation to each variable at the S-box output twice. The rotated variables are then XOR-ed with the original one. Let $z_0, \ldots, z_4$ denote the output of the linear diffusion layer. The linear functions applied to each variable are defined as:

$$\begin{aligned}
z_0 &= y_0 \oplus (y_0 \ggg 19) \oplus (y_0 \ggg 28),\\
z_1 &= y_1 \oplus (y_1 \ggg 61) \oplus (y_1 \ggg 39),\\
z_2 &= y_2 \oplus (y_2 \ggg 1) \oplus (y_2 \ggg 6),\\
z_3 &= y_3 \oplus (y_3 \ggg 10) \oplus (y_3 \ggg 17),\\
z_4 &= y_4 \oplus (y_4 \ggg 7) \oplus (y_4 \ggg 41).
\end{aligned} \tag{3}$$

[1] Implementations for 8-bit, 32-bit, 64-bit architectures can be found at https://github.com/ascon/ascon-c

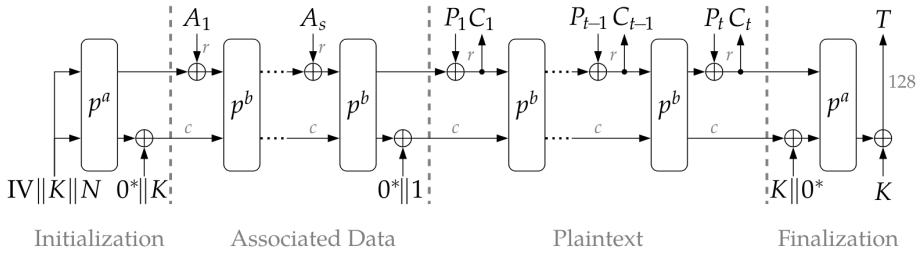
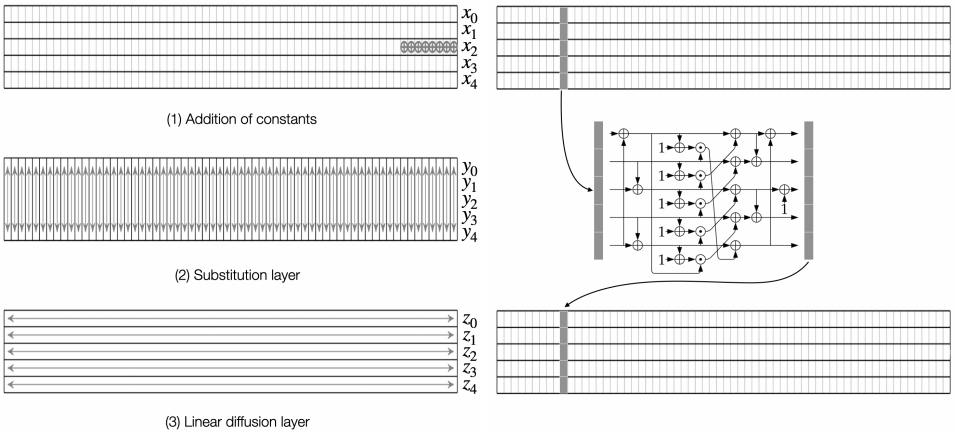


Figure 1: Encryption in Ascon (Dobraunig et al., 2021).



(a) Three steps of a round (b) An S-box computation.

Figure 2: Each step in a round (Dobraunig et al., 2021).

2.3 Selection function

This work relies on the selection function in the attack of (Samwel and Daemen, 2017). The output of the linear diffusion layer is chosen as the attack point. The selection function is derived by analyzing how information leaks through the S-box computation. As in (Samwel and Daemen, 2017), we only focus on y_0, y_1 and y_4 in Equation 2 as their computations contain non-linear terms between the key and the nonce.

We consider y_4 as an example. Let the superscript j denote the index of the j-th bit of a 64-bit variable, where $0 \leq j \leq 63$. The j-th bit of y_4 is computed as:

$$y_4^j = n_1^j(k_0^j \oplus 1) \oplus n_0^j \oplus k_0^j \mathtt{IV}^j \oplus k_0^j.$$

Following Bertoni *et al.* (Bertoni et al., 2012a), the term $k_0^j \mathtt{IV}^j \oplus k_0^j$ can be removed because, for the fixed correct key in the device, this term is independent of the nonce and contributes a constant amount to the activity that drives the targeted power consumption of the register containing y_4. This removal results in $\tilde{y}_4^j$, where:

$$\tilde{y}_4^j = n_1^j(k_0^j \oplus 1) \oplus n_0^j. \tag{4}$$

We next account the linear diffusion operation. Recall from Equation 1 that the 64-bit output variable z_4 of this layer is computed as:

$$z_4 = y_4 \oplus (y_4 \ggg 7) \oplus (y_4 \ggg 41).$$

The computation of the j-th bit of z_4 ($0 \leq j \leq 63$) thus is:

$$z_4^j = y_4^j \oplus y_4^{j+57} \oplus y_4^{j+23}. \tag{5}$$

The additions $j+57$ and $j+23$ are implicitly taken modulo 64. Applying Equation 4 to Equation 5 results in the selection function $\tilde{z}_4^j$, which is used to recover k_0 (three bits at a time):

$$\begin{aligned}\tilde{z}_4^j = &\left(n_1^j(k_0^j \oplus 1) \oplus n_0^j\right) \\ &\oplus \left(n_1^{j+57}(k_0^{j+57} \oplus 1) \oplus n_0^{j+57}\right) \\ &\oplus \left(n_1^{j+23}(k_0^{j+23} \oplus 1) \oplus n_0^{j+23}\right).\end{aligned} \tag{6}$$

Similarly, we can derive the selection functions $\tilde{z}_0^j$ for recovering k_0, and $\tilde{z}_1^j$ for recovering k_1. The detailed derivation steps are provided in Appendix A. Here, we present the final result of $\tilde{z}_1^j$:

$$\begin{aligned}\tilde{z}_1^j = &\left(n_0^j(k_{01}^j \oplus 1) \oplus n_1^j\right) \\ &\oplus \left(n_0^{j+3}(k_{01}^{j+3} \oplus 1) \oplus n_1^{j+3}\right) \\ &\oplus \left(n_0^{j+25}(k_{01}^{j+25} \oplus 1) \oplus n_1^{j+25}\right),\end{aligned} \tag{7}$$

where $k_{01}^j = k_0^j \oplus k_1^j$. Note that k_1^j is not directly recovered, instead, k_{01}^j is recovered when $\tilde{z}_1^j$ is used as the selection. Then, k_1^j is derived as $k_1^j = k_{01}^j \oplus k_0^j$, with k_0^j recovered from the CPA using $\tilde{z}_4^j$ as the selection function. In this following sections, we will use Equation 6 and Equation 7 as the selection functions to recover the two key halves k_0 and k_1.

2.4 Experiment setup

We use a ChipWhisperer Lite board, integrated with an STM32F303 32-bit ARM target microcontroller, to record the power consumption traces. The device operates with a default clock frequency of 7.37 MHz. The ChipWhisperer board is connected to a MacBook Air M1 with 16 GB of RAM via a USB cable. To create a scenario as realistic as possible, we record the power traces during the executions of the 32-bit optimized ARMv6 implementation by the Ascon team,[2] which is well-suited to our microcontroller.

3 MULTI-BIT SELECTION FUNCTION

Using Equation 6 and Equation 7 as the selection functions, as in some successful CPA attacks (Samwel and Daemen, 2017; Weissbart and Picek, 2023; Roussel et al., 2023), means to exploit the leakages of single bits z_4^j and z_1^j (in 64-bit variables z_4 and z_1). These leakages are modeled as the Hamming weight of $\tilde{z}_4^j$ and $\tilde{z}_1^j$, denoted by $\mathrm{HW}(\tilde{z}_4^j) = \tilde{z}_4^j$ and $\mathrm{HW}(\tilde{z}_1^j) = \tilde{z}_1^j$, since the Hamming weight of a bit is the bit value itself. In software implementations, the 64-bit variables z_4 and z_1 are usually implemented by eight 8-bit or two 32-bit words, depending on device architecture. The activity of these words (load/store) is known to leak information about their contained data through power consumption. The ideal attack scenario is to consider the entire word length to make hypotheses. In this scenario, the hypothetical power consumption tends to highly correlate to the power traces. For example, using all 8 bits of an S-box output as the intermediate variable in a CPA attack on an 8-bit AES implementation is more efficient than using only the most significant bit (Brier et al., 2004). However, this makes the attack computationally infeasible for large word size such as 32 bits, since the CPA needs to be performed a large number of times.

[2] https://github.com/ascon/ascon-c/tree/v1.2/crypto_aead/ascon128v12/armv6

The CPA attacks on Ascon in the literature (Samwel and Daemen, 2017; Weissbart and Picek, 2023; Roussel et al., 2023) only use one bits z_4^j and z_1^j of z_4 and z_1 for the hypothetical power consumption. These attacks were still successful because, as pointed out by Brier *et al.* (Brier et al., 2004), a partial correlation still exists if only a part of the word is used. Specifically, the partial correlation coefficient ρ_d calculated from d independent bits among m bits ($d \leq m$) and the correlation coefficient ρ_m calculated from all m bits has a the following relation:

$$\rho_d = \rho_m \sqrt{\frac{d}{m}}.$$

The attacks in (Samwel and Daemen, 2017; Weissbart and Picek, 2023; Roussel et al., 2023) correspond to $d = 1$. The above equation also indicates that increasing the value of d results in the partial correlation ρ_d becoming closer to ρ_m. In this section, we extend z_4^j and z_1^j in Equation 6 and Equation 7 to multi-bit selection functions ($d > 1$) for CPA attacks on software implementations (in common architectures where m can be 8, 16, 32 or 64 bits). In Subsection 3.1, we provide the details of the extension and its advantages. Then, we present the experiment for this extension in Subsection 3.2 and discuss the results in Subsection 3.3.

3.1 Extension method

We consider the selection function $\tilde{z}_4^j$ in Equation 6 here. The same approach is applied for $\tilde{z}_1^j$. We present the practical results for both of them. Let $z_4^{j..j+d}$ denote d bits of the 64-bit variable z_4, from index j to $j+d-1$ ($d \geq 1$). We extend the 1-bit function $\tilde{z}_4^j$ (Equation 6) to the d-bit function $\tilde{z}_4^{j..j+d}$ as follows:

$$\begin{aligned}\tilde{z}_4^{j..j+d} = &\left(n_1^{j..j+d}(k_0^{j..j+d} \oplus \mathbf{1}) \oplus n_0^{j..j+d}\right) \\ &\oplus \left(n_1^{j+57..j+57+d}(k_0^{j+57..j+57+d} \oplus \mathbf{1}) \oplus n_0^{j+57..j+57+d}\right) \\ &\oplus \left(n_1^{j+23..j+23+d}(k_0^{j+23..j+23+d} \oplus \mathbf{1}) \oplus n_0^{j+23..j+23+d}\right).\end{aligned} \quad (8)$$

For better understanding, Figure 3 depicts the bit locations involving in the computation of $\tilde{z}_4^{j..j+d}$. Using this d-bit selection function, we can recover $3d$ bits of the key, $k_0^{j..j+d}$, $k_0^{j+57..j+57+d}$, and $k_0^{j+23..j+23+d}$, after each CPA run. This means that the higher value of d, the more recovered key bits. These $3d$ key bits are indexed by the following set determined by the value of j:

$$\{j, \ldots, j+d-1\} \cup \{j+57, \ldots, j+56+d\} \cup \{j+23, \ldots, j+22+d\},$$

	Number of bits			
Selection function	d	1	2	3
Key recovery	$3d$	3	6	9
Involved nonce bits	$6d$	6	12	18
Number of CPA runs		**48**	**26**	**19**

Table 1: Number of CPA runs for the full key recovery.

where $0 \leq j \leq 63$ (with additions implicitly modulo 64). To recover the full 64-bit k_0, we need to run the CPA multiple times with different values of j such that the recovered key bit indexes cover the range from 0 to 63. This implies that the effort required for the attack can be minimized by determining the minimum number of CPA runs. In other words, we try to minimize the overlaps among the recovered key bit indexes. For example, when $d = 2$, performing CPA with $j = 0$ and $j = 34$ leads to the recovered of the key bits at indexes (0,1,57,58,23,24) and (34,35,27,28,57,58). We see that the bits indexed by 57 and 58 are recovered twice, which is redundant.

To address this, we formalize a set cover problem and employ a SAT solver to solve it.[3] A similar approach is applied to find the minimum number of CPA runs for the full 64-bit k_1 recovery. The total number of CPA runs for recovering the full 128-bit key is the sum of those of k_0 and k_1. Appendix B includes the sets of indexes j from our finding.

Table 1 compares the results for different values of d. While our extension is generic and works with any d, this paper focuses on specific cases where $d \in \{1,2,3\}$. This is due to the need of traversing all 2^{6d} possible nonce values to evaluate the distribution of hypothetical power consumption for our analyses in the next sections. This becomes computationally infeasible on classical computers when $d \geq 4$. We will detail this issue in the subsequent section.

From Table 1, we can infer the following advantages of CPA using multi-bit selection functions ($d > 1$):

- The number of CPA runs needed is significantly reduced. Specifically, for $d = 1$, 48 CPA runs are required, while this number decreases to 26 for $d = 2$ and 19 for $d = 3$.
- As a result of the reduction in the number of CPA runs, using multi-bit selection functions enables *second-order success*. This involves selecting the top two candidates with the highest correlation after each CPA run, rather than just the single top candidate. A brute-force search is then performed across these candidates to determine the correct key. The brute-force space is 2^{26} for $d = 2$ and 2^{19} for $d = 3$, both of which are computationally feasible. In contrast, the brute-force space for $d = 1$, 2^{48}, likely remains inefficient on a classical computer.

[3]The source code is included in `https://github.com/nvietsang/multibitcpa-ascon`

3.2 Experimental results

We now conduct an experiment to validate our extension. We perform the CPA using selection functions corresponding to $d = 1$, $d = 2$ and $d = 3$ and measure the success rate for each case. Measuring success rate for the full key recovery is time-consuming, as it requires repeating the analysis numerous times (typically a hundred or more) for various numbers of traces. Instead, we estimate this success rate by focusing on the recovery of $3d$ bits of k_0 and $3d$ bits of k_1. Specifically, we repeat the $3d$-bit key recoveries 100 times for k_0 and k_1 with different indexes j. These two success rates for these partial recoveries are each raised to the power of the number of CPA runs needed, and then multiplied together to estimate the success rate for the full key recovery.

We present in Figure 4 the success rates corresponding to selection functions with different values of d. In this experiment, the CPA uses the first-order success, *i.e.*, the key candidate with the highest correlation is chosen as the best candidate. For $d = 1$, the success rate begins to converge toward 100% at around 5600 traces. However, for $d = 2$ and $d = 3$, we observe an unexpected outcome: the success rates remain at 0% regardless the number of traces. This anomaly prompts a deeper investigation into the underlying cause of this behavior.

3.3 Factors influencing performance

We now investigate the cause behind the poor performance of multi-bit selection functions, as observed in Figure 4. While we detail here the analysis for $d = 2$, the same analysis approach is applicable for $d > 2$. We first consider a CPA run and look at the correlations between the hypothetical power consumption for each key candidate and the recorded power traces. The multi-bit selection function with $d = 2$ is supposed to recover $3d = 6$ key bits after each CPA run. There are thus $2^6 = 64$ possible key candidates.

Figure 5 shows the convergence of the correlations for all key candidates as the number of traces increases. It is evident that a small group of key candidates, containing the correct one, is distinguishable from the rest. This behavior is unlike the traditional case, where only the correct candidate exhibits a distinctly high correlation. Even when experimented with a larger number of traces, this outcome remains

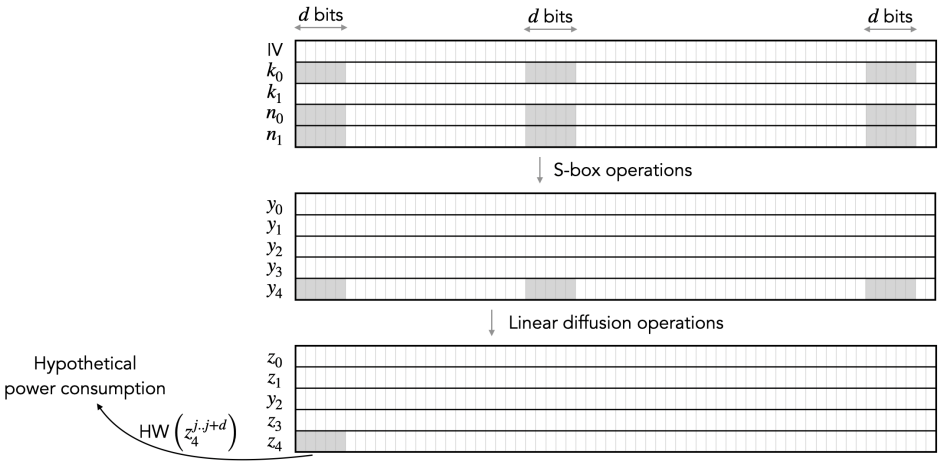


Figure 3: Illustration of d-bit selection function (Equation 8) for $j = 0$.

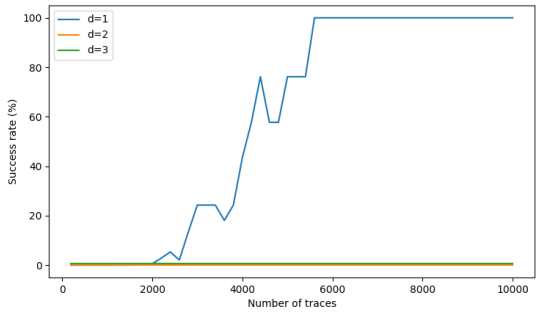


Figure 4: Success rates of the full key recovery for different values of d.

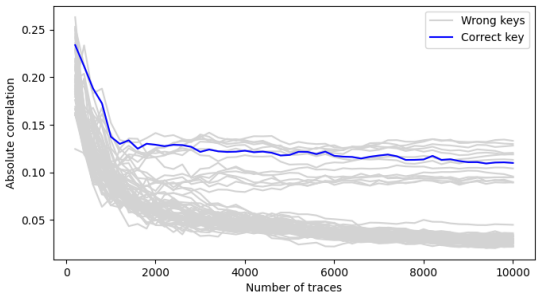


Figure 5: Correlations for all key candidates over increasing number of traces.

unchanged.

Table 2 shows the ranks of the key candidates corresponding to Figure 5, sorted by their correlations. We can align that the distinct group includes 15 key candidates. Notably, the correlation of the correct candidate is not the highest and is close to those of the other candidates in the group. Consequently, the CPA picks the incorrect candidate at the top-ranked choice, leading to the success rate of 0% as shown in Figure 4. However, this issue does not occur with the CPA using the 1-bit selection function ($d = 1$), as its success rate still converges to 100%. This suggests that the root cause may lie in the differences between

Rank	Key	Corr.	Rank	Key	Corr.
1	10	0.133	33	6	0.030
2	32	0.130	34	26	0.030
3	2	0.128	35	27	0.030
4	42	0.121	36	35	0.029
5	40	0.119	37	53	0.029
6	8	0.119	38	55	0.029
7	34	0.113	39	44	0.028
8	0	0.110	40	3	0.028
9	1	0.104	41	57	0.028
10	5	0.096	42	50	0.028
11	4	0.095	43	45	0.028
12	17	0.095	44	37	0.027
13	16	0.090	45	41	0.027
14	21	0.089	46	12	0.026
15	20	0.089	47	46	0.026
16	31	0.045	48	49	0.026
17	60	0.036	49	33	0.026
18	61	0.035	50	54	0.026
19	15	0.035	51	47	0.026
20	23	0.034	52	58	0.026
21	11	0.033	53	48	0.026
22	19	0.033	54	62	0.026
23	22	0.033	55	13	0.025
24	39	0.033	56	28	0.025
25	7	0.033	57	24	0.025
26	43	0.032	58	25	0.023
27	18	0.032	59	59	0.023
28	56	0.031	60	51	0.023
29	14	0.030	61	38	0.023
30	30	0.030	62	63	0.022
31	36	0.030	63	9	0.022
32	52	0.030	64	29	0.022

Table 2: Key candidates ranked by correlations at 10000 traces. The correct candidate is highlighted in blue.

the selection functions.

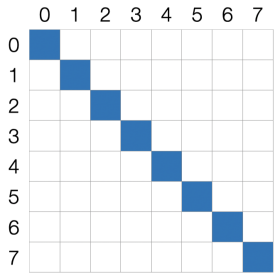


Figure 6: Correlations between distributions of all possible key pairs when $d = 1$. Blue and white cells correspond to correlation coefficient of 1 and 0, respectively.

Table 2 further indicates that, in addition to the correct key, several incorrect candidates are also highly correlated to the power traces. To investigate this, we analyze the distributions generated by each key candidate across all possible values of the associated nonce bits. For every possible key pair, we calculate the correlation of their distributions to determine whether they are partially correlated. Figure 6 presents the results of this calculation for $d = 1$, and Figure 7 presents the results for $d = 2$. It is clear that, for $d = 2$, the distributions of some key pairs are partially correlated (represented by light blue cells in Figure 7). In contrast, this behavior does not occur for $d = 1$. Furthermore, we observe that the small group of key candidates with high correlations in the CPA corresponds to the group of key candidates whose distributions are partially correlated to that of the correct key. For example, the top 15 candidates in Table 2 align with the (light) blue cells in the first row of Figure 7, indicating the correlations between correct key's distribution (0 in this case) and those of the other key candidates. This confirms the influence of the partial correlations between the key candidates.

These partial correlations are absent for $d = 1$, implying that the root cause lies in the extension of the selection function to $d > 1$. Indeed, the extension can be seen as concatenating multiple identical 1-bit selection functions, which introduces the partial correlations between the distributions of key candidates observed when $d > 1$. In fact, this concatenation is unavoidable if we want to use multi-bit selection functions due to the bitsliced-oriented design of Ascon. Specifically, the S-box operates in the vertical dimension, taking 1 bit per 64-bit variable as input and producing 1 bit per 64-bit variable as output (see Figure 2b). Meanwhile, the words (whose activities consume power) store bits in the horizontal dimension, representing the bits within a 64-bit variable (see Figure 2a). In other words, the bits in a variable have the involvement of multiple independent S-box operations.

Given the poor performance of the multi-bit selection functions compared to the 1-bit selection function, as shown in Figure 4, the question arises: is it still possible to use them for key recovery? We will give a positive answer for this question in the next section.

4 EFFICIENT KEY RECOVERY ALGORITHM

As established in the previous section, CPA with multi-bit selection functions fails in key recovery due to partial correlations among the distributions produced by many key candidates. In this section, we show how these partial correlations can be utilized to develop an efficient key recovery algorithm. The proposed algorithm is described in detail in Subsection 4.1 and experimentally validated in Subsection 4.2.

4.1 Proposed algorithm

We use the case $d = 2$ for explanation of the algorithm's idea hereafter, however, the proposed algorithm is generic and works with any $d > 1$. We present the results for both $d = 2$ and $d = 3$ below. From Figure 7, we observe that the distribution of each key candidate is (partially) correlated to itself and those of 14 other candidates (15 non-white cells in each row). These 15 candidates form a *group* which is unique for each key candidate. For example, the groups corresponding to the candidates 0 and 1, denoted by $\mathcal{G}[0]$ and $\mathcal{G}[1]$, are:

$$\mathcal{G}[0] = [0,1,2,4,5,8,10,16,17,20,21,32,34,40,42],$$
$$\mathcal{G}[1] = [0,1,3,4,5,9,11,16,17,20,21,33,35,41,43].$$

We note that the total 2^{3d} groups, $\mathcal{G}[0],\ldots,\mathcal{G}[2^{3d}-1]$, can be precomputed from the correlation matrix of distributions (as in Figure 7).

Let R be a ranking array containing all 2^{3d} key candidates, sorted by descending correlations after the classical CPA steps in Section 2. $R[0]$ holds the candidate with the highest correlation (rank 1). An example of R is the column of key candidates in Table 2. Let t be the number of elements in each group $\mathcal{G}[k]$, where $0 \le k \le 2^{3d}-1$. In our experiment, $t = 15$ for $d = 2$ and $t = 169$ for $d = 3$, and these values are the same for all groups.

The core idea of the algorithm is to determine the group $\mathcal{G}[k]$ that the top t candidates $R[0],\ldots,R[t-1]$ most likely belong to. The candidate k is then returned as the best choice for the correct key. We refer

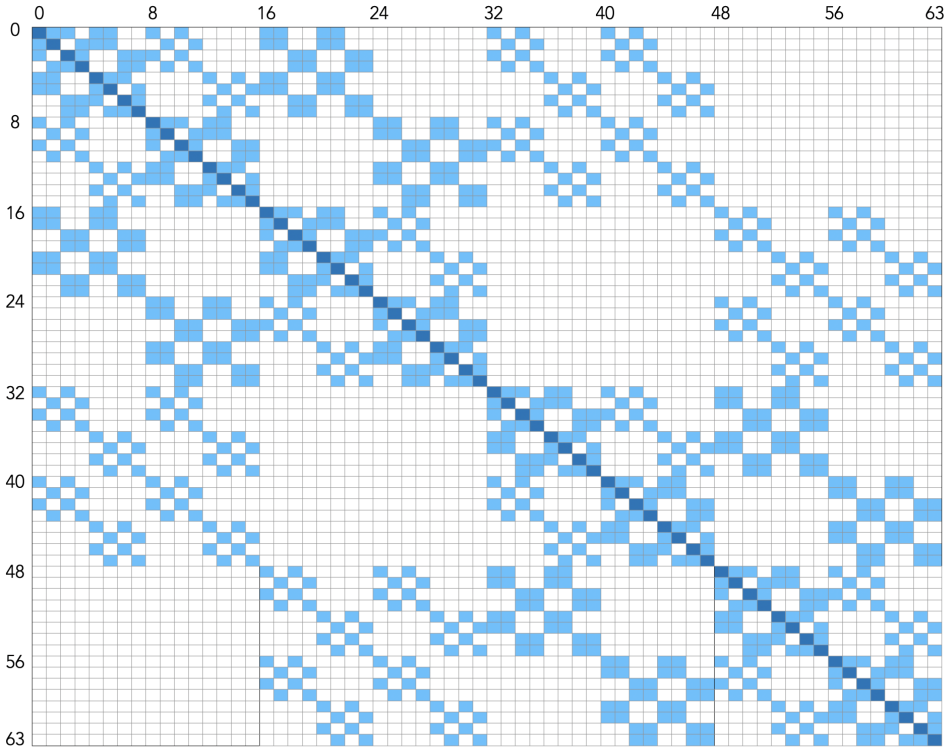


Figure 7: Correlations between distributions of all possible key pairs when $d = 2$. Blue, light blue and white cells correspond to correlation coefficient of 1, 0.5 and 0, respectively.

Data: set of traces $\mathcal{T}$, number of bits d,
groups $\mathcal{G}$, threshold t, success order o
Result: top o key candidates
$R \leftarrow$ classical CPA on $\mathcal{T}$;
$S \leftarrow [0, 0, \ldots, 0]$;
for *i from 0 to* $t-1$ **do**
  for *k from 0 to* $2^{3d}-1$ **do**
    if $R[i] \in \mathcal{G}[k]$ **then**
      $S[k] \leftarrow S[k]+1$;
    end
  end
end
$L \leftarrow$ indexes of elements in S sorted by descending scores;
$L[0], \ldots, L[o-1]$;

Algorithm 1: Key recovery for CPA using multi-bit selection functions ($d > 1$)

to t as a *threshold* in the algorithm. To measure the degree of the matching between the top t candidates and a group $\mathcal{G}[k]$, a score $S[k]$ is used, for $0 \leq k \leq 2^{3d}-1$. Each score $S[k]$ ranges from 0 to t, where t denotes a perfect match. Algorithm 1 presents the details of the key recovery steps. In this algorithm, we use a parameter o for the success order. It returns o key candidates with the top o highest scores in S.

Let us take an example where $d = 2$. Suppose R contains 64 key candidates ranked as in Table 2. By applying the scoring strategy described in Algorithm 1, we obtain the scores shown in Table 3. Here, $S[0] = 15$ is the highest score, implying that $k = 0$ is the most likely correct key candidate. This is because the elements $R[0], \ldots, R[14]$ perfectly match the group $\mathcal{G}[0]$. This outcome aligns with our expectation, as $k = 0$ is indeed the correct key.

Although our method is generic for any $d > 1$, a bottleneck lies in the reliance on precomputed groups $\mathcal{G}[k]$, where $0 \leq k \leq 2^{3d}-1$. This computation requires determining the distribution of the selection function output for all key candidates. Specifically, it necessitates evaluating the selection function for each of 2^{6d} associated nonce values and 2^{3d} key candidates, resulting a time complexity of 2^{9d}. A high d value makes this computation challenging for a classical computer. For instance, the time complexity reaches 2^{36} when $d = 4$, which may exceed the capabilities of a classical computer. Therefore, we limit our experiments in the next section to $d = 2$ and $d = 3$, which are computationally feasible on our personal laptop.

Candidate k	Score $S[k]$	Candidate k	Score $S[k]$
0	15	32	8
1	8	33	2
2	8	34	8
3	2	35	2
4	8	36	2
5	8	37	2
6	2	38	2
7	2	39	2
8	8	40	2
9	2	41	8
10	8	42	2
11	2	43	8
12	2	44	2
13	2	45	2
14	2	46	2
15	2	47	2
16	8	48	2
17	8	49	2
18	2	50	2
19	2	51	2
20	8	52	2
21	8	53	2
22	2	54	2
23	2	55	2
24	2	56	2
25	2	57	2
26	2	58	2
27	2	59	2
28	2	60	2
29	2	61	2
30	2	62	2
31	2	63	2

Table 3: Scores of 64 key candidates for $d = 2$.

4.2 Experimental results

We now conduct an experiment to validate the proposed key recovery algorithm. The experimental setup is identical to that described in the previous section. In particular, we reuse the same set of traces and apply the proposed algorithm for multi-bit selection functions ($d = 2$ and $d = 3$). For comparison, we also perform an analysis using the classical CPA with the 1-bit selection function ($d = 1$).

Figure 8 shows the success rates of the full key recovery using the d-bit selection functions with different values of d. For multi-bit selection functions ($d = 2$ and $d = 3$), their success rates gradually converge to 100% as the number of traces increases, in contrast to the 0% success rates observed in Figure 4. Specifically, the successes rates reach 100% for $d = 2$ with 7200 traces and for $d = 3$ with 8200 traces. This demonstrates the effectiveness of our proposed key recovery algorithm.

Nevertheless, compared to the 1-bit selection function, the multi-bit selection functions require more traces to achieve 100% success rates. Additionally, it can be observed that the convergence speed decreases with increasing d. This is likely because more traces are needed to make the correlations associated to a group of key candidates stand out from the rest. A higher d corresponds to a bigger group. Meanwhile, the 1-bit selection function only requires the correlation of a single key candidates to stand out.

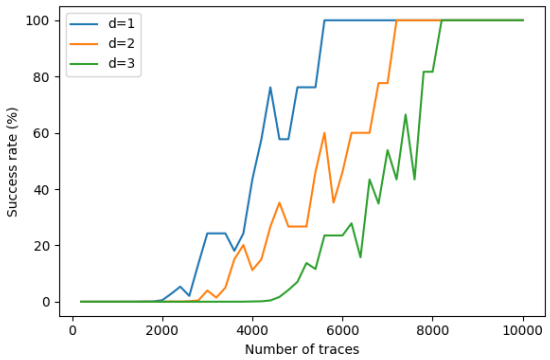
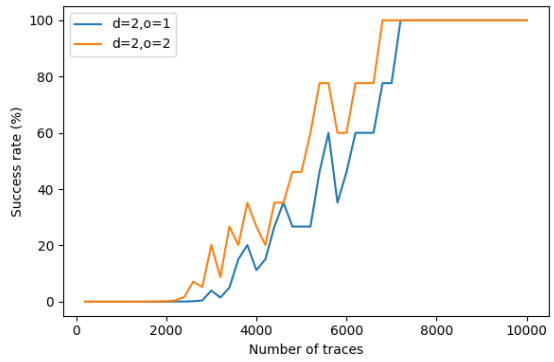


Figure 8: Full key recovery success rates after applying efficient algorithm for $d = 2$ and $d = 3$. The success order is $o = 1$ for all cases.

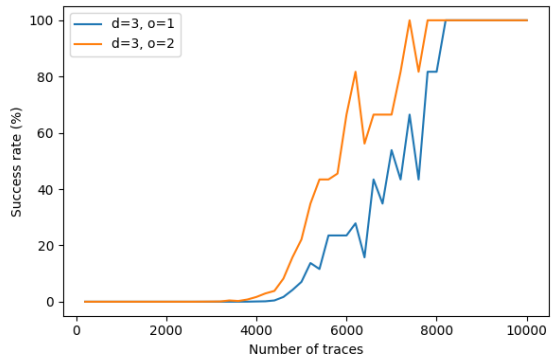
As discussed in Subsection 3.1, there are two main advantages of using multi-bit selection functions, including the significant reduction in the number of CPA runs and the application of second-order success. We now consider the latter. It can be seen that Algorithm 1 is generic for o-order success. We conduct an experiment for the case where $o = 2$ (*i.e.*, second-order) with $d = 2$ and $d = 3$. Figure 9 compares the success rates for $o = 2$ and $o = 1$. As expected, the success rates for $o = 2$ are higher than those for $o = 1$ in both cases.

5 CONCLUSION & DISCUSSION

In this paper, we extended the 1-bit selection function to a multi-bit selection function. Initially, this extension caused the CPA to fail in identifying the correct key candidate due to Ascon's bitsliced-oriented design. To address this, we conducted a comprehensive analysis from both theoretical and experimental perspectives to uncover the reasons behind this failure. Leveraging the insights from this analysis, we proposed an efficient key recovery algorithm tailored for the multi-bit selection function. We further provided experimental results demonstrating the effectiveness of this algorithm. Additionally, we showed that the multi-bit selection function offers advantages, including a reduction in the number of CPA runs required for full key recovery and the ability to apply second-order success in CPA.



(a) $d = 2$



(b) $d = 3$

Figure 9: Full key recovery success rates with first- and second-order successes for $d = 2$ and $d = 3$.

Countermeasures. CPA attacks exploit the dependency between a device's power consumption and intermediate values of the executed cryptographic algorithms. A well-known mitigation strategy is to eliminate or at least reduce this dependency. One approach involves randomizing power consumption by performing intermediate computations at different time moments. This can be achieved by randomly inserting dummy operations during execution to disrupt the power trace alignment, or by shuffling the operations (Kocher et al., 1999). Another widely studied approach is masking intermediate values with randomness (Chari et al., 1999; Goubin and Patarin, 1999), ensuring that power consumption is independent of these intermediate values. This technique is typically implemented at the algorithm level.

Future work. A potential direction for future work is the development of a more efficient key recovery algorithm for the multi-bit selection function. As discussed earlier, while our proposed algorithm is effective for key recovery, it becomes computationally infeasible for $d > 3$. Additionally, the resulting success rates remain lower than those achieved with the 1-bit selection function, while we expect that incorporating more bits into the hypotheses for power consumption could improve success rates. A promising direction to address these challenges is exploring machine learning-based and profiling-based techniques.

ACKNOWLEDGEMENTS

This work was supported by the French *Agence Nationale de la Recherche* through the grant ANR-22-CE39-0008 (project PROPHY).

REFERENCES

AES (2001). Advanced Encryption Standard (AES). National Institute of Standards and Technology, NIST FIPS PUB 197, U.S. Department of Commerce.

Bertoni, G., Daemen, J., Debande, N., Le, T.-H., Peeters, M., and Van Assche, G. (2012a). Power analysis of hardware implementations protected with secret sharing. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture Workshops*, pages 9–16.

Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. (2012b). Duplexing the sponge: Single-pass authenticated encryption and other applications. In Miri, A. and Vaudenay, S., editors, *SAC 2011*, volume 7118 of *LNCS*, pages 320–337. Springer, Berlin, Heidelberg.

Brier, E., Clavier, C., and Olivier, F. (2004). Correlation power analysis with a leakage model. In Joye, M. and Quisquater, J.-J., editors, *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, Berlin, Heidelberg.

Chari, S., Jutla, C. S., Rao, J. R., and Rohatgi, P. (1999). Towards sound approaches to counteract power-analysis attacks. In Wiener, M. J., editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 398–412. Springer, Berlin, Heidelberg.

Dobraunig, C., Eichlseder, M., Mendel, F., and Schläffer, M. (2021). Ascon v1.2: Lightweight authenticated encryption and hashing. *Journal of Cryptology*, 34(3):33.

Goubin, L. and Patarin, J. (1999). DES and differential power analysis (the "duplication" method). In Koç, Çetin Kaya. and Paar, C., editors, *CHES'99*, volume 1717 of *LNCS*, pages 158–172. Springer, Berlin, Heidelberg.

Kocher, P. C., Jaffe, J., and Jun, B. (1999). Differential power analysis. In Wiener, M. J., editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, Berlin, Heidelberg.

Ramezanpour, K., Abdulgadir, A., Diehl, W., Kaps, J.-P., , and Ampadu, P. (2020). Active and passive side-channel key recovery attacks on Ascon. NIST Lightweight Cryptography Workshop.

Roussel, N., Potin, O., Dutertre, J., and Rigaud, J. (2023). Security evaluation of a hybrid CMOS/MRAM as-

con hardware implementation. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2023, Antwerp, Belgium, April 17-19, 2023*, pages 1–6. IEEE.

Samwel, N. and Daemen, J. (2017). DPA on hardware implementations of Ascon and Keyak. In *Proceedings of the Computing Frontiers Conference*, CF'17, page 415–424, New York, NY, USA. Association for Computing Machinery.

Tunstall, M., Hanley, N., McEvoy, R., Whelan, C., Murphy, C., and Marnane, W. (2007). Correlation power analysis of large word sizes. `http://www.geocities.ws/mike.tunstall/papers/THMWMM.pdf`.

Weissbart, L. and Picek, S. (2023). Lightweight but not easy: Side-channel analysis of the ascon authenticated cipher on a 32-bit microcontroller. Cryptology ePrint Archive, Paper 2023/1598. `https://eprint.iacr.org/2023/1598`.

APPENDIX A: SELECTION FUNCTIONS

The j-th bit of y_1 and y_4 are computed as:

$$y_0^j = k_0^j(n_1^j \oplus 1) \oplus n_0^j \oplus k_0^j k_1^j \oplus k_0^j \mathtt{IV}^j \oplus k_1^j \oplus \mathtt{IV}^j,$$
$$y_1^j = n_0^j(k_1^j \oplus k_0^j \oplus 1) \oplus n_1^j \oplus k_1^j k_0^j \oplus k_1^j \oplus k_0^j \oplus \mathtt{IV}^j.$$

In y_0^j, we remove $k_0^j k_1^j \oplus k_0^j \mathtt{IV}^j \oplus k_1^j \oplus \mathtt{IV}^j$ as they contribute a constant amount to the power consumption. For the same reason, $k_1^j k_0^j \oplus k_1^j \oplus k_0^j \oplus \mathtt{IV}^j$ is removed from y_1^j.

$$\tilde{y}_0^j = k_0^j(n_1^j \oplus 1) \oplus n_0^j,$$
$$\tilde{y}_1^j = n_0^j(k_{01}^j \oplus 1) \oplus n_1^j,$$

where $k_{01}^j = k_0^j \oplus k_1^j$.

Recall the linear operations applied on the y_0 and y_1:

$$z_0 = y_0 \oplus (y_0 \ggg 19) \oplus (y_0 \ggg 28),$$
$$z_1 = y_1 \oplus (y_1 \ggg 61) \oplus (y_1 \ggg 39).$$

The j-th bit of z_0 and z_1 are thus computed as:

$$z_0^j = y_0^j \oplus y_0^{j+36} \oplus y_0^{j+45},$$
$$z_1^j = y_1^j \oplus y_1^{j+3} \oplus y_1^{j+25}.$$

We then apply the linear operations for $\tilde{y}_0^j$ and $\tilde{y}_1^j$:

$$\begin{aligned} \tilde{z}_0^j = &\left(k_0^j(n_1^j \oplus 1) \oplus n_0^j\right) \\ &\oplus \left(k_0^{j+36}(n_1^{j+36} \oplus 1) \oplus n_0^{j+36}\right) \\ &\oplus \left(k_0^{j+45}(n_1^{j+45} \oplus 1) \oplus n_0^{j+45}\right). \end{aligned} \tag{9}$$

$$\begin{aligned} \tilde{z}_1^j = &\left(n_0^j(k_{01}^j \oplus 1) \oplus n_1^j\right) \\ &\oplus \left(n_0^{j+3}(k_{01}^{j+3} \oplus 1) \oplus n_1^{j+3}\right) \\ &\oplus \left(n_0^{j+25}(k_{01}^{j+25} \oplus 1) \oplus n_1^{j+25}\right). \end{aligned} \tag{10}$$

APPENDIX B: KEY INDEXES FOR CPA RUNS

The script of the SAT problem is included in `https://github.com/nvietsang/multibitcpa-ascon`. We present here the minimum number of CPA runs for different values of d.
For $d = 1$:

- 24 indexes j for k_0 recovery: 1, 6, 7, 9, 11, 12, 17, 22, 23, 25, 27, 28, 33, 34, 38, 39, 43, 44, 48, 49, 54, 55, 59, 60.
- 24 indexes j for k_1 recovery: 3, 5, 6, 7, 13, 15, 17, 22, 24, 26, 32, 33, 34, 39, 41, 43, 45, 50, 51, 52, 53, 58, 60, 62.

For $d = 2$:

- 12 indexes j for k_0 recovery: 3, 9, 14, 19, 24, 30, 35, 41, 46, 51, 56, 62.
- 14 indexes j for k_1 recovery: 2, 10, 11, 20, 22, 28, 30, 37, 39, 43, 46, 48, 55, 57.

For $d = 3$:

- 10 indexes j for k_0 recovery: 20, 23, 33, 36, 39, 42, 45, 48, 51, 60.
- 9 indexes j for k_1 recovery: 1, 7, 13, 20, 29, 32, 39, 48, 58.