Practical Second-Order CPA Attack on Ascon with Proper Selection Function

Viet Sang Nguyen, Vincent Grosso, and Pierre-Louis Cayrel

Laboratoire Hubert Curien, Université Jean Monnet, Saint-Étienne 42100, France
viet.sang.nguyen@univ-st-etienne.fr
vincent.grosso@univ-st-etienne.fr
pierre.louis.cayrel@univ-st-etienne.fr

Abstract. Ascon has recently been selected by the National Institute of Standards and Technology (NIST) as the lightweight cryptography standard. Consequently, it is utilized in a multitude of environments and devices. In this study, we examine the potential vulnerability of Ascon software implementations to Correlation Power Analysis (CPA) attacks. First, we conduct a comprehensive analysis of different approaches from the literature for choosing the selection function used to compute intermediate values in a CPA attack. Through both theoretical explanation and experimental validation, we demonstrate how these choices influence the success of the attack. Second, leveraging insights from our analysis, we present, to the best of our knowledge, the first successful and practical second-order CPA attack on a masked software implementation provided by the Ascon team running on a 32-bit microcontroller. Our results show that the full 128-bit key can be recovered in 7 hours through the analysis of 360,000 traces on classical laptop.

Keywords: Correlation Power Analysis · Ascon · Masking.

1 Introduction

Nowadays, the use of small computing devices, such as RFID tags, sensors, and smart cards, is increasingly common. Although being time-proven to be a robust cipher, the Advanced Encryption Standard (AES) [1] is often too resource-intensive for deployment in such low-end devices. This limit of the AES highlights the need for a more *lightweight* cipher. In this context, NIST initiated a lightweight cryptography competition to seek a new standard. After a rigorous selection process, Ascon [9] was announced as the new standard for lightweight cryptography in February 2023. Prior to this, Ascon had also been included in the final portfolio of the CAESAR competition. The careful analysis in these two selection processes enforces the confidence of the security of Ascon in a *black-box model*, in which the adversary only has access to the inputs and outputs.

However, the black-box model is not always sufficient to ensure security in practice. Especially when implemented and executed in embedded devices, cryptographic algorithms can be vulnerable to Side-Channel Attacks (SCA), which exploit *physical leakages* from the devices (*e.g.*, power consumption, execution time, electromagnetic radiations). Since the introduction of Differential Power Attack (DPA) by Kocher *et al.* [12], power analysis attacks have become a prominent research area. Over the years, many attack techniques have been developed, for example, Template Attacks (TA) by Chari *et al.* [8], Correlation Power Analysis (CPA) by Brier *et al.* [6], Mutual Information Analysis (MIA) by Batina *et al.* [2], Soft Analytical Side-Channel Analysis (SASCA) by Veyrat-Charvillon *et al.* [22], Deep Learning Side-Channel Attacks (DLSCA) by Maghrebi *et al.* [14].

With the expected widespread deployment of Ascon in embedded devices, where power leakages pose a significant threat, the need for studies on power analysis attacks against implementations of Ascon is growing. So far, there has not been much attention on this research area for Ascon. Samwel and Daemen [20] introduced the first successful CPA attack on a noisy hardware implementation. In their work, the authors constructed an effective *selection function* for computing the *intermediate variable* targeted by the attacks. Using the same selection function, Roussel *et al.* [19] and Weissbart and Picek [24] also successfully performed CPA attacks on a hybrid CMOS/MRAM hardware implementation and a ARMv7m software implementation, respectively. Ramezanpour *et al.* [17] conducted DPA and CPA attacks with a different selection function, but reported that they failed to recover the key. The authors later proposed a deep learning-based power analysis, which succeeded in key recovery. You *et al.* [25] introduced an efficient template attack on a 32-bit software implementation. Lou *et al.* [13] presented a SASCA attack with simulated traces for an 8-bit implementation.

To protect against power analysis attacks, *masking* [7,10] is the most widely used countermeasure. The core concept behind masking is to split the sensitive variables into multiple shares and carry out the computations on these shares. Ascon's design, which features an efficient bitsliced implementation of the S-boxes, facilitates the use of masking. Several masked software implementations were published by the Ascon team.[1] Note that masking increases the attack complexity rather than offering complete protection. A method of attacking masked schemes is to combine the leakages from the individual shares and perform a CPA on the aggregated leakages. This technique is known as *higher-order* CPA [11,15,23]. When two shares are involved, the attack is referred to as a *second-order* CPA. Weissbart and Picek [24] attempted to perform a second-order CPA on a masked software implementation of Ascon, but reported a failure. The authors then proposed a successful deep learning-based power analysis, which was later improved by Rezaeezade *et al.* [18].

Most of the state-of-the-art power attacks on (protected) implementations of Ascon, including template attack [25], SASCA [13], deep learning attacks [24,18], are *profiled attacks*, in which the powerful adversary is assumed to have a full control on a copy of the targeted device and can obtain *a priori* knowledge about the implementation details. In contrast, the CPA attack, which our work focuses on, is a *non-profiled attack* corresponding to a weaker adversary. The adversary is only able to observe the device's leakages, *i.e.*, the power consumption when

[1] See `https://github.com/ascon/simpleserial-ascon`

the cryptographic algorithm is executed. No detailed knowledge about the device is required. The goal of CPA is to recover the key using a statistical analysis on the key-dependent physical leakages.

An important factor that significantly affects the success of a CPA attack is the choice of the selection function for computing intermediate values. In the literature, different approaches of choosing this function have been proposed, leading to either a success [20] or a failure [17] in key recovery. One approach relies on heuristics, such as using the S-box computation as the selection function [17], which has been well-studied in CPA attacks on AES. Another approach derives from observing of how processed data leaks into the power consumption [20]. In both cases [20,17], whether successful or not, there is a notable lack of analysis regarding the impact of these choices of the selection function on the success of the CPA attack.

Contributions. First, we provide a comprehensive analysis of the selection functions used in the literature. Through both theoretical explanation and experimental validation, we demonstrate that different choices of the selection function can determine the success or failure of a CPA attack on Ascon. Second, leveraging insights from our analysis, we present, to our knowledge, the first successful and practical second-order CPA attack against a masked software implementation of Ascon. We detail how the attack can be performed with modest resource requirements. To validate our attack path, we use the 32-bit ARMv6 masked implementation provided by the Ascon team. Power traces are recorded from executions of this implementation on a 32-bit STM32F303 microcontroller. Our results show that the full 128-bit key can be recovered successfully in 7 hours using 360K traces.

For the sake of reproducibility, we publish the source code of the experiments as well as traces at:

`https://anonymous.4open.science/r/K572-ULB2-B1B0/`

Outline. This paper is organized as follows. Section 2 provides the background knowledge. Section 3 presents a thorough analysis of the selection functions used in the literature. Section 4 presents the practical second-order CPA attack. Finally, Section 5 concludes our work and provides some perspectives.

2 Preliminaries

In this section, we first briefly recall the principle of the Correlation Power Analysis (CPA) attacks. Next, we present the self-contained background of the Ascon cipher. Finally, we provide the information on the devices and the setup in our experiments.

2.1 CPA attacks

The goal of CPA is to recover the key based on a number of power traces recorded while the cryptographic algorithm is executed. The main advantage of CPA is that it does not require detailed knowledge about the cryptographic device. Knowing the algorithm that is executed by the device is usually sufficient. CPA attacks analyze the dependence between the power consumption at specific moments and the processed data. The attack procedure consists of the following five steps:

1. Choose an intermediate variable of the executed algorithm as the attack point. This intermediate variable needs to be a function $f(d, k)$, called *selection function*, of a part of the key k and the known non-constant data d (*e.g.*, plaintext).
2. Measure the power consumption of the device while it executes the cryptographic algorithm ℓ times. For each execution, the adversary records the data value d involved in the selection function and a power trace of s samples. Then, ℓ data values are written as a vector $\mathbf{d} = (d_1, \ldots, d_\ell)$, and ℓ power traces are written as a matrix $\mathbf{T}$ of size $\ell \times s$. It is important to note that the traces must be correctly aligned.
3. Calculate hypothetical intermediate values for every possible candidate of k. Let $\mathbf{k} = (k_1, \ldots, k_p)$ be the vector of p possible candidates of k. Given the two vectors $\mathbf{d}$ and $\mathbf{k}$, the adversary calculates the hypothetical intermediate values with the selection function $f(d, k)$. This calculation results in a matrix $\mathbf{V}$ of size $\ell \times p$.
4. Map hypothetical intermediate values to hypothetical power consumption values. The adversary chooses a leakage model to map each value in $\mathbf{V}$ to a power consumption value. In this work, we use the Hamming weight model. This step results in a hypothetical power consumption matrix $\mathbf{H}$ of size $\ell \times p$.
5. Compare the hypothetical power consumption values with the power traces. The adversary uses the Pearson's correlation coefficient to compare the hypothetical power consumption values of each key candidate with the measured traces at every position. Specifically, he calculates the correlation coefficient between each column $\mathbf{h}_i$ of the matrix $\mathbf{H}$ and each column $\mathbf{t}_j$ of the matrix $\mathbf{T}$, resulting the element $r_{i,j}$ of the matrix $\mathbf{R}$ of size $p \times s$, where

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

   In the above equation, the values $h_{u,i}$ and $t_{u,j}$ (resp. $\overline{h}_i$ and $\overline{t}_j$) denote the u-th elements (resp. mean values) of the columns $\mathbf{h}_i$ and $\mathbf{t}_j$.

The key can be recovered based on the fact that the higher value $r_{i,j}$ indicates the better match between the columns $\mathbf{h}_i$ and $\mathbf{t}_j$. Let `ck` be the index of the correct key $k_{\texttt{ck}}$ (*i.e.*, the key that is used in the device) in the vector $\mathbf{k}$, and `ct` be the index of the power consumption values $\mathbf{t}_{\texttt{ct}}$ that depend on the intermediate values $\mathbf{v}_{\texttt{ck}}$. The columns $\mathbf{h}_{\texttt{ck}}$ and $\mathbf{t}_{\texttt{ct}}$ should be strongly correlated. Thus, the highest value $r_{\texttt{ck},\texttt{ct}}$ in the matrix $\mathbf{R}$ reveals the indexes of the correct key `ck` and the position `ct`.

2.2 Ascon

Ascon [9] is a suite of Authenticated Encryption with Associated Data (AEAD) and hashing algorithms, using the duplex sponge construction [4]. This paper considers the recommended version of authenticated cipher, Ascon-128 (referred to as Ascon throughout the paper). The encryption process of Ascon is depicted in Figure 1. It takes as input a key K of 128 bits, a nonce N of 128 bits, 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. This tag is used to authenticate the ciphertexts in the decryption process.

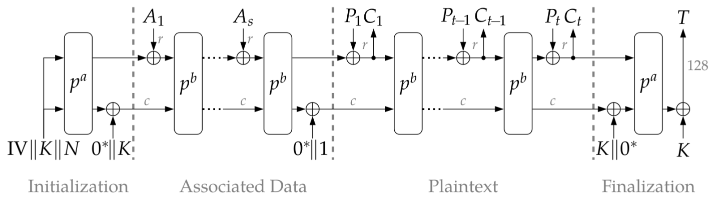
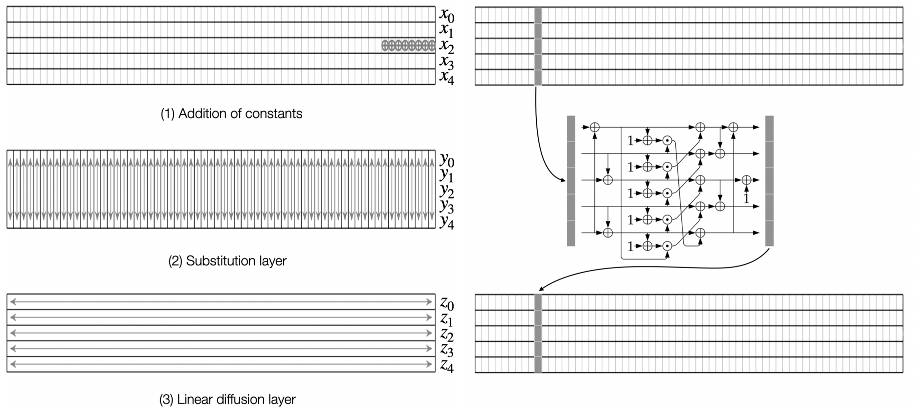


Fig. 1: Encryption in Ascon [9].



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

Fig. 2: Each step in a round [9].

The permutations, denoted by p^a and p^b, are the core of the 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, as depicted in Figure 2. The 320-bit state is split into five words of 64 bits. These words can be stored in one or more registers, facilitating the translation from mathematical description to efficient implementation.[2]

Let $x_0, \dots, x_4$ denote five 64-bit words of the round input. In the first step, a round constant is added to the rightmost eight bits of the word x_2. As the step of constant addition is not important in our attack, we simplify the notation by also denoting the output of the first step as $x_0, \dots, x_4$. The second step is a non-linear transformation operating on five bits, one bit from each word of the first step output $x_0, \dots, x_4$. Let $y_0, \dots, y_4$ denote the output state of the S-box, and $\mathbf{1}$ (in bold) denote a word of full 64 bit 1s. The algebraic normal form (ANF) of the S-box with operations performed on the entire 64-bit words (bitsliced form) can be written 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 `IV` is stored in the word x_0, the two 64-bit halves of the key $(k_0, k_1) = K$ are stored in the words x_1 and x_2, and the two 64-bit halves of the nonce $(n_0, n_1) = N$ are stored in the words x_3 and x_4. The S-box computation in the first round of the initialization phase, where our attack focuses on, thus can be written as (the constant addition step is omitted for the sake of simplicity):

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

The third step, linear diffusion, rotates each word at the S-box output twice and XORs with itself. Let $z_0, \dots, z_4$ denote the output of the linear diffusion layer. The linear functions applied to each word are:

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

[2] Implementations for 8-bit, 16-bit, 32-bit, 64-bit architectures can be found at `https://github.com/ascon/ascon-c`

2.3 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 is run with the default clock frequency 7.37 MHz. The ChipWhisperer board is connected to a MacBook Air M1 with 16 GB of RAM via a USB cable. All the analyses in this paper are also conducted on this computer. The details of the specific implementations used for each attack will be provided later.

3 Choices of selection function

The first and important step of a CPA is the selection of an intermediate variable as the attack point. This intermediate variable must be the output of a function (referred to as the selection function) that takes as input a small portion of the key and known non-constant data. As evidenced in certain prior works [20,24,17,19], an intermediate variable in the first round of the initialization phase seems well-suited for this purpose. This is due to the fact that the first round's inputs are the key and the nonce (see Figure 1), where the nonce can be regarded as the known non-constant data.

In the literature, to our knowledge, there exists two approaches of choosing the intermediate variable and the selection function in the first round for CPA attacks on Ascon. The first approach, used by Ramezanpour *et al.* [17], is to straightforwardly choose the S-box output as the intermediate variable and the S-box computation as the selection function. This is similar to the choice of S-box output as the attack point in many well-studied CPA attacks on AES. Applying this approach, Ramezanpour *et al.* reported a failure for their attack (before introducing a successful deep learning attack), but did not provide any explanation. The second approach, proposed by Daemen and Samwel [20] and later used in [24,19], is to choose the linear diffusion layer output as the intermediate variable and *fine-tune* the S-box computation for the selection function. Applying this approach, the attacks in [20,24,19] succeeded in recovering the key. Daemen and Samwel provided the rationale behind their adjustment in the S-box function to derive the selection function, but did not analyze how it impacts the success of the CPA attack. In other words, the authors did not explain why it is necessary to fine-tune the S-box computation instead of using it directly as the selection function.

In this section, we take a closer look into the two approaches. For each of them, we begin with a brief description of the selection function, and then analyze its impact on the success of the CPA attack. To simplify distinction, we refer the first approach as using the *pure* S-box computation (Subsection 3.1), and the second approach as using the *fine-tuned* S-box computation (Subsection 3.2), as the selection function.

3.1 Pure S-box computation as selection function

In Equation 2, the S-box computation is written in a bitsliced form in which 64 parallel applications of the 5-bit S-box (corresponding to the entire 64-bit words) are performed at once. For analysis, we consider a single application of the S-box. Let the superscript j denote the index of the j-th bit of a 64-bit word, where $0 \leq j \leq 63$. The computation of the five S-box output bits $y_0^j, \ldots, y_4^j$ is written as:

$$\begin{aligned}
y_0^j &= n_1^j k_0^j \oplus n_0^j \oplus k_1^j k_0^j \oplus k_1^j \oplus k_0^j \mathtt{IV}^j \oplus k_0^j \oplus \mathtt{IV}^j, \\
y_1^j &= n_1^j \oplus n_0^j k_1^j \oplus n_0^j k_0^j \oplus n_0^j \oplus k_1^j k_0^j \oplus k_1^j \oplus k_0^j \oplus \mathtt{IV}^j, \\
y_2^j &= n_1^j n_0^j \oplus n_1^j \oplus k_1^j \oplus k_0^j \oplus 1, \\
y_3^j &= n_1^j \mathtt{IV}^j \oplus n_1^j \oplus n_0^j \mathtt{IV}^j \oplus n_0^j \oplus k_1^j \oplus k_0^j \oplus \mathtt{IV}^j, \\
y_4^j &= n_1^j k_0^j \oplus n_1^j \oplus n_0^j \oplus k_0^j \mathtt{IV}^j \oplus k_0^j.
\end{aligned} \tag{4}$$

It can be observed that the values $y_0^j, y_1^j, y_2^j, y_3^j$ take as input two bits of the nonce (n_0^j, n_1^j) that are known non-constant data, and two bits of the key (k_0^j, k_1^j) that need to be guessed in our attack. The fifth bit corresponds to known constant data belonging to the `IV`. Recall that `IV` is the 64-bit initialization vector, `IV` = `80400c0600000000` in hexadecimal. A small difference in y_4^j is that only one bit of the key (k_0^j) is involved.

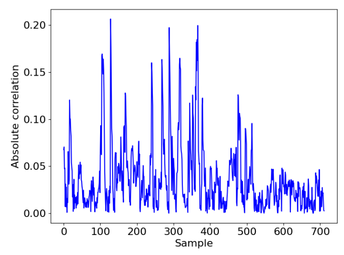
A single S-box output bit as the intermediate variable. We now analyze the impact on the success of attacks if one chooses an output bit in $y_0^j, \ldots, y_4^j$ as the intermediate variable and the S-box computation as the selection function. Let us first consider the computation of y_0^j. There are 4 possible key candidates for (k_0^j, k_1^j), and 4 possible values of the known non-constant data (n_0^j, n_1^j). We present the distribution of the y_0^j corresponding to every possible key candidate. As we can see, there are two key pairs resulting in distributions that are *fully correlated* to each other (correlation coefficient equals ± 1). We then summarize the correlations between the distributions of every possible key pair for y_0^j in Table 2 (at top left). Suppose 1 among 4 possible key candidates is the correct key. This analysis of y_0^j implies that if y_0^j is selected as the intermediate variable, there will always exist an incorrect key candidate which hypothetical power consumption values correspond to are as highly correlated with the power traces as those of the correct key. Consequently, the correct key and this incorrect key cannot be distinguished in CPA, where the hypothetical power consumption values of the correct key are expected to exhibit the highest correlation with the power traces. As in Table 1, such pairs of correct key and incorrect key, for example, are $(0,0)$ and $(0,1)$, $(1,0)$ and $(1,1)$. Figure 3 illustrates the experiment for this analysis. It can be observed that the correlation traces for the candidate pairs $(0,0)$ and $(0,1)$ (in blue), as well as $(1,0)$ and $(1,1)$ (in red), are identical. As can be observed in Table 1, the value of the `IV` bit has no impact on the correlation score. This is expected, given that when $\mathtt{IV}^j = 1$, it negates all the output of the selection function for $\mathtt{IV}^j = 0$, both for the correct key and the incorrect key.

Note that the keys in those pairs are also not distinguishable in a DPA attack, where the values of y_0^j are used to divide the traces into two sets (one for $y_0^j = 0$

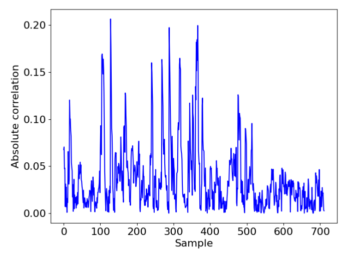
	(k_0^j, k_1^j)			
(n_0^j, n_1^j)	(0,0)	(0,1)	(1,0)	(1,1)
(0,0)	0	1	1	1
(0,1)	0	1	0	0
(1,0)	1	0	0	0
(1,1)	1	0	1	1
Correlation	-1		1	

	(k_0^j, k_1^j)			
(n_0^j, n_1^j)	(0,0)	(0,1)	(1,0)	(1,1)
(0,0)	1	0	1	1
(0,1)	1	0	0	0
(1,0)	0	1	0	0
(1,1)	0	1	1	1
Correlation	-1		1	

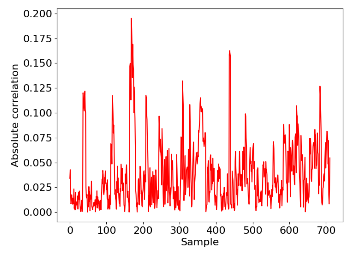
Table 1: Distribution of y_0^j corresponding to every possible key candidate when $\texttt{IV}^j = 0$ (left) and when $\texttt{IV}^j = 1$ (right).



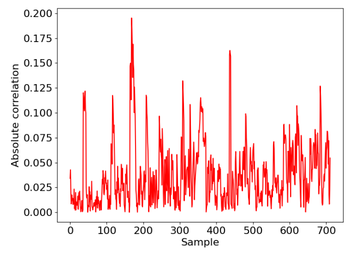
(a) $(k_0^j, k_1^j) = (0, 0)$



(b) $(k_0^j, k_1^j) = (0, 1)$



(c) $(k_0^j, k_1^j) = (1, 0)$



(d) $(k_0^j, k_1^j) = (1, 1)$

Fig. 3: Correlation traces when using y_0^j as the intermediate variable. The calculations use 1000 traces recorded from the execution of the reference implementation.

and the other for $y_0^j = 1$). For instance, the division into two sets will be identical for the two key candidates $(0, 0)$ and (0,1), or $(1, 0)$ and $(1, 1)$, as the resulting distributions of y_0^j for these candidates are identical, as shown in Table 1. As a consequence, the difference of means of the two sets will also be identical for the two candidates. In the DPA attack of Ramezanpour *et al.* [17], the authors chose y_0^j as the selection function. The authors reported that their DPA attack failed to find the correct key with more than 40K traces, but did not provide the reason. According to our analysis, their choice of the intermediate variable and the selection function could be the explanation for this failure.

(k_0^j, k_1^j)	(0,0)	(0,1)	(1,0)	(1,1)
(0,0)	1	1	-	-
(0,1)	1	1	-	-
(1,0)	-	-	1	1
(1,1)	-	-	1	1

y_0^j

(k_0^j, k_1^j)	(0,0)	(0,1)	(1,0)	(1,1)
(0,0)	1	-	-	1
(0,1)	-	1	1	-
(1,0)	-	1	1	-
(1,1)	1	-	-	1

y_1^j

(k_0^j, k_1^j)	(0,0)	(0,1)	(1,0)	(1,1)
(0,0)	1	1	1	1
(0,1)	1	1	1	1
(1,0)	1	1	1	1
(1,1)	1	1	1	1

y_2^j and y_3^j

k_0^j	0	1
0	1	0
1	0	1

y_4^j

Table 2: Absolute correlations of distributions associated to all possible key pairs for $y_0^j, y_1^j, y_2^j, y_3^j, y_4^j$ when $\texttt{IV}^j = 0$. The entries with value 0 are indicated by "-" to facilitate reading.

We conduct a similar analysis for each of y_1^j, y_2^j, y_3^j and y_4^j, and present the correlations between the distributions of all possible key pairs for each case in Table 2. The results are similar to those of y_0^j, except for y_4^j. Specifically, for any given correct key in cases of y_1^j, y_2^j and y_3^j, there will always be an incorrect key that produces the same distribution, rendering the correct key indistinguishable from the incorrect one in DPA and CPA attacks. This is even worse in the cases of y_2^j and y_3^j (represented by the table of full correlations with values of 1 in Table 2), as all other incorrect key candidates yield identical distributions (or opposite distributions corresponding to the correlation coefficient of -1). Thus, we conclude that CPA attacks (resp. DPA attacks) using y_0^j, y_1^j, y_2^j or y_3^j as the selection function, which aims to identify the key candidate corresponding to the highest value of correlation (resp. of the mean difference) as the correct one, will not succeed in obtaining a unique correct key, regardless the number of traces. We note that this conclusion considers a single S-box output bit as the selection function.

We observe an exception in the case of y_4^j. The distributions of the two possible key candidates (since only k_0^j is involved in y_4^j) are uncorrelated. This implies that y_4^j as the selection function could yield a unique key candidate given enough traces. This also suggests that if fixing, for example, $k_1^j = 0$ in y_0^j's table, or $k_0^j = 0$ in y_1^j's table, reduces the table to one similar to y_4^j. Using y_0^j and y_1^j with those fixes then could also lead to a unique key candidate. However, as we will show later in Subsection 3.2, using y_4^j or y_0^j and y_1^j with the fixes as selection functions makes the DPA and CPA attacks prone to failure.

Hamming weight of S-box output as the intermediate variable. We now consider the case where the Hamming weight of the 5-bit S-box output is used as the intermediate variable and the computations in Equation 4 as the selection function.[3] Employing the Hamming weight of the S-box output, as done by Ramezanpour *et al.* [17] in their CPA attack on Ascon, is a very common approach in CPA attacks on AES. Recall that there are still 4 possible key candidates for (k_0^j, k_1^j) and 4 possible values for (n_0^j, n_1^j). As in the above analysis, we calculate the the Hamming weight distributions of the S-box output for every key candidate:

$$\mathrm{HW}(y_0^j||y_1^j||y_2^j||y_3^j||y_4^j),$$

where $\mathrm{HW}(\cdot)$ denotes the Hamming weight and $||$ denotes the concatenation. We then calculate the correlation between distributions generated by all possible key pairs, as shown in Table 3. We see that no key pairs with fully correlated distributions are observed. There are, however, still some very high correlations, for example, 0.90 between (1,0) and (1,1) in the left table, 0.93 between (0,1) and (0,0) in the right table. This suggests that the hypothetical power consumption values corresponding to some incorrect key candidates (besides the correct one) are also highly correlated to the power traces, making it difficult to distinguish the correct key. Especially in practical scenarios where the traces are heavily affected by noise, the CPA may fail or require a very large number of traces to find the correct key (similar to the remark of Brier *et al.* [6]).

(k_0^j, k_1^j)	(0,0)	(0,1)	(1,0)	(1,1)
(0,0)	1.00	0.15	0.89	0.87
(0,1)	0.15	1.00	0.48	0.09
(1,0)	0.89	0.48	1.00	0.90
(1,1)	0.87	0.09	0.90	1.00

(k_0^j, k_1^j)	(0,0)	(0,1)	(1,0)	(1,1)
(0,0)	1.00	0.93	0.52	0.17
(0,1)	0.93	1.00	0.48	0.27
(1,0)	0.52	0.48	1.00	0.09
(1,1)	0.17	0.21	0.09	1.00

Table 3: Absolute correlations between the Hamming weight distributions of the S-box output for each key pair when $\mathtt{IV}^j = 0$ (left) and when $\mathtt{IV}^j = 1$ (right).

Our analysis shows that using the Hamming weight of the S-box output as the intermediate variable is not effective for CPA attacks. In [17], Ramezanpour *et al.* adopted this approach for their CPA attack. The authors reported that their attack failed to recover the correct key even after using more than 40K traces, but they did not provide any justification. Since we do not have access to their implementation, we cannot determine the precise cause of the failure. However, we believe that the insights from our analysis here may contribute to explaining this outcome.

[3] This is specific to the hardware implementation design where a register stores a 5-bit S-box output, with 1 bit from each of the 5 words. In other words, the register is designed to operate along the vertical dimension in Figure 2a. This design, adopted by Ramezanpour *et al.* [17], differs from the intent of the reference implementation, where a register is meant to store an entire word or part of a word, corresponding to the horizontal dimension in Figure 2a.

3.2 Fine-tuned S-box computation as selection function

This approach was introduced by Daemen and Samwel [20], who chose the output of the linear diffusion layer in their hardware implementation as the attack point, corresponding to the location of the registers. The activity of these registers at the end of each round (load/store) is assumed to leak information through power consumption. A notable contribution of their work is the adjustment applied to S-box computation before using it as the selection function. We now recall this adjustment and then analyze its impact on the success of CPA attacks.

As in [20], we only consider y_0^j, y_1^j and y_4^j in Equation 4 as their computations contain non-linear terms between the key and the nonce. Let us focus on y_0^j as an example. Its computation in Equation 4 is rewritten as follows:

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

Following Bertoni *et al.* [3], the term $k_0^j k_1^j \oplus k_0^j \mathtt{IV}^j \oplus k_1^j \oplus \mathtt{IV}^j$ can be removed because, for the fixed correct key in the device, it is independent of the nonce and contributes a constant amount to the activity that drives the targeted power consumption of the register containing y_0. Note that this removal is similar to fixing $k_1^j = 0$ (and $\mathtt{IV}^j = 0$) as discussed about the reduction for y_0^j in Table 2. The fine-tuned version of y_0^j, denoted by $\tilde{y}_0^j$, is:

$$\tilde{y}_0^j = k_0^j(n_1^j \oplus 1) \oplus n_0^j. \tag{5}$$

As the attack point is the activity of the register at the linear diffusion layer output (not at the S-box output), we take the operation of this layer into account. Recall from Equation 1 that the first 64-bit output word z_0 of this layer is computed as:

$$z_0 = y_0 \oplus (y_0 \ggg 19) \oplus (y_0 \ggg 28).$$

The computation of the j-th bit of z_0 $(0 \le j \le 63)$ thus is:

$$z_0^j = y_0^j \oplus y_0^{j+36} \oplus y_0^{j+45}. \tag{6}$$

The additions $j+36$ and $j+45$ are implicitly taken modulo 64. Applying Equation 5 to Equation 6 results in the fine-tuned version of z_0^j, denoted by $\tilde{z}_0^j$, which is used as the selection function to recover k_0 (three bits at a time):

$$\begin{aligned}\tilde{z}_0^j = &\left(k_0^j(n_1^j \oplus 1) \oplus n_0^j\right) \\ &\oplus \left(k_0^{j+36}(n_1^{j+36} \oplus 1) \oplus n_0^{j+36}\right) \\ &\oplus \left(k_0^{j+45}(n_1^{j+45} \oplus 1) \oplus n_0^{j+45}\right).\end{aligned} \tag{7}$$

Similarly, we can derive the selection functions for recovering k_0 by fine-tuning y_4^j, and for recovering k_1 by fine-tunning y_1^j. The detailed derivation steps are provided in Section A. Here, we present the fine-tuned version of z_1^j, denoted by $\tilde{z}_1^j$:

$$\begin{aligned}\tilde{z}_1^j = &\left(n_0^j(k_{01}^j \oplus 1) \oplus n_1^j\right) \\ &\oplus \left(n_0^{j+3}(k_{01}^{j+3} \oplus 1) \oplus n_1^{j+3}\right) \\ &\oplus \left(n_0^{j+25}(k_{01}^{j+25} \oplus 1) \oplus n_1^{j+25}\right),\end{aligned} \tag{8}$$

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 $\tilde{z}_0^j$ as the selection function.

Impact of fine-tuning. Let us analyze the selection function $\tilde{z}_0^j$. A similar analysis applies to $\tilde{z}_1^j$ and we present here the results for both $\tilde{z}_0^j$ and $\tilde{z}_1^j$. We begin by examining the core of $\tilde{z}_0^j$, which is $\tilde{y}_0^j$ (Equation 5). As before, we calculate the distribution of $\tilde{y}_0^j$ for all possible candidates for k_0^j (2 candidates in total) in Table 4. It can be seen that the distributions produced by the two key candidates are uncorrelated to each other.

	k_0^j	
(n_0^j, n_1^j)	0	1
(0,0)	0	1
(0,1)	0	0
(1,0)	1	0
(1,1)	1	1
Correlation	0	

	k_1^j	
(n_0^j, n_1^j)	0	1
(0,0)	0	0
(0,1)	1	1
(1,0)	1	0
(1,1)	0	1
Correlation	0	

Table 4: Distribution of $\tilde{y}_0^j$ (left) and $\tilde{y}_1^j$ (right) corresponding to every possible key candidate.

We then extend this calculation to the selection function $\tilde{z}_0^j$. Table 5 presents the correlations between distributions of all possible key pairs. Note that 3 key bits and 6 nonce bits involve in $\tilde{z}_0^j$. We thus have 8 key candidates. As we can see, the distribution associated with an arbitrary key is uncorrelated with that of any other key. This makes the correlation between the hypothetical power consumption associated with the correct key and the power traces stand out those of the incorrect keys. Figure 4 illustrates the experimental result for this analysis. It can be seen that the prominent peaks appear exclusively in the correlation trace of a single (correct) key candidate $(0, 0, 1)$. This explains the success of the attacks in [20], as opposed to the failure of the attack in [17], which relied on using the pure S-box computation as the selection function.

Impact of linear diffusion layer. Recall that Daemen and Samwel [20] choose the linear diffusion layer output as the attack point since it is where the registers

$(k_0^j, k_0^{j+36}, k_0^{j+45})$ or $(k_1^j, k_1^{j+3}, k_1^{j+25})$	(0,0,0)	(0,0,1)	(0,1,0)	(0,1,1)	(1,0,0)	(1,0,1)	(1,1,0)	(1,1,1)
(0,0,0)	1	-	-	-	-	-	-	-
(0,0,1)	-	1	-	-	-	-	-	-
(0,1,0)	-	-	1	-	-	-	-	-
(0,1,1)	-	-	-	1	-	-	-	-
(1,0,0)	-	-	-	-	1	-	-	-
(1,0,1)	-	-	-	-	-	1	-	-
(1,1,0)	-	-	-	-	-	-	1	-
(1,1,1)	-	-	-	-	-	-	-	1

Table 5: Absolute correlations of distributions associated to all possible key pairs using the selection functions $\tilde{z}_0^j$ and $\tilde{z}_1^j$. The entries with value 0 are indicated by "-" to facilitate reading.

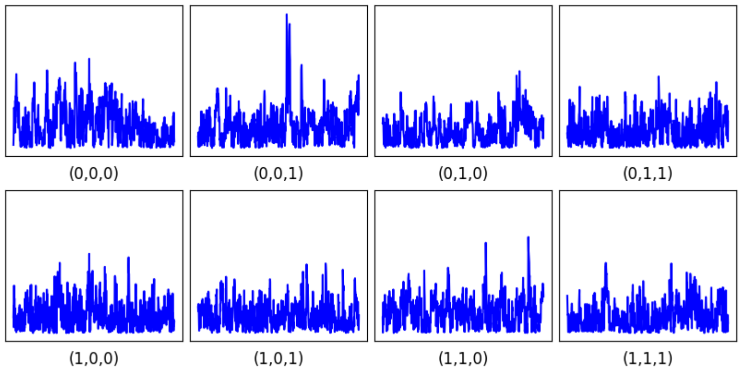


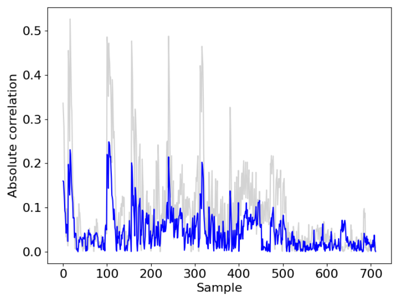
Fig. 4: Correlation traces for all key candidates when using $\tilde{z}_0^j$ as the intermediate variable. The calculations use 1000 traces recorded from the execution of the reference implementation.

locate in their hardware implementation. In software implementations, the term "register" refers to variables or memory locations used to emulate the behavior of hardware registers. Thus, a variable update after an operation can be seen as a register activity consuming power. Previously, we demonstrated that the pure S-box output $(y_0^j, \dots, y_4^j)$ produce distributions that are correlated to each other for some pairs of key candidates (Table 2). This leads to the fact that employing one of $y_0^j, \dots, y_4^j$ as the intermediate variable results in multiple key candidates ranking equally with the correct key. We then showed that the fine-tuned S-box functions $(\tilde{y}_0^j, \tilde{y}_1^j, \tilde{y}_4^j)$ yield distributions that are uncorrelated for all possible pairs of key candidates (Table 4). Now, we are interested in investigating whether $\tilde{y}_0^j, \tilde{y}_1^j, \tilde{y}_4^j$ are also good choices for the intermediate variable (in addition to $\tilde{z}_0^j, \tilde{z}_1^j, \tilde{z}_4^j$) in software implementations. In other words, we aim to determine whether accounting for the linear operations really impacts the success of CPA attacks, or is just primarily relevant for attacks targeting hardware implementations with registers at the output of the linear diffusion layer (as in [20]).

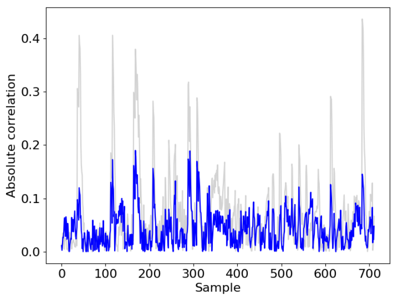
Let us consider $\tilde{y}_0^j$ from Equation 5 with two possible key candidates, $k_0^j = 0$ and $k_0^j = 1$. Below are the results of $\tilde{y}_0^j$ for each key candidate:

$$\tilde{y}_0^j = k_0^j(n_1^j \oplus 1) \oplus n_0^j = \begin{cases} n_0^j & \text{if } k_0^j = 0, \\ n_0^j \oplus n_1^j \oplus 1 & \text{if } k_0^j = 1. \end{cases}$$

We will use the visualization of peaks in correlation traces for explanation. First, when $k_0^j = 0$, $\tilde{y}_0^j = n_0^j$, meaning that the values of the intermediate variable $\tilde{y}_0^j$ are identical to the values of the nonce bit n_0^j. As a result, the values of $\tilde{y}_0^j$ become correlated with the power consumption caused by the activity of the registers containing n_0^j (in addition to that of the registers containing y_0^j). Consequently, many peaks appear in the correlation trace for $k_0^j = 0$, as the blue peaks shown in Figure 5a. To support this explanation, we determine the locations where the activity of the registers containing n_0^j cause the power consumption, as illustrated by the light gray peaks in Figure 5a.[4]



(a) $k_0^j = 0$



(b) $k_0^j = 1$

Fig. 5: Correlation traces when using $\tilde{y}_0^j$ as the intermediate variable. The calculations use 1000 traces recorded from the execution of the reference implementation.

Second, when $k_0^j = 1$, $\tilde{y}_0^j = n_0^j \oplus n_1^j \oplus 1$, meaning that the values of the intermediate variable $\tilde{y}_0^j$ are the inverse of $n_0^j \oplus n_1^j$ (correlation of -1). In the S-box computation shown in Listing 1.2 (Section C), there does exist the operation $n_0^j \oplus n_1^j$. As a result, the values of $\tilde{y}_0^j$ become correlated with the power consumption

[4] For each of the recorded nonces, we extract a byte value from the 64-bit word n_0 that contains the bit n_0^j. We then compute the correlation between the Hamming weights of those values and the power traces.

caused by this operation, leading to the appearance of many peaks appear in the correlation trace for $k_0^j = 1$, as the blue peaks shown in Figure 5b. Similarly to before, we identify the locations where the activity of the operation $n_0^j \oplus n_1^j$ causes the power consumption, as illustrated by the light gray peaks in Figure 5b, to support our argument.[5]

To sum up, high peaks appear in correlation traces for all key candidates, making the CPA prone to failure in recovering the correct key. The analyses on $\tilde{y}_4^j$ and $\tilde{y}_1^j$ yield analogous results. In contrast, peaks appear only for the correct key candidate when the linear diffusion layer is accounted for, as in Figure 4. This demonstrates that the linear diffusion layer in the selection functions of $\tilde{z}_0^j$ (Equation 7) and $\tilde{z}_1^j$ (Equation 8) plays an important role in the success of the CPA attacks, even in software implementations.

4 Second-order correlation power analysis

In the previous section, we thoroughly analyzed various approaches for choosing the intermediate variable and the selection function, along with their impact on the success of CPA attacks. Our analysis shows that $\tilde{z}_0^j$ (Equation 7) and $\tilde{z}_1^j$ (Equation 8) are effective choices for the intermediate variables to recover the first and the second halves of the key. In this section, we apply these two selection functions in order to perform a second-order CPA attack on a masked software implementation with two shares. For our experiment, we use the 32-bit ARMv6 implementation[6] submitted to the call for protected software implementations of finalists in the NIST lightweight cryptography standardization process by the Ascon team.[7]

A second-order CPA attack consists of two phases: power traces pre-processing and the standard CPA. In the first phase, samples within a trace are combined to produce a *pre-processed trace*. This combination can cause the length of each trace to increase quadratically, significantly raising the attack complexity. Therefore, we detail the pre-processing steps below to ensure that the attack remains time and memory efficient in practice. In the second phase, the standard CPA, as described in Section 2, is applied to the pre-processed traces. Since this is a conventional approach, we directly present the experimental results below.

4.1 Pre-processing power traces

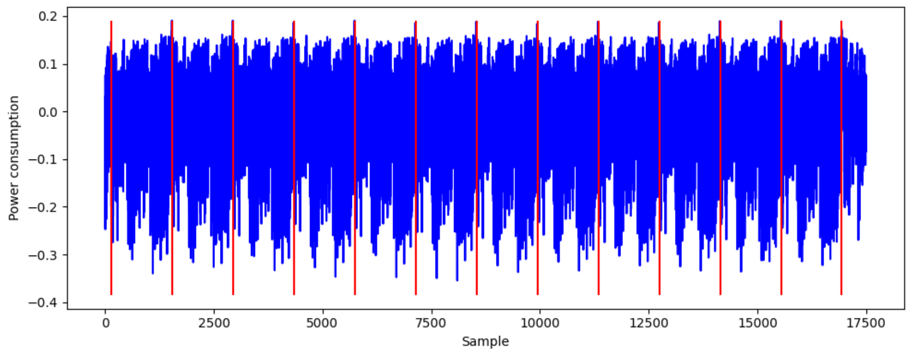
We begin by configuring an execution of the encryption with the minimum number of rounds, using an empty nonce and an empty plaintext. This setup results in 12 rounds for the initialization phase and 12 rounds for the finalization phase

[5] Similarly to before, but for $n_0 \oplus n_1$ instead of n_0.

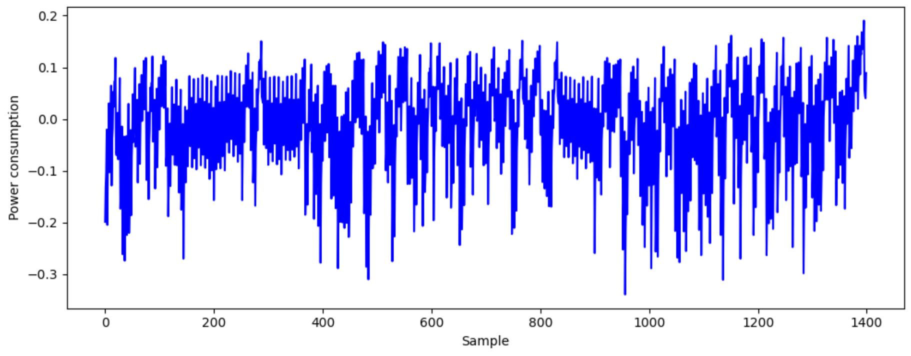
[6] `https://github.com/ascon/simpleserial-ascon/tree/main/Implementations/crypto_aead/ascon128v12/protected_bi32_armv6`

[7] `https://cryptography.gmu.edu/athena/LWC/Call_for_Protected_Software_Implementations.pdf`

(see Figure 1). During the execution of these 24 rounds, we record a power consumption trace. Variance calculations are then applied to determine the length and starting index of each round, based on the assumption that the correctly aligned frames will minimize variance (*i.e.*, 12 frames should align well when overlapped). Figure 6a shows the power consumption trace for the first 12 rounds of the initialization phase, while Figure 6b illustrate the frame corresponding to the first round, consisting of 1400 samples. We focus on the power consumption of this first round, as our attack utilizes the linear diffusion layer outputs $\tilde{z}_0^j$ and $\tilde{z}_1^j$ as intermediate variables.



(a) Power consumption of the first 12 rounds.



(b) Power consumption of the first round.

Fig. 6: Power consumption of initialization rounds.

Let $\mathbf{t} = [t_1, \ldots, t_s]$ represent a power trace containing s samples, where in our case, $s = 1400$. We can combine the samples within $\mathbf{t}$ using various methods, such as normalized product, absolute difference, or sum [16,21]. Among these, the normalized product has been shown to be the most effective when applying Pearson's correlation coefficient with the Hamming weight leakage model [21]. In this work, we adopt the normalized product for trace pre-processing. According to this method, the sample $t'_{i,j}$ in the pre-processed trace $\mathbf{t}'$ derived from two power samples t_i and t_j $(1 \leq i, j \leq s)$ in $\mathbf{t}$, is calculated as:

$$t'_{i,j} = (t_i - \bar{t})(t_j - \bar{t}),$$

where $\bar{t}$ is the mean of $\mathbf{t}$. There are a total of $s(s+1)/2 = 980700$ possible pairs (i, j) from $s = 1400$ samples. This large number will significantly increase the time and memory cost of the CPA. However, we note that the computation of the shares of the first round output occurs within a limited time span, making it unnecessary to consider all possible pairs. To address this, we introduce a parameter called the *window size*, denoted w, which estimates the maximum distance between the leakages of the two shares in the trace $\mathbf{t}$. Consequently, the number of samples in the pre-processed trace $\mathbf{t}'$ (*i.e.*, the number of pairs (i, j)) becomes:

$$W = (s - w)w + \sum_{i=1}^{w} i = w\left(t - \frac{w-1}{2}\right).$$

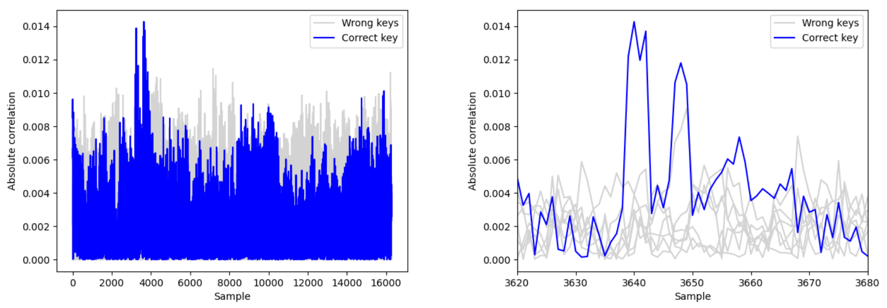
To further reduce the number of samples s in $\mathbf{t}$, we make an educated assumption that the S-box computation typically dominates the computation time in a round. Moreover, the less costly linear diffusion computation (*i.e.*, the attack point) occurs near the end of the round. Based on this insight, we focus on the last quarter of the samples in each trace. Specifically, we consider the last 350 samples (from 1050 to 1400 in Figure 6b). This reduces the value of s to 350. Additionally, we set the window size to $w = 50$. With these parameters, the number of samples in each pre-processed trace becomes $W = 16275$.

4.2 Results

To efficiently perform the CPA with a large number of traces, we implement an *incremental* second-order CPA as introduced by Bottinelli and Bos [5]. This approach enables to gradually compute Pearson's correlation coefficients with a smaller, tunable number of traces instead of processing the entire dataset at once. Recall that $\tilde{z}_0^j$ (Equation 7) and $\tilde{z}_1^j$ (Equation 8) are used to recover the two halves (k_0, k_1) of the key, three bits at a time.

Figure 7 shows the results of the second-order CPA for recovering three bits of k_0 using $\tilde{z}_0^j$ as the selection function. Several peaks corresponding to the correct key are clearly visible at around sample 3640. Figure 8 illustrates how the correlation coefficients depend on the number of traces. It can be observed that around 100K traces are required for the second-order CPA to reliably distinguish the correct key from the others.

Recall that each application of the CPA recovers three bits of k_0, indexed by $(j, j+36, j+45)$, or three bits of k_1, indexed by $(j, j+3, j+25)$, where $0 \leq j \leq 63$ (with additions implicitly modulo 64). To recover the full 128-bit key, the CPA must be applied multiple times to different tuples of key bit indexes. These tuples must collectively cover all indexes from 0 to 63 for both k_0 and k_1. Minimizing



(a) Full-length correlation traces (b) Zoomed-in view of the peak

Fig. 7: Correlation traces for all key candidates. The calculations use 160K traces.

the number of such tuples is crucial to reduce the effort required for the attack. For example, performing the CPA three times to recover the key bits of k_0 at the tuples of indexes $(0, 36, 45)$, $(19, 55, 0)$ and $(28, 0, 9)$ (corresponding to $j = 0$, $j = 19$ and $j = 28$, respectively) results in the bit at index 0 being recovered three times, which is redundant.

To address this, we investigate the minimum number of index tuples. Weissbart and Picek [24] reported that 30 CPA runs are needed to recover k_0 and 33 runs to recover k_1, totaling 63 runs. In this work, we formalize the problem as a set cover problem and solve it using a SAT solver. Instead of utilizing the MaxSAT solver, we employ a classical SAT solver with a cardinality constraint, which is more efficient when multiple solutions are optimal. Our results show that only 23 CPA runs are needed for k_0 and 24 for k_1, reducing the total to 47 runs. The Python script used for this optimization is provided in Section B. It should be noted that with a smaller cardinality constraint for k_0 and k_1, unsatisfiable problems are obtained. Consequently, the number of CPA runs reported is optimal.

We present the dependence between the success rate and the number of power traces for the full key recovery in Figure 9. Since directly measuring the success rate for the full key recovery is time-consuming, we employ an estimation approach. Specifically, we measure the success rates of recovering three key bits of k_0 and k_1 by performing the CPA 100 times with different index tuples. These success rates obtained are then raised to the power of 23 and 24, respectively, reflecting the number of CPA runs needed to recover the full 128-bit key. Multiplying these two results provide an estimate of the success rate for full key recovery. As in Figure 9, about 360K traces are sufficient to achieve 100% success for the full key recovery. To validate this estimation, we perform an actual full key recovery experiment using 360K traces. As expected, we succeeded in recovering the full 128-bit key in about 7 hours.

In our second-order CPA, the analysis phase is very efficient as we implement the incremental computation [5] to process a small number of traces at a time.

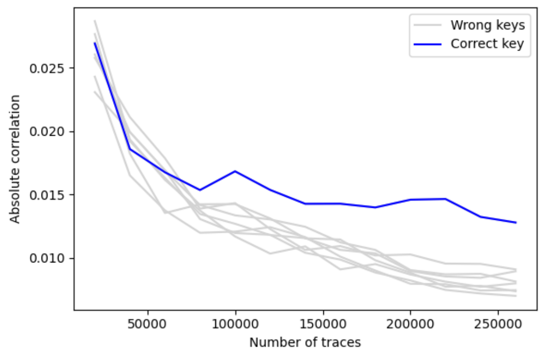


Fig. 8: Correlation for all key candidates depending on the number of traces.

For example, if we set this number to 20K, the computation consumes 0.06 GB of RAM and takes about 30 seconds. To recover three bits of the key with 100% success rate, we need to process 360K traces, which thus takes 9 minutes, while the memory cost (0.06 GB) does not change thanks to the incremental computation. The most time-consuming phase of the attack is the collection of the power traces. With a collection speed of 448 traces per minute, acquiring 360K traces requires approximately 13.4 hours to ensure a 100% success rate.

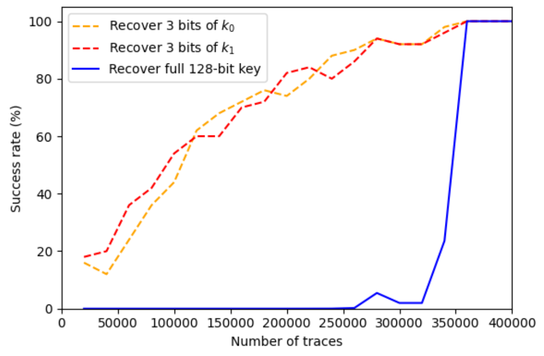


Fig. 9: Success rate of the full key recovery.

5 Conclusion and perspectives

In this paper, we investigated various aspects of the CPA attack on (protected) software implementations of Ascon. We first provided a thorough analysis of the impact of different approaches for choosing the selection function on the success of the attack. Building on these insights, we presented the first practical and successful second-order CPA attack against a masked software implementation. We demonstrated how the attack can be performed efficiently with modest time and memory resources. By validating our attach path on the 32-bit ARMv6 masked implementation provided by the Ascon team, we successfully recovered the full 128-bit key in 7 hours using 360K power traces. In conclusion, this work provides a deeper understanding of selection functions in CPA attacks and a concrete demonstration of their effectiveness against masked implementations.

This work focuses exclusively on 1-bit selection functions. A promising direction for future research is to explore how multi-bit selection functions could further reduce the number of traces required for successful attacks, potentially enhancing the efficiency of CPA attacks against protected implementations.

References

1. Advanced Encryption Standard (AES). National Institute of Standards and Technology, NIST FIPS PUB 197, U.S. Department of Commerce (Nov 2001)
2. Batina, L., Gierlichs, B., Prouff, E., Rivain, M., Standaert, F.X., Veyrat-Charvillon, N.: Mutual information analysis: a comprehensive study. Journal of Cryptology **24**(2), 269–291 (Apr 2011). https://doi.org/10.1007/s00145-010-9084-8
3. Bertoni, G., Daemen, J., Debande, N., Le, T.H., Peeters, M., Van Assche, G.: Power analysis of hardware implementations protected with secret sharing. In: 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture Workshops. pp. 9–16 (2012). https://doi.org/10.1109/MICROW.2012.12
4. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Duplexing the sponge: Single-pass authenticated encryption and other applications. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 320–337. Springer, Berlin, Heidelberg (Aug 2012). https://doi.org/10.1007/978-3-642-28496-0_19
5. Bottinelli, P., Bos, J.W.: Computational aspects of correlation power analysis. Journal of Cryptographic Engineering **7**(3), 167–181 (Sep 2017). https://doi.org/10.1007/s13389-016-0122-9
6. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Berlin, Heidelberg (Aug 2004). https://doi.org/10.1007/978-3-540-28632-5_2
7. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 398–412. Springer, Berlin, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_26
8. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski Jr., B.S., Koç, Çetin Kaya., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Berlin, Heidelberg (Aug 2003). https://doi.org/10.1007/3-540-36400-5_3
9. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2: Lightweight authenticated encryption and hashing. Journal of Cryptology **34**(3), 33 (Jul 2021). https://doi.org/10.1007/s00145-021-09398-9
10. Goubin, L., Patarin, J.: DES and differential power analysis (the "duplication" method). In: Koç, Çetin Kaya., Paar, C. (eds.) CHES'99. LNCS, vol. 1717, pp. 158–172. Springer, Berlin, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48059-5_15

11. Joye, M., Paillier, P., Schoenmakers, B.: On second-order differential power analysis. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 293–308. Springer, Berlin, Heidelberg (Aug / Sep 2005). https://doi.org/10.1007/11545262_22
12. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 388–397. Springer, Berlin, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_25
13. Luo, S., Wu, W., Li, Y., Zhang, R., Liu, Z.: An efficient soft analytical side-channel attack on ascon. Springer-Verlag, Berlin, Heidelberg (2022). https://doi.org/10.1007/978-3-031-19208-1_32, https://doi.org/10.1007/978-3-031-19208-1_32
14. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. Cryptology ePrint Archive, Report 2016/921 (2016), https://eprint.iacr.org/2016/921
15. Messerges, T.S.: Using second-order power analysis to attack DPA resistant software. In: Koç, Çetin Kaya., Paar, C. (eds.) CHES 2000. LNCS, vol. 1965, pp. 238–251. Springer, Berlin, Heidelberg (Aug 2000). https://doi.org/10.1007/3-540-44499-8_19
16. Prouff, E., Rivain, M., Bévan, R.: Statistical analysis of second order differential power analysis. Cryptology ePrint Archive, Report 2010/646 (2010), https://eprint.iacr.org/2010/646
17. Ramezanpour, K., Abdulgadir, A., Diehl, W., Kaps, J.P., , Ampadu, P.: Active and passive side-channel key recovery attacks on Ascon. NIST Lightweight Cryptography Workshop (2020), https://csrc.nist.gov/CSRC/media/Events/lightweight-cryptography-workshop-2020/documents/papers/active-passive-recovery-attacks-ascon-lwc2020.pdf, https://csrc.nist.gov/CSRC/media/Events/lightweight-cryptography-workshop-2020/documents/papers/active-passive-recovery-attacks-ascon-lwc2020.pdf
18. Rezaeezade, A., Basurto-Becerra, A., Weissbart, L., Perin, G.: One for all, all for ascon: Ensemble-based deep learning side-channel analysis. Cryptology ePrint Archive, Report 2023/1922 (2023), https://eprint.iacr.org/2023/1922
19. Roussel, N., Potin, O., Dutertre, J., Rigaud, J.: Security evaluation of a hybrid CMOS/MRAM ascon hardware implementation. In: Design, Automation & Test in Europe Conference & Exhibition, DATE 2023, Antwerp, Belgium, April 17-19, 2023. pp. 1–6. IEEE (2023). https://doi.org/10.23919/DATE56975.2023.10137126, https://doi.org/10.23919/DATE56975.2023.10137126
20. 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
21. Standaert, F.X., Veyrat-Charvillon, N., Oswald, E., Gierlichs, B., Medwed, M., Kasper, M., Mangard, S.: The world is not enough: Another look on second-order DPA. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 112–129. Springer, Berlin, Heidelberg (Dec 2010). https://doi.org/10.1007/978-3-642-17373-8_7
22. Veyrat-Charvillon, N., Gérard, B., Standaert, F.X.: Soft analytical side-channel attacks. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part I. LNCS, vol. 8873, pp. 282–296. Springer, Berlin, Heidelberg (Dec 2014). https://doi.org/10.1007/978-3-662-45611-8_15
23. Waddle, J., Wagner, D.: Towards efficient second-order power analysis. In: Joye, M., Quisquater, J.J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 1–15. Springer, Berlin, Heidelberg (Aug 2004). https://doi.org/10.1007/978-3-540-28632-5_1

24. Weissbart, L., Picek, S.: Lightweight but not easy: Side-channel analysis of the ascon authenticated cipher on a 32-bit microcontroller. Cryptology ePrint Archive, Paper 2023/1598 (2023), `https://eprint.iacr.org/2023/1598`, `https://eprint.iacr.org/2023/1598`
25. You, S.C., Kuhn, M.G., Sarkar, S., Hao, F.: Low trace-count template attacks on 32-bit implementations of ASCON AEAD. IACR TCHES **2023**(4), 344–366 (2023). `https://doi.org/10.46586/tches.v2023.i4.344-366`

A Derivation of selection functions

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

$$\begin{aligned} y_1^j &= n_0^j(k_1^j \oplus k_0^j \oplus 1) \oplus n_1^j \oplus k_1^j k_0^j \oplus k_1^j \oplus k_0^j \oplus \mathtt{IV}^j, \\ y_4^j &= n_1^j(k_0^j \oplus 1) \oplus n_0^j \oplus k_0^j \mathtt{IV}^j \oplus k_0^j. \end{aligned}$$

In y_1^j, we remove $k_1^j k_0^j \oplus k_1^j \oplus k_0^j \oplus \mathtt{IV}^j$ as they contribute a constant amount to the power consumption. For the same reason, $k_0^j \mathtt{IV}^j \oplus k_0^j$ is removed in y_4^j.

$$\begin{aligned} \tilde{y}_1^j &= n_0^j(k_{01}^j \oplus 1) \oplus n_1^j, \\ \tilde{y}_4^j &= n_1^j(k_0^j \oplus 1) \oplus n_0^j, \end{aligned}$$

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

Recall the linear operations applied on the y_1 and y_4:

$$\begin{aligned} z_1 &= y_1 \oplus (y_1 \ggg 61) \oplus (y_1 \ggg 39), \\ z_4 &= y_4 \oplus (y_4 \ggg 7) \oplus (y_4 \ggg 41). \end{aligned}$$

The j-th bit of z_1 and z_4 are thus computed as:

$$\begin{aligned} z_1^j &= y_1^j \oplus y_1^{j+3} \oplus y_1^{j+25}, \\ z_4^j &= y_4^j \oplus y_4^{j+57} \oplus y_4^{j+23}. \end{aligned}$$

We then apply the linear operations for $\tilde{y}_1^j$ and $\tilde{y}_4^j$:

$$\begin{aligned} \tilde{z}_1^j = &\left(n_0^j(k_{01}^j \oplus 1) \oplus n_1^j\right) \\ &\oplus \left(n_0^{j+3}(k_{01}^{j+3} \oplus 1) \oplus n_1^{j+3}\right) \\ &\oplus \left(n_0^{j+25}(k_{01}^{j+25} \oplus 1) \oplus n_1^{j+25}\right). \end{aligned} \tag{9}$$

$$\begin{aligned} \tilde{z}_4^j = &\left(n_1^j(k_0^j \oplus 1) \oplus n_0^j\right) \\ &\oplus \left(n_1^{j+57}(k_0^{j+57} \oplus 1) \oplus n_0^{j+57}\right) \\ &\oplus \left(n_1^{j+23}(k_0^{j+23} \oplus 1) \oplus n_0^{j+23}\right). \end{aligned} \tag{10}$$

B Tuples of indexes for key recovery

```python
import pycryptosat as cs
import sys

# Cardinality constraint system proposed by Sinz
# sSat: the CNF system used
# setId: list of the variable id of the set
# nb_sets: number of sets
# cc: cardinality constraint
# startextra: index of additional variable unused so far, can
    be a value large enough
# For more details C. Sinz, Towards an Optimal CNF Encoding
   of Boolean Cardinality Constraints.
def Cardinality_Constraints(sSat,setId,nb_sets,cc,startextra)
   :
    sSat.add_clause([-(setId[0]), startextra])
    for j in range (2,cc+1):
        sSat.add_clause([-(startextra+j-1)])
    for i in range(2,nb_sets):
        sSat.add_clause([-(setId[i-1]),
                         startextra+(cc)*(i-1)])
        sSat.add_clause([-(startextra+(cc)*(i-2)),
                         startextra+(cc)*(i-1)])
        for j in range(2,cc+1):
            sSat.add_clause([-(setId[i-1]),
                             -(startextra+cc*(i-2)+j-2),
                             startextra+(i-1)*cc+j-1])
            sSat.add_clause([-(startextra+cc*(i-2)+j-1),
                             startextra+(i-1)*cc+j-1])
        sSat.add_clause([-(setId[i-1]),
                         -(startextra+cc*(i-2)+cc-1)])
    sSat.add_clause([-(setId[nb_sets-1]),
                -(startextra+cc*(nb_sets-2)+cc-1)])
    startextra+=(nb_sets-1)*cc

if __name__ == "__main__":
    # Arguments
    # Run ‘python3 64 36 45 23‘ for k0
    # Run ‘python3 64 3 25 24‘  for k1
    n  = int(sys.argv[1]) # n = 64 indexes
    s1 = int(sys.argv[2]) # 1st shift
    s2 = int(sys.argv[3]) # 2nd shift
    ub = int(sys.argv[4]) # upper bound for number of subsets

    # Compute all subset
    list_set=[]
    for i in range(n):
        list_set.append([i,(i+s1)%n,(i+s2)%n])

    # Save the universe
    universe=list(range(n))

    # Create the solver object
    sSat=cs.Solver()

    # Add constraint
    # For each element of the universe, we should select at least one subset containing the element
    for i in universe:
        # create an empty disjuction
        clause_presence=[]
        for j in list_set:
            # if subset j contains the element i, we add the variable corresponding to the set j  to the disjunctions
            if i in j:
                clause_presence += [list_set.index(j)+1]
        # One variable in clause_presence must be set to true
        # Add disjunction to the conjunction
        sSat.add_clause(clause_presence)

    # Cardinality constraints
    Cardinality_Constraints(sSat,
                list(range(1,len(list_set)+1)),
                len(list_set),
                ub,
                len(list_set)+1)

    # solve it
    satq,solution=sSat.solve()
    # SAT or UNSAT
    print(satq)
    # if SAT print the solution
    if(satq):
        # for all variable corresponding to a subset
        for i in range(1,len(list_set)+1):
            #  if True then the subset has been selected
            if solution[i]:
                print(list_set[i-1])
            # else the subset has not been selected each element of this subset should appear in at least one other selected subset
```

Listing 1.1: Python script to find the minimum number of tuples.

C Impact of linear layer in selection function

23 tuples for k_0 $(j, j+36, j+45)$		24 tuples for k_1 $(j, j+3, j+25)$	
(1,37,46)	(29, 1,10)	(3, 6,28)	(34,37,59)
(2,38,47)	(32, 4,13)	(5, 8,30)	(39,42, 0)
(5,41,50)	(35, 7,16)	(6, 9,31)	(41,44, 2)
(8,44,53)	(39,11,20)	(7,10,32)	(43,46, 4)
(11,47,56)	(42,14,23)	(13,16,38)	(45,48, 6)
(12,48,57)	(45,17,26)	(15,18,40)	(50,53,11)
(15,51,60)	(49,21,30)	(17,20,42)	(51,54,12)
(18,54,63)	(52,24,33)	(22,25,47)	(52,55,13)
(19,55, 0)	(55,27,36)	(24,27,49)	(53,56,14)
(22,58, 3)	(59,31,40)	(26,29,51)	(58,61,19)
(25,61, 6)	(62,34,43)	(32,35,57)	(60,63,21)
(28, 0, 9)		(33,36,58)	(62, 1,23)

Table 6: Tuples of indexes for each 3-bit key recovery. The number of tuples should be minimized while the range from 0 to 63 must be covered.

```
x0 = x0 ^ x4
x4 = x4 ^ x3         // n0 ^ n1 (in the first round)
x2 = x2 ^ x1
// Start of keccak S-box
t0 = x0 ^ (~x1 & x2)
t1 = x1 ^ (~x2 & x3)
t2 = x2 ^ (~x3 & x4)
t3 = x3 ^ (~x4 & x0)
t4 = x4 ^ (~x0 & x1)
// End of keccak S-box
y0 = t0 ^ t4
y1 = t1 ^ t0
y2 = ~t2
y3 = t3 ^ t2
y4 = t4
```

Listing 1.2: C implementation of Ascon S-box.