# Lightweight but Not Easy: Side-channel Analysis of the Ascon Authenticated Cipher on a 32-bit Microcontroller

Léo Weissbart and Stjepan Picek

Radboud University, Nijmegen, The Netherlands
`firstname.lastname@ru.nl`

**Abstract.** Ascon is a recently standardized suite of symmetric cryptography for authenticated encryption and hashing algorithms designed to be lightweight. The Ascon scheme has been studied since it was introduced in 2015 for the CAESAR competition, and many efforts have been made to transform this hardware-oriented scheme to work with any embedded device architecture. Ascon is designed with side-channel resistance in mind and can also be protected with countermeasures against side-channel analysis. Up to now, the effort of side-channel analysis is mainly put on hardware implementations, with only a few studies being published on the real-world side-channel security of software implementations.
In this paper, we give a comprehensive view of the side-channel security of Ascon implemented on a 32-bit microcontroller for both the reference and a protected implementation. We show different potential leakage functions that can lead to real-world leakages and demonstrate the most potent attacks that can be obtained with the corresponding leakage functions. We present our results using correlation power analysis (CPA) and deep learning-based side-channel analysis and provide a practical estimation of the efforts needed for an attacker to recover the complete key used for authenticated encryption.
Our results show that the reference implementation is not side-channel secure since an attacker can recover the full key with 8 000 traces using CPA and around 1 000 traces with deep learning analysis. While second-order CPA cannot recover any part of the secret, deep learning side-channel analysis can recover partial keys with 800 traces on the protected implementation. Unfortunately, the model used for multi-task key recovery lacks the generalization to correctly recover all partial keys for the full key attack.

**Keywords:** Side-channel analysis · Deep learning · Multi-task learning · Lightweight cryptography · Authenticated encryption

## 1 Introduction

In the field of symmetric cryptography, the need for lightweight primitives is crucial in the development of secure, fast, and low-consumption designs that can be used for embedded devices in resource-constrained environments but also for high-bandwidth applications. In this effort, the National Institute of Standards and Technology (NIST) initiated a lightweight cryptography standardization process in 2018. The goal was set to decide on a new standard for lightweight cryptography by comparing submitted designs of block ciphers, hash functions, message authentication codes (MACs), authenticated encryption with associated data (AEAD), and pseudorandom functions (PRFs). The process ended on 7 February 2023, with the selection of the Ascon family for standardization. Ascon [DEMS21b] has been designed with side-channel resistance in mind, and the designers have provided a protected implementation of the cipher. During the standardization process, side-channel evaluation has been performed on every finalist [MBA+23], and protected implementation of Ascon in hardware and software has been proved resistant against side-channel analysis. However, this standard is still new, and the side-channel resistance of Ascon implementations has yet to gain maturity.

The Ascon team provided implementations for numerous platforms. Additionally, a masking implementation is also provided with Domain-Oriented Masking (DOM) [GMK16]. The number of shares can be configured up to five shares. The DOM implementation is available for hardware and software platforms. The choice of DOM is motivated by the flexibility and robustness of the masking scheme against higher-order attacks because of the randomization of the shares. The choice of DOM is also attractive because of the low overhead for increasing the number of shares and the reduced need for fresh randomness compared to a threshold implementation masking scheme.

The side-channel analysis report obtained from the common effort of several evaluation laboratories and researchers during the last round of the standardization process has evaluated Ascon with classical side-channel analysis methods, namely Test vectors leakage assessment (TVLA), $\chi^2$-test, and correlation power analysis. Those evaluations have been performed on hardware and software implementations and confirmed side-channel resistance even with second-order CPA, considering several millions of traces [MBA+23].

This paper focuses on showing that deep learning side-channel analysis (DLSCA) can be applied to Ascon. Moreover, we aim to enlarge the possible attack surface of Ascon in the context of profiling side-channel analysis. For this purpose, we consider different leakage models that can expose information about the key during the initialization phase of Ascon. We explore different leakage models to learn if DLSCA can be applied on an unprotected implementation of Ascon and if the same leakage can also be applied on a protected implementation.

The core challenge we try to address in this paper is to conduct a profiling side-channel analysis on Ascon with deep learning and compare the results with the known CPA attack. The contributions of this paper are as follows:

- We provide a dataset for profiling side-channel analysis on Ascon from a reference software implementation and a first-order protected implementation with DOM [1].
- We show that DLSCA can successfully recover the key using fewer traces than CPA for both unprotected and protected Ascon implementations. The CPA attack on the unprotected dataset can recover the key in 8 000 traces against 1 000 traces for DLSCA. On the protected dataset, a second-order CPA fails to recover any part of the secret key, while DLSCA can recover certain partial keys in 800 traces.
- We show different leakage models that could be used to evaluate the leakage of the Ascon S-box function to recover the key. Namely, the S-box output value and the output state's register value. While the first can be simplified to obtain better results for CPA, the second can result in better results using DLSCA in the presence of side-channel countermeasures.
- With a single task model, we can obtain the partial key under DLSCA with less than 20 traces, but the effort of the attacker to recover the full key can be underestimated with this method. For this reason, we build a multi-task model that can analyze the traces and simultaneously make decisions for every partial key separately.

The rest of the paper is organized as follows. Section 2 presents the background on Ascon and deep learning techniques considered in this paper. Section 3 provides the related work applying side-channel analysis as well as known attacks on Ascon. Section 4 discusses

[1] https://anonymous.4open.science/r/ascon_datasets-6B5F

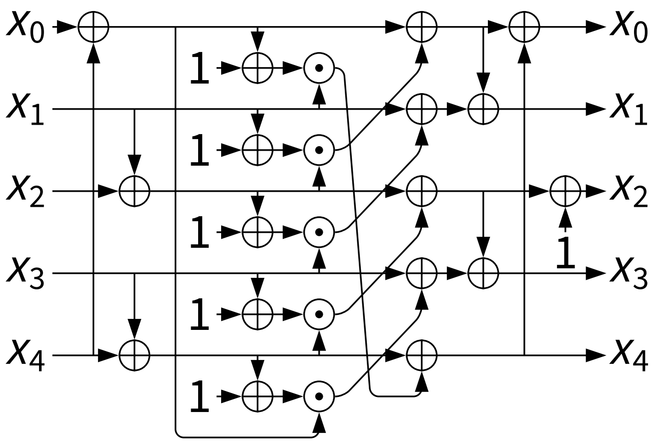


Figure 1: Ascon S-box.

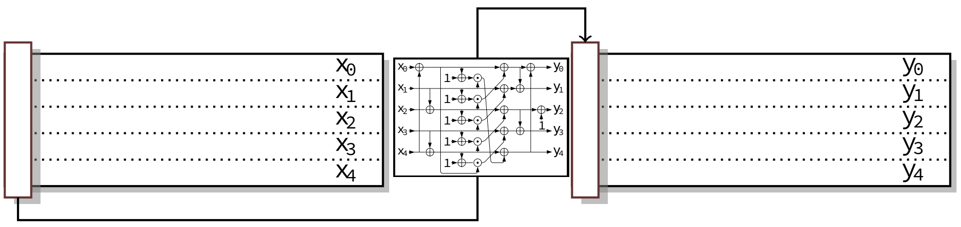


Figure 2: The Ascon S-box applied on the first column of the state.

the leakage models we considered. Section 5 presents the experimental results for DLSCA and a comparison with CPA. In Section 6, we apply the multi-task learning methodology for a deep learning attack to recover every partial key. Finally, Section 7 gives concluding remarks and elaborates on possible future work.

## 2 Background

### 2.1 Ascon

Ascon is a family of authenticated encryption and hashing algorithms standardized by NIST [DEMS21a]. It is a permutation-based block cipher with a 320-bit state divided into five words of 64-bits $(x_0, x_1, x_2, x_3, x_4)$. The permutation is applied iteratively on the state in an SPN-like fashion. The two parameters $(a, b)$ of Ascon are the number of rounds and the number of bits to be rotated in the permutation. The Ascon round transformation consists of a 1-byte addition of a round constant, a 5-bit S-box applied bit-sliced, and a linear diffusion layer.

The Ascon's S-box (Figure 1) is designed in a bit-sliced fashion that operates on the 5 bits columns of the state, 64 times in parallel, as depicted in Figure 2.

The linear diffusion layer is applied on each register of the state as described in

Eqs. (1)-(5).

$$x_0 = x_0 + (x_0 >>> 19) + (x_0 >>> 28) \tag{1}$$
$$x_1 = x_1 + (x_1 >>> 61) + (x_1 >>> 39) \tag{2}$$
$$x_2 = x_2 + (x_2 >>> 01) + (x_2 >>> 06) \tag{3}$$
$$x_3 = x_3 + (x_3 >>> 10) + (x_3 >>> 17) \tag{4}$$
$$x_4 = x_4 + (x_4 >>> 07) + (x_4 >>> 41) \tag{5}$$

## 2.2 Convolutional Neural Network

In a neural network, the smallest computing units are called neurons. Each neuron in a neural network is a mathematical function on its own. The output of several neurons acting on the same inputs can be aggregated in a so-called layer, and layers can be assembled in a graph-like structure to form a neural network. The first layer is the input layer, and the last layer is the output layer. The layers in between are called hidden layers. When the number of hidden layers is larger than one, the network is called a deep neural network. A neuron is a mathematical function that takes an input and outputs a value. The input of a neuron is the output of the previous layer, and the output of a neuron is the input of the next layer. The output of a neuron is computed as follows:

$$y = f(\sum_{i=1}^{n} w_i x_i + b), \tag{6}$$

where $x_i$ is the input of the neuron, $w_i$ is the weight of the input, $b$ is the bias of the neuron, and $f$ is the activation function.

The activation function is a non-linear function that is applied to the output of the neuron. The activation function is used to introduce non-linearity to the network. The most common activation functions are the sigmoid function, the hyperbolic tangent (Tanh) function, the rectified linear unit (ReLU) function, the scaled exponential linear unit (SeLU) function, and the Softmax function. The sigmoid function is commonly used after the output layer to transform the output of the layer into a probability distribution as the sum of the outputs sums to one, and is defined as follows[2]:

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{7}$$

The hyperbolic tangent function is defined as follows:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{8}$$

The ReLU function is defined as follows:

$$f(x) = max(0, x). \tag{9}$$

The SeLU function is defined as follows:

$$f(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leq 0, \end{cases} \tag{10}$$

where $\lambda$ and $\alpha$ are specific constants of the function.

The Softmax function is defined as follows:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}. \tag{11}$$

[2]The Softmax function is an extension on sigmoid function when there are more than two classes.

A neural network as described so far, represented as a collection of neurons, is commonly denoted as multilayer perceptron (MLP) and is a simpler form of a deep neural network.

A neural network containing at least one convolutional layer is called a convolutional neural network (CNN). A convolutional layer is a layer that applies a convolution operation on the input. The convolution operation is a mathematical operation that is defined as follows:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau, \tag{12}$$

where $f$ and $g$ are two functions. The convolution operation is applied to the input with multiple kernels. Each kernel is a matrix of weights smaller than the input size and is sliding over the input, computing the convolution operation at each position. The output of the convolution operation is called a feature map and represents a transformation of the input that holds an abstract representation of information of the input.

In deep learning, CNNs have been proven to be useful in various applications from image recognition to natural language processing [LJB+89, GMH13]. CNNs are also used in side-channel analysis to recover the key of a cryptographic implementation [MPP16, JXLW20, FYHF21, NTIY23].

## 2.3 Profiled SCA and Evaluation Metrics

Profiling SCAs map inputs (side-channel traces) to outputs (probability vector of key hypotheses). Profiling SCA consists of two phases. In the first phase, commonly called the profiling phase, the attacker builds a profiling model, which is based on the side-channel measurements coming from a clone device. In the second phase, called the attack phase, the attacker leverages the knowledge obtained from the clone device (in the form of a profiling model) and uses that knowledge to break the target device. Commonly, the output of the attack is a vector of probabilities representing the likelihood of a key guess being the correct key. Examples of profiling attacks are the template attack (TA) [CRR02] and machine (deep) learning-based attacks [HGM+11, MPP16, PWP22]. To evaluate the performance of profiling SCAs, there are several commonly used metrics with the most common ones being the success rate (SR) and the guessing entropy (GE).

**Success Rate** The success rate is the rate of successfully guessed bits of the key. When considering an attack on a traceset with a fixed key, the success rate is the rate of successfully guessed bits $k_i$ of the key $\mathbf{k}^* = (k_i^*)_{i\in\mathbb{K}}$, with $\mathbb{K}$ the length of the key in bits.

$$SR = \frac{1}{\mathbb{K}} \sum_{i=1}^{\mathbb{K}} \delta_{k_i,k_i^*} \tag{13}$$

with $\delta_{i,j}$ being the Kronecker delta function equal to one if $i = j$ and zero else.

**Guessing Entropy** The guessing entropy is the average number of guesses needed to find the correct key. The partial guessing entropy is often used to evaluate the performance of a side-channel attack on a subset of the key.

The output of a side-channel attack using $Q$ attack traces on a key $\mathbf{k}$ in a key space $|\mathbb{K}|$ is a key guessing vector $\mathbf{g} = [g_1, g_2, \ldots, g_{|\mathbb{K}|}]$ given in decreasing order of probability, where $g_1$ is the most probable guess and $g_{|K|}$ is the least probable guess. The guessing entropy is the average position of the correct key in this key guessing vector.

$$GE = E\left(rank_{k^*}(\mathbf{g})\right) \tag{14}$$

with $rank_{k^*}(\mathbf{g})$ being the position of the correct key $k^*$ in the key guessing vector $\mathbf{g}$, and $E$ the expectation operator. The partial guessing entropy is the average position of the correct key in the key guessing vector restricted to a subset of the key (e.g., one key byte).

## 3 Related Work

Side-channel analysis is commonly considered when evaluating the security of symmetric cryptographic primitives. Differential power analysis was first introduced by Kocher et al. and was originally applied to DES [KJJ99]. The authors showed that the leakage of the Hamming weight of the intermediate values of the S-box function can be used to recover the key successfully, using the difference-of-means as a statistical analysis method to compare the hypothetical power consumption values with the recorded traces. Another way to determine the relationship between data is to use the Pearson correlation coefficient. This method represents the pillar of the non-profiled SCA and is commonly referred to as Correlation Power Analysis (CPA) [BCO04, CCD00].

Deep learning side-channel analysis has been a promising technique for profiling side-channel analysis since the work of Maghrebi et al. [MPP16]. Later, Benadjila et al. [PSB+18] introduced a dataset for profiling side-channel analysis on AES protected with Boolean masking and showed that a convolutional neural network (CNN) is an efficient approach to recovering the AES key. Many other research works have also shown that deep learning can be a powerful profiling attack against AES [MPP16, PSK+18, KFYF21, RK21]. More recently, a number of works consider the multi-task paradigm in DLSCA, showcasing it can be more powerful than the single-task approach [Mag20, MO23b, MO23a, MS23].

Since Ascon was introduced in 2016 for the CAESAR competition, several works have been published on the security of Ascon. The authors of [SD17] demonstrated a CPA attack on a hardware implementation of Ascon. The authors showed that bits of the output state of the round function can lead to a key recovery. In [RAD20], the authors attacked the Ascon S-box operation of a hardware implementation that executes operations in a sequential mode. They could separate the different bitwise S-box operations to apply a horizontal attack, and they used reinforcement learning to recover the key from the leakage of the S-box. The authors also employed an autoencoder to perform dimensionality reduction given the sub-traces of every S-box operation. In [SS23], the authors used transfer learning from a well-fitting model for AES on the ASCAD dataset to improve DLSCA on an unprotected Ascon implementation for RISCV microcontroller. In general, the literature on machine learning-based SCAs on ciphers different from AES is relatively sparse, especially considering lightweight symmetric ciphers, see, e.g., [HPGM20, HPGM16, MWM21]. This indicates more work is needed to understand how to mount powerful attacks and how much of the knowledge is transferable from one target to another.

## 4 Leakage Models

The target of side-channel attacks against Ascon is the value of the key used in the initialization phase (Figure 3). The attack at this point is possible because we know the content of the state at initialization, except for the key.During the first permutation round, the state is composed of a 64-bit initialization value, the 128-bit key, and a user-defined 128-bit nonce.

The non-linear properties of the Ascon S-box with the controllable nonce enable possible leakage that can be exploited with SCA. One intuitive leakage model that can be applied is the S-box output. The Ascon S-box is a 5-bit S-box that is applied to the columns of the state. It is possible to consider this leakage differently depending on the cipher implementation.

The output state registers $x_0, .., x_5$ from Figure 1 can be rewritten in the algebraic

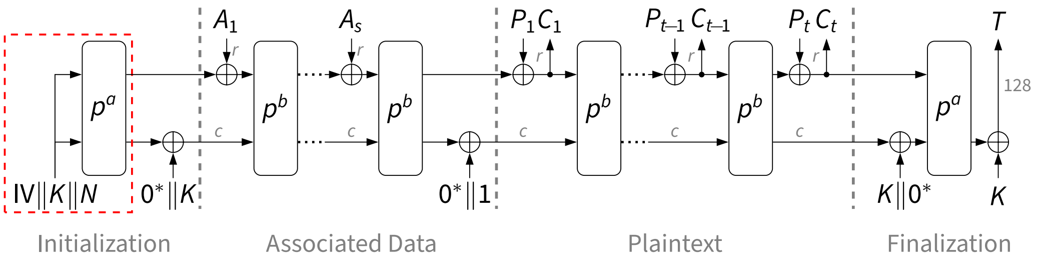


Figure 3: AEAD Encrypt.

normal form (ANF) as follows:

$$y_0 = x_1(x_4 + x_2 + x_0 + 1) + x_3 + x_2 + x_0 \tag{15}$$
$$y_1 = (x_3 + 1)(x_2 + x_1) + x_2x_1 + x_4 + x_0 \tag{16}$$
$$y_2 = x_4(x_3 + 1) + x_2 + x_1 + 1 \tag{17}$$
$$y_3 = (x_0 + 1)(x_4 + x_3) + x_2 + x_1 + x_0 \tag{18}$$
$$y_4 = x_1(x_4 + x_0 + 1) + x_3 + x_4. \tag{19}$$

## 4.1 Leakage Models for Differential Attacks

DPA attacks use datasets of many power traces from a cryptographic device operating with the same key to exploit the data dependency of the power consumption. As a consequence, the activity of the computation that has a constant contribution to the power consumption will be equal for each trace collected and will cancel out in the analysis. Thus, all the variables contributing with a constant amount to the activity can be removed from the previous notation, i.e., the terms with $x_0, x_1, x_2$, or combinations of those.

$$y_0 = x_4x_1 + x_3 \tag{20}$$
$$y_1 = x_3(x_2 + x_1 + 1) + x_4 \tag{21}$$
$$y_2 = x_4(x_3 + 1) + 1 \tag{22}$$
$$y_3 = (x_4 + x_3)(x_0 + 1) \tag{23}$$
$$y_4 = x_4(x_1 + 1) + x_3 \tag{24}$$

In Eqs. (22) and (23), $y_2$ and $y_3$ do not involve computation on the bits of the key, and thus cannot be used as leakage functions. However, in Eqs. (20) and (24), it can be noticed that both $y_0$ and $y_4$ have a relation with a bit of state $x_1$, and can be used interchangeably to recover the first half of the key. By attacking the register $y_1$ in Eq. (21), the leakage from the value of $x_1 + x_2$ can be learned. This term can be used in conjunction with the information recovered from the previous leakage on $x_1$ to get the second half of the key related to $x_2$, as also pointed out in [SD17].

The content of register $x_1$ or $x_1 + x_2$ is the secret denoted $k$, and $x_3$ and $x_4$ are denoted $m$ and $m'$, respectively.

$$y_0 = km' + m$$
$$y_1 = m(k + 1) + m'$$
$$y_4 = m'(k + 1) + m.$$

The application of the linear diffusion layer on the output of the S-box function permits

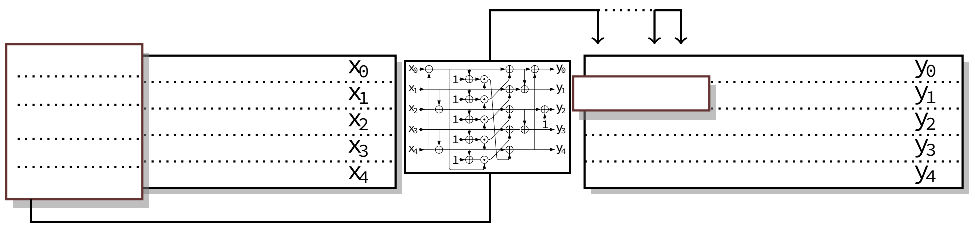


Figure 4: Computation of the leakage value of a given output register.

to obtain the following expressions:

$$S0_i(M,K^*) = k_0^* m_i' + m_i + k_1^* m_{i+45}' + m_{i+45} + k_2^* m_{i+36}' + m_{i+36} \tag{25}$$

$$S1_i(M,K^*) = m_i(k_0^*+1) + m_i' + m_{i+3}(k_1^*+1) + m_{i+3}' + m_i' + m_{i+25}(k_2^*+1) + m_{i+25}' \tag{26}$$

$$S4_i(M,K^*) = m_i'(k_0^*+1) + m_i + m_{i+57}'(k_1^*+1) + m_{i+57} + m_{i+23}'(k_2^*+1) + m_{i+23}. \tag{27}$$

Eqs. (25), (26), and (27) show the leakage functions that can be used to recover the key when targeting the output of the round function.

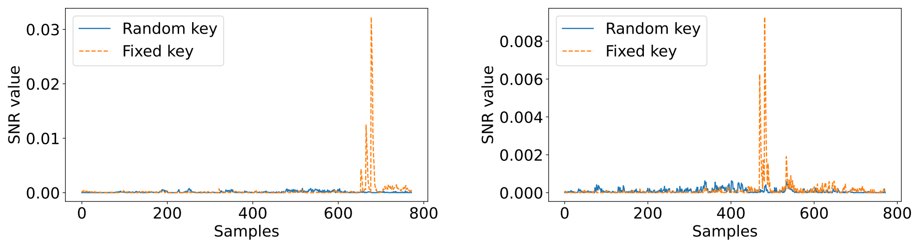
This leakage function directly aims to correlate the power consumption with the S-box output value. Since the S-box operation works on a column of the state, the storage of the output should be uncorrelated with the operation and is understandable from the bit-sliced nature of the operation. The leakage is based on the value of only one bit of the output state. Thus, using the Identity or Hamming weight power model makes no difference for a software implementation. The Hamming distance power model can be used when targeting a hardware implementation.

## 4.2 Leakage Models that Apply Better for Profiled Attacks

For profiled attacks, the profiling traceset is composed of traces collected from encryption with random keys, and the attack traceset is collected with a fixed key for the attack phase. From Eq. (19), it is possible to exploit the leakage that only depends on the input register $x_1$, which contains the first half of the key. It is then possible to obtain the second half of the key from the leakage of any other register because $y_0, y_1, y_2$ and $y_3$ depend on $x_2$. In this paper, we focus on the leakage of $y_2$ (see Eq. (17)).

While the Ascon S-box is bit-sliced, the output leakage can be stronger for bytes of the output state when considering a software implementation because of the architecture of the microprocessor and leakage during the storing operation. Alternatively, slices of the state can be considered to observe the concatenated leakage for a single byte slice, as shown in Figure 4.

When considering this leakage function, the goal is to observe a correlation between the power consumption and the storage of the output state. Because the architecture of the studied microcontroller is 32-bit, the stored output is large, and it can be difficult to get a correlation based on the full-length variable. With this leakage function, we aim to obtain partial information of the value stored, and the size of the variable can be determined smaller than 32-bit. The partial correlation on a large word can be used to obtain information about the value stored in a register, as shown in [THM+07]. Repeating this attack on successive parts of the register makes it possible to recover the secret value using fewer traces and computing power than when using the full register correlation.



(a) Round function output register $y_4$. (b) Round function output register $y_1$.

Figure 5: SNR for unprotected implementation round function leakage model.

## 5 Experimental Result

### 5.1 Implementation

In this paper, we consider a software implementation of Ascon. The C implementation from the Ascon team can be found on their GitHub repository [Tea]. They provide implementations for every Ascon mode and several platforms. In this work, only the AEAD implementation of Ascon-128 v1.2 optimized for ARMv7m microcontrollers is considered. The Ascon-128 v1.2 is the recommended implementation when using Ascon for AEAD for a key of 128-bit. Instead of using the c-reference implementation, we use the 32-bit optimized implementation they provide with our target device architecture to ensure the closest results to a real scenario.

Traces are collected with a ChipWhisperer Lite board, an 8-bit precision oscilloscope, coupled with the STM32F4 target.[3] The target microcontroller is a 32-bit platform running at a default clock frequency of 7.37MHz. Traces are collected in a manner to only contain power samples during the first round of the initialization permutation for both reference and protected implementations.

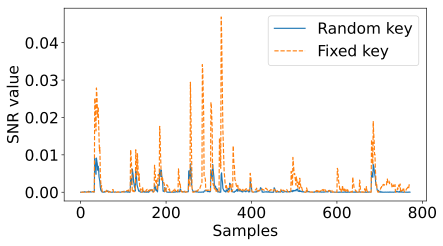
The unprotected implementation dataset contains 60 000 traces with 772 samples each. In this dataset, 50 000 traces are collected with random keys, and 10 000 traces with a fixed key. The protected implementation dataset contains 560 000 traces with 1 408 samples each. The implementation uses bit-interleaved domain-oriented masking with two shares. In this dataset, 500 000 traces are collected with random keys, and 60 000 traces with a fixed key.

### 5.2 Signal-to-Noise (SNR) for Leakage Models

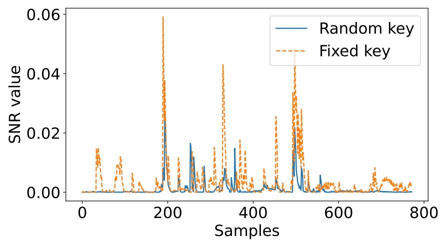
Figure 5 depicts the SNR of the round function leakage model with fixed key and random key datasets. We can observe that for both registers' leakage, only the dataset with a fixed key shows an important leakage. This observation supports the leakage established in Section 4 for non-profiled leakage models. We can also observe that leakage from register $y_4$ is stronger than the leakage from register $y_1$ by a factor of almost 4. This difference of leakage between $y_1$ and $y_4$ can result from the difference of the registers used inside the ARM microcontroller. It is also interesting to notice the leakage position for the two registers. While the leakage of the register $y_4$ is around sample 700, the register $y_1$ leaks mostly around sample 500, confirming that the two registers are treated in separate instructions in the considered implementation.

Figure 6 assembles the SNRs for the register S-box output leakage models given different numbers of input bits. We can also observe that the leakage patterns remain the same for

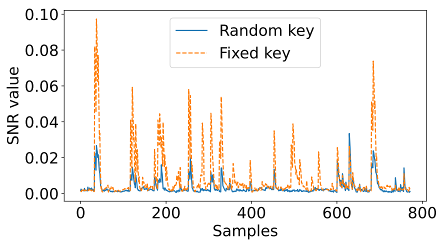
[3] https://www.newae.com/products/NAE-CWLITE-ARM



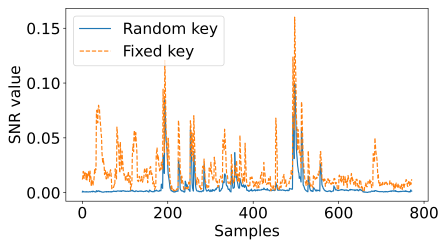
(a) 1-bit S-box output register $y_4$.



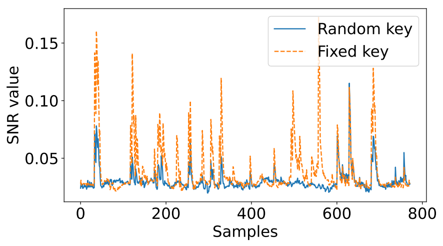
(b) 1-bit S-box output register $y_2$.



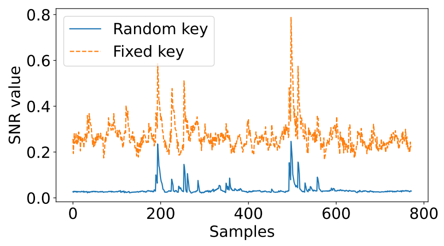
(c) 4-bit S-box output register $y_4$.



(d) 4-bit S-box output register $y_2$.



(e) 8-bit S-box output register $y_4$.



(f) 8-bit S-box output register $y_2$.

Figure 6: SNR for unprotected implementation S-box leakage models.

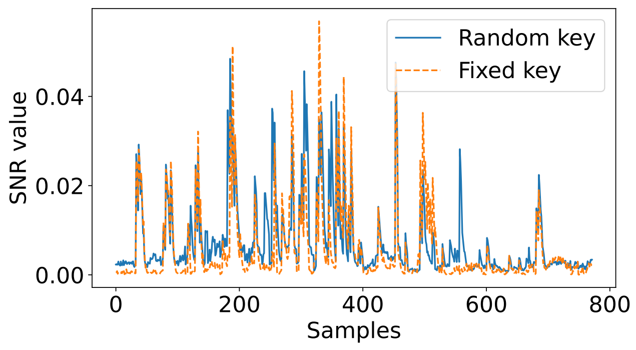


Figure 7: SNR for unprotected implementation S-box leakage on output state column.

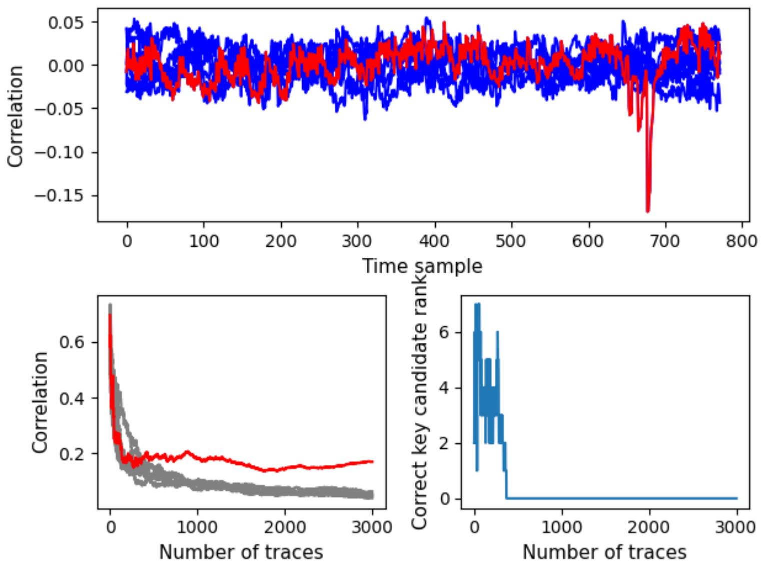


Figure 8: Correlation power analysis targeting the first bit of the state output.

all SNRs when considering more bits in the leakage and confirm the leakage model chosen in Section 4. From all SNRs, it can be noticed that the traceset with fixed key shows higher leakages than for random keys. While the SNRs suggest a correlation between these intermediate values and power consumption, it is not possible to argue whether the leakage can be exploited by a profiling attack. It can also be noticed that the more bits are considered, the stronger the SNR. Thus, it is easier only to consider the 8-bit output register model later in the experiments, as it is the most promising leakage among the three displayed. This choice is also justified because the implementation is byte-oriented. Thus, we can assume better leakage for bytes.

Finally, the SNR obtained from the S-box output column shows higher overall leakage across all samples, as shown in Figure 7. It can be noticed that leakages from a fixed key dataset and random keys dataset overlap the most among all previously displayed SNRs. This result could indicate that this leakage model can be the most present one from those considered and can be used for the profiling attack because the samples involved in this leakage are the same for the random and fixed key traces.

## 5.3 Correlation Power Analysis on Unprotected Implementation

For CPA, we use the output of the round function as the leakage function. This leakage was used in previous works [SD17, RAD20, SS23], and is a baseline for non-profiled attacks.

In Figure 8, we see that CPA can successfully recover one partial key (i.e., 3 bits) with less than 1 000 traces. In the top figure, we display the correlation of all trace samples for every key candidate, with the correct key in red. We can see that the correct key candidate has a significantly higher correlation around sample 690. In the bottom left figure, we display the highest correlation value for all samples against the number of evaluated traces for the CPA. We can observe that the correlation for the correct key candidate becomes higher than other key candidates after 400 traces and stays higher the more traces are evaluated. The bottom right figure shows the rank of the correct key against the number of evaluated traces, and it shows more clearly the convergence of the correct key candidate to zero within 400 traces. To confirm that this attack can be applied to recover every partial key, we describe how to repeat the previous partial key recovery efficiently.

Table 1: Best set of column indices to recover the key from output state CPA in registers $y_4$ and $y_1$.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $y_4$ | 0 | 1 | 2 | 7 | 8 | 9 | 14 | 15 | 16 | 20 | 21 | 22 | 28 | 29 | 30 |
| | 35 | 36 | 37 | 42 | 43 | 44 | 45 | 49 | 50 | 51 | 52 | 56 | 57 | 58 | 63 |
| $y_1$ | 0 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 12 | 13 | 15 | 16 | 17 |
| | 18 | 20 | 21 | 22 | 24 | 26 | 34 | 37 | 42 | 46 | 50 | 51 | 52 | 53 | 54 |
| | 55 | 56 | 58 | | | | | | | | | | | | |

The attack described above targets a sequence of three bits to get halves of the key given the two registers $y_4$ and $y_1$. When considering the register $y_4$, the sequence of bits of the key $[i, i+57, i+23]$ is related to the index of column $i$ of the output state. With one CPA, the attacker can obtain one sequence of three bits of the first half of the key. One can do as many CPAs as there are bits to guess and only consider the first guessed bit, disregarding the rest of the found value. However, the least number of CPAs to perform to guess each bit of the first half of the key is defined by the minimal set of sequences with the least overlapping bits.

Given the distribution of all partial keys in register $y_4$, the best search can be obtained by walking through the indices in Table 1. This set of indices requires performing 30 CPA attacks to recover the first half of the key. We can do a similar walk with the indices for register $y_1$ with a total of 33 CPA attacks to recover the second half of the key.

The indices are obtained with the following methodology. First, we build a list of the bit sequences (i.e., the partial keys) and their index in the binary representation of the key for all columns in the state. Then, we remove every second sequence that has an overlapping bit index. From this reduced list, we iteratively append the partial keys for which the sequence contain one bit of the key is missing from all the sequences in the reduced list. The final list contains the minimum number of key indices for which an attacker should repeat CPA attacks to recover the 128 bits of the key.

To recover the complete 128-bit key, we must repeat the same attack 63 times on several bits of the output state.

The result of the full key recovery is described with the success rate metric. With this metric, it is possible to evaluate the minimum number of traces needed to recover every partial key, and a success rate of one translates to a successful attack. The success rate of the full key recovery is shown in Figure 9. With the fixed key dataset, it is possible to correctly guess all the bits of the 128-bit key using around 8 000 traces.

## 5.4 Correlation Power Analysis on Protected Implementation

We now consider the protected implementation against second-order CPA. The leakage model remains the same as for the unprotected implementation. We apply window averaging with a window size of 8 to reduce the number of samples to analyze, and then combine samples with normalized multiplication for the second-order CPA attack. The number of samples in the raw traces is 1 408. By applying the pre-processing for second-order CPA, we reduce the number of samples of the traces to 15 400 samples. The window averaging is essential to reduce the number of samples to a reasonable number for the second-order CPA, as the number of samples is squared from the raw traces. The combination method uses normalized multiplication, as it enables efficient combining leakages of shares contained in two separate samples with normalization.

The output of the CPA targeting the first bit of the S-box output is shown in Figure 10. We can see that even after 60 000 traces, the correlation of the first bit of the S-box output is not significant enough to recover the key. The upper figure shows the correlation of every key candidate for all time samples. From this figure, we can see that the correct key candidate (in red) does not have a correlation value higher than any other key candidate

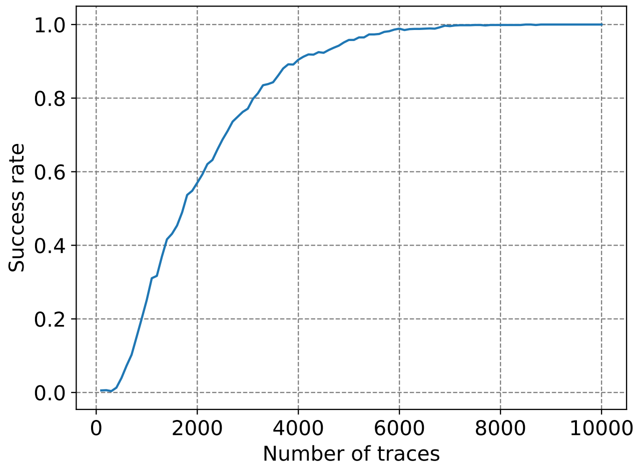


Figure 9: Success rate of the CPA.

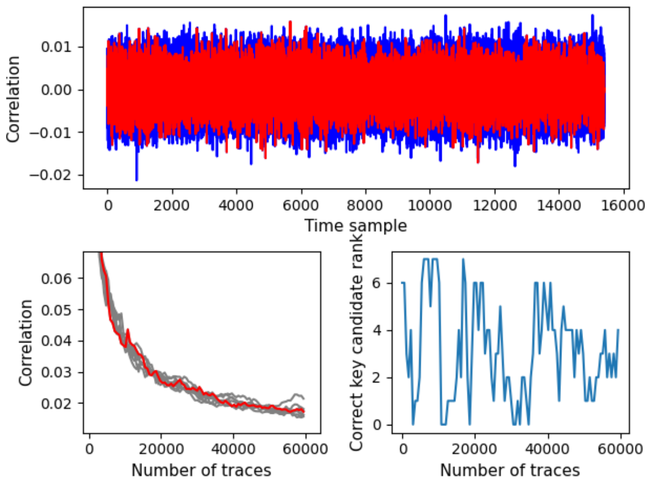


Figure 10: CPA targeting the first bit of the state output for the protected implementation.

for any sample. The bottom left figure shows the highest correlation value from all key candidates against the number of evaluated traces. We can observe from this figure that the correlation of the correct key candidate does not increase with more traces considered up to 60 000 traces. If the value to correlate would show a potential to end with a successful attack, the correlation value for the correct key candidate (in red) should start to increase, but instead, the correlation value stays indistinguishable from the other key candidates. The bottom right figure shows the rank of the correct key candidate among all possibilities for all the steps of evaluated traces. The rank never reaches zero and does not seem to converge, either. The results clearly show that the attack is unsuccessful for the considered number of traces.

Because the partial key attack on the protected implementation is not successful, we do not consider the success rate of the full key recovery.

## 5.5 Deep Learning Attack

For a deep learning attack to be successful, a well-fitting model should be trained. Finding such a model can be challenging, depending on the leakage model and dataset considered. To find such a model, the state-of-the-art Bayesian optimization (BO) methods for hyperparameter tuning [Moc77, RW06, Ngu19, WNS21] are commonly considered to be the best approach for sequential model-based global optimization. Note that such hyperparameter tuning approaches are already used in DLSCA and give good results, see, e.g., [WPP20, RWPP21]. In this paper, we adopted the Tree-structured Parzen Estimator (TPE) approach from [BBBK11]. This approach is based on the Gaussian process for tree-structured configuration hyperparameter spaces and is well suited for a high dimensional model like CNNs with the number of layers as a hyperparameter, where the evaluation of the surrogate function is cheap. The principle of BO is to minimize an objective function using a surrogate function, a probabilistic model of the score obtained with the objective function given a set of hyperparameters. This method helps to efficiently search the hyperparameter search space by deciding the next most promising step toward the best set of hyperparameters based on the results of previously evaluated sets. This method is also known to obtain better results compared to the grid search and random search methods in fewer evaluations [TEM$^+$21]. The BO search is faster to converge toward a well-fitting model and can be used to make a better decision on when to stop the search for a better-performing model. The TPE approach adds several parameters in the algorithm to scale the exploration of the search space, taking into account the tree-structure base of a deep neural network with a high number of hyperparameters. This algorithm can also estimate the expected improvement direction to explore in the search space.

To define the search space, we first design a hyper architecture (i.e., hypermodel), which represents a guideline on which we can define the directions for our models to grow.

The Bayesian optimization method is applied for network architecture search with guessing entropy in the validation set as a surrogate function. The expected behavior of this search method is to explore the hyperparameter search space to maximize the architecture that could lead to a model with the fastest convergence to an attack with a guessing entropy of zero.

We set a few rules to shape the network architecture of a CNN to reduce the hyperparameter search space and match known well-performing designs for the analysis of 1-dimensional signals. The main principle is to use stacked convolutional layers followed by several fully-connected layers. The number of channels of the convolution layers starts from a small number and increases by a factor of 8 for every new convolution block. The architecture of the CNN hypermodel is described in Figure 11. First, the input data goes through a batch normalization layer, followed by $n\_conv\_blocks$ convolutional blocks constituted by one convolution layer, an activation function layer, a batch normalization layer present every two convolutional blocks, and an average pooling layer. After the convolutional blocks, there is a flatten layer to reshape the data and, finally, $n\_fc\_layers$ of linear and activation layers. The last linear layer outputs the decision of the neural network.

The training process follows the same procedure for all models. The training is done with a batch size of 128, and the 'Adam' optimizer [KB15]. The loss function is the cross-entropy loss, and the number of epochs is set to 200. These hyperparameters are selected manually to reduce the workload of the BO and we note we achieve good performance with such a model. During every evaluation step of the model, the validation set is used to compute guessing entropy, together with accuracy and loss. These three metrics are tracked during the training process to assess the model's performance. In a particular case when the number of convolutional blocks is zero, the network consists only of fully connected layers, and we call it a multilayer perceptron (MLP).

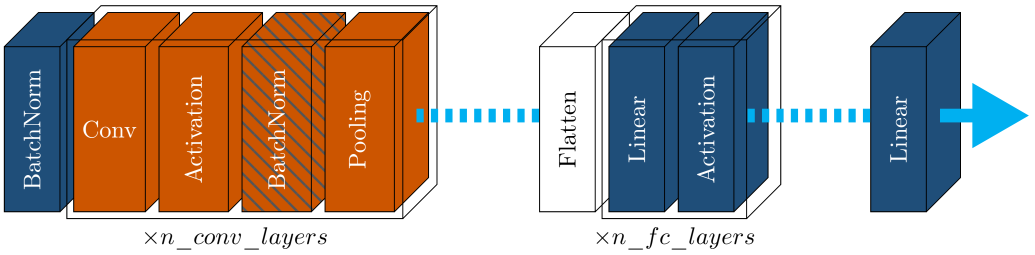


Figure 11: Hypermodel of a Convolutional Neural Network.

The hyperparameter search space is described according to the network architecture rules in Table 2.

Table 2: Hyperparameter space explored in the optimization process

| **Hyperparameter** | **Range** | **Step** |
|---|---|---|
| learning_rate | $[1e-5, \ldots, 1e-3]$ | Log |
| activation_function | [relu, selu, tanh] | None |
| n_conv_layers | $[0, \ldots, 3]$ | 1 |
| kernel_size | $[3, \ldots, 80]$ (for each conv layer) | 1 |
| n_fc_layers | $[1, \ldots, 5]$ | 1 |
| size_fc_layers | $[10, \ldots, 500]$ (for each fc layer) | 10 |

We train models on our unprotected Ascon dataset for the S-box output leakage model, as discussed in Section 4, for which the leakage function should be more adapted to the use of a random key traceset during the training phase.From the model search, the best-found model has two convolution blocks with a kernel size of 16 and 11, respectively, and two fully connected layers of 100 and 50 neurons, respectively. The number of epochs to reach the fastest guessing entropy of zero is 50 epochs with a learning rate of $1e-5$, and the attack using this model reaches a guessing entropy of zero after 20 attack traces. In the results shown in Figure 12, we can see the guessing entropy with an increasing number of traces on top and the accuracy and loss of the model during training for the training and validation sets at the bottom. While the accuracy and loss are sometimes misleading to understand the performance of a model for side-channel analysis [PPM+23], we can observe that the training loss does decrease together with the validation loss, indicating a generalization of the model on validation data.

The same training is applied to the first-order masking Ascon dataset using the same S-box output leakage model. The best model found has two convolution blocks with kernel sizes 27 and 18, and five fully connected layers of sizes 110, 230, 20, 140, and 500. The number of epochs to reach the fastest guessing entropy of zero is 20 with a learning rate of $1e-5$, and it reaches a guessing entropy of zero after 15 traces. The results of the model are shown in Figure 13.

When using the 8-bit register S-box output leakage model, we can find a good model for the unprotected dataset. With this leakage model, the partial key that is targeted is 8-bit long, thus the guessing entropy we obtain ranges in $2^8$. The convergence to GE zero is reached after 200 traces. In Figure 14, we show the results of the best-found model. The model is composed of two convolution blocks with kernel sizes 30 and 5, and four fully connected layers of sizes 500, 470, 10, and 480. The number of epochs to reach the fastest guessing entropy of zero is 160, with a learning rate of $1e-5$. The accuracy and loss of the model indicate that the model learns the leakage and stabilizes at the latest stage.

However, the same leakage model cannot lead to a satisfying model when training with the protected dataset. In Figure 15, it can be seen that the attack does not lead to a

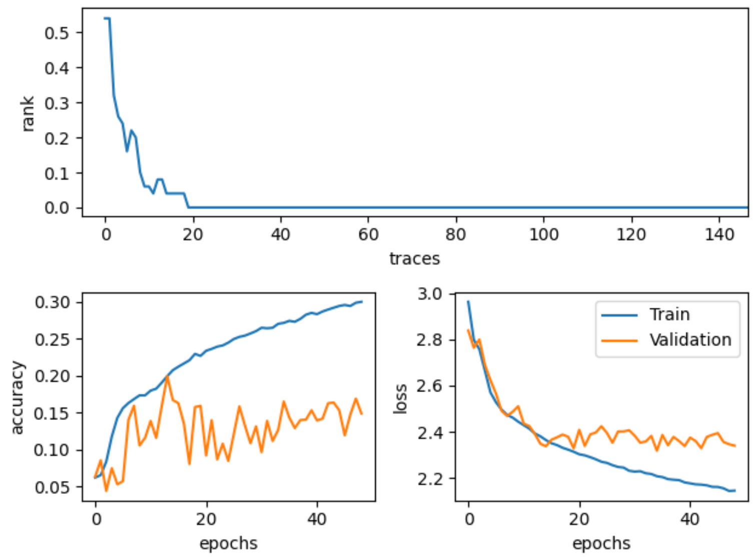


Figure 12: Results with an MLP model on S-box output leakage.

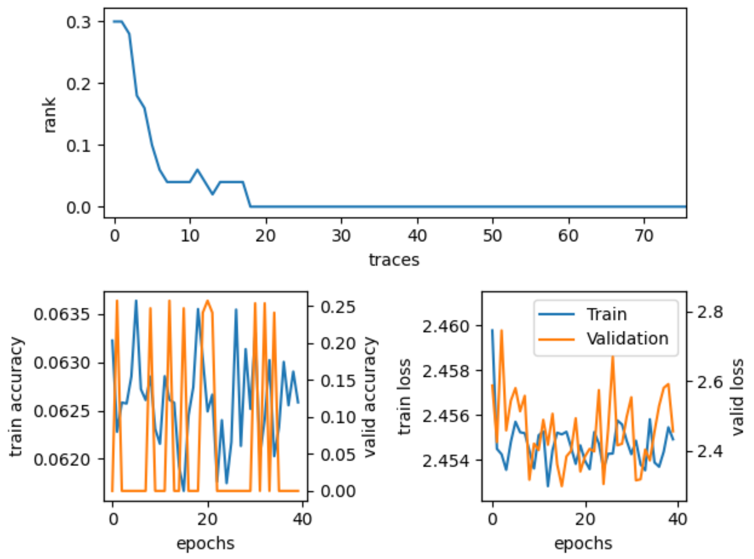


Figure 13: Results of the best-found CNN model on first-order masking S-box output leakage.

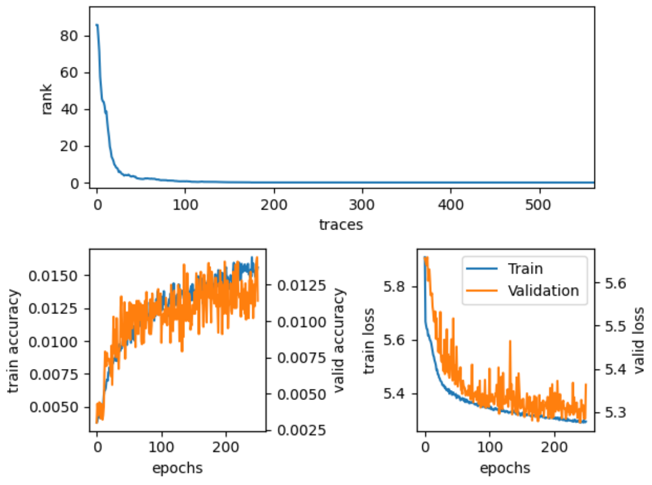


Figure 14: Results of the best-found CNN model on unprotected 8-bit S-box register output leakage model.

model capable of reducing the guessing entropy of the key. The rank value for the correct key-byte guess stays around the value 128, attesting to a random output prediction from the model. The accuracy and loss of the model show that the model does not learn from leakage at all, as the values remain stable from the first epochs until the last trained epoch. It can be concluded that the masking scheme is effective in protecting this specific leakage, even for DLSCA.

Still, the best-found model for the S-box output leakage model on the unprotected dataset is better than the CPA. The CPA can recover the key after 800 traces, while the best-found model can recover the key after 20 traces when exploiting leakage of the S-box output state. On the protected implementation, the best-found model for the S-box output leakage model can do partial key recovery with 20 traces, while CPA cannot recover the key, even after 60 000 traces. Our proposed DLSCA is better than CPA for the unprotected dataset, as it can recover the partial keys with 40 times fewer traces and can also do partial key recovery on the protected dataset with the same effort, while CPA is unsuccessful. Still, there the question remains whether this result can be improved.

## 6 Multi-task Results

The previous models all considered only a partial key. It is possible to build a model with multiple outputs, where each would estimate a different partial key and thus obtain the full key of the attacked dataset with a single model evaluation. This problem is called multi-task learning. The idea resides in the fact that a CNN can extract features from the learning set and use the different feature maps independently to output probabilities for different tasks with separated MLPs. Some works have successfully applied multi-task models to the well-known masked AES dataset like ASCAD [MO23b, MO23a, MS23] as discussed in Section 3.

To construct a multi-task model, we follow the same architecture as in Figure 11, but the output of the flatten layer is connected to multiple independent fully connected models in parallel. Each fully connected model will output the probability for a partial key (as in the previous section) and form an oracle for the full key guess. The training process of a multi-task model is similar to a single-task model. For each output of the fully connected models, we construct labels corresponding to traces in the dataset for the given partial

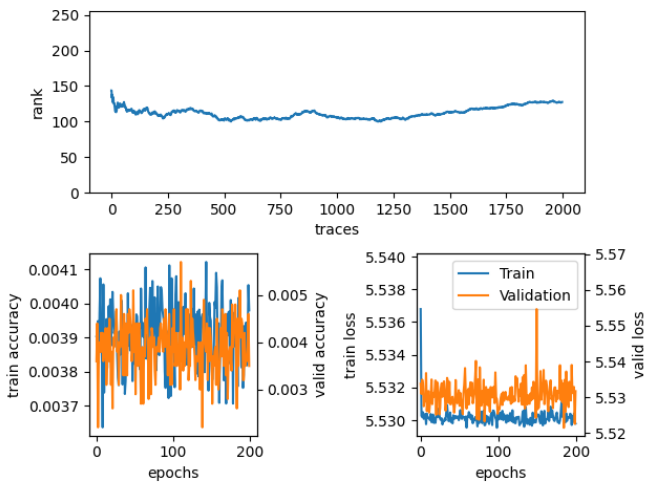


Figure 15: Results of the best-found CNN model on first-order masking 8-bit S-box register output leakage model.

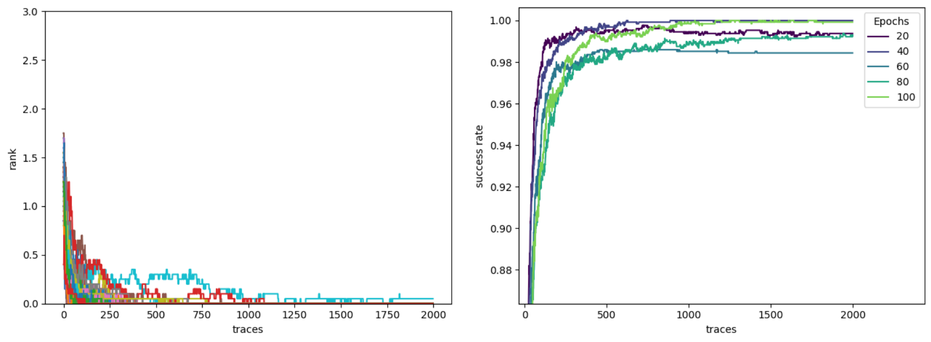
key, and the loss function is computed as the sum of the categorical cross-entropy of every branch. The backward loop during the training phase will update the weights of each fully connected model and the convolution layers to fit the leakage function for every output simultaneously.

In Figure 16a, we can see the guessing entropy of individual partial keys from the multi-task model trained on the unprotected dataset. Each line in the figure represents the guessing entropy of a partial key. We can observe that all partial keys converge to a guessing entropy of zero but at different speeds. Note that to obtain a successful multi-task attack, all partial keys should reach a guessing entropy of zero.

In Figure 16b, we present the success rate of the attack. A success rate of one is reached when the model correctly evaluates every partial key correctly for a given test traceset. We can observe that the success rate reaches values above 0.98 after only 20 epochs, and the more epochs, the closer to one goes the success rate. For the best number of epochs, the success rate reaches one after an average of 1 000 traces. Compared to the results obtained from the CPA attack on the same dataset, the multi-task model can obtain the key with $8\times$ fewer traces.

For the protected dataset, we use a multi-task model with the same hyperparameters obtained for the best single-task model. In Figure 17, we see the guessing entropy of every partial key of the multi-task model. While some partial keys converge to a guessing entropy of zero, others converge to a fixed position with consistent errors. The errors stem from the prediction of the model that does not output random ranking for every trace (as a poorly trained model would) but ranks the correct key candidate at a fixed position for every trace. This position seems to be different for all tasks and ranges between all values of the ranking vector. The value of the error for different partial keys is evenly distributed and leads to a random success rate when aggregating all partial keys results. The multi-task model trained on the protected dataset can recover some partial keys, but cannot generalize the knowledge for all partial keys.

Compared to DLSCA on AES, the number of output classes is reduced due to the fact that our S-box output leakage function is a value between 0 and $2^3$, instead of $2^8$ as for the AES S-box. The ranking of partial keys in the guessing vector makes a bigger difference when the difference between the probabilities is smaller (i.e., when the prediction is difficult).



(a) Guessing entropy of the multi-task CNN model for every partial key at the end of model training.

(b) Success rate of the full key recovery using the multi-task model for increasing number of epochs.

Figure 16: Multi-task results on unprotected dataset.

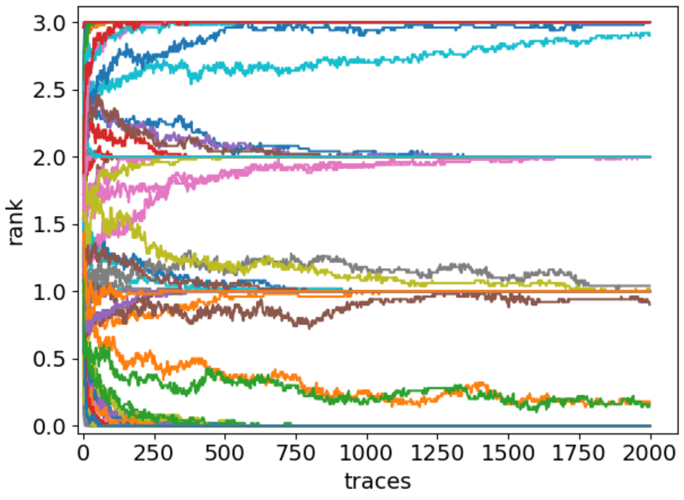


Figure 17: Guessing entropy of the multi-task CNN model for every partial key on the masked implementation at the end of model training.

## 7 Conclusions and Future Work

In this paper, we have evaluated the side-channel resistance of Ascon implementations against CPA and DLSCA. We have shown two different leakages that can be exploited: the S-box output and the register of the state after the S-box operation. Our results show it is possible to obtain successful deep learning attacks for both leakage models on the unprotected Ascon dataset. The best-found model for the S-box output leakage model can recover the key after 20 traces, and the best-found model for the register leakage model can recover the key after 200 traces. Both leakage models are better than the results obtained from the CPA attack. On the protected implementation of Ascon, our results show that the masking scheme is effective for the register output leakage model, as we cannot obtain a fitting model for the attack on this leakage. However, the masking does not prevent the S-box output leakage model.The best-found model for the S-box output leakage model can recover a partial key after only 20 traces.

This work gives the first example of a side-channel attack on a protected software implementation of Ascon. The exploited leakages we present are not specific to software and could be, depending on the implementation, applied to hardware. What is more, since domain-oriented masking was mostly designed to protect hardware with side-channel analysis in mind, DLSCA should be more challenging. In future work, we plan to explore protected hardware implementation of Ascon with the presented leakage models and model optimization methods. It could also be interesting to apply non-profiled DLSCA to Ascon.

## References

[BBBK11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[CCD00] Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous. Differential power analysis in the presence of hardware countermeasures. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings*, volume 1965 of *Lecture Notes in Computer Science*, pages 252–263. Springer, 2000.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[DEMS21a] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon PRF, MAC, and short-input MAC. Cryptology ePrint Archive, Report 2021/1574, 2021. https://eprint.iacr.org/2021/1574.

[DEMS21b] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *J. Cryptol.*, 34(3):33, 2021.

[FYHF21] Yuta Fukuda, Kota Yoshida, Hisashi Hashimoto, and Takeshi Fujino. Deep learning side-channel attacks against lightweight SCA countermeasure RSM-AES. In *Asian Hardware Oriented Security and Trust Symposium, AsianHOST 2021, Shanghai, China, December 16-18, 2021*, pages 1–6. IEEE, 2021.

[GMH13] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 6645–6649. IEEE, 2013.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In Begül Bilgin, Svetla Nikova, and Vincent Rijmen, editors, *Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016*, page 3. ACM, 2016.

[HGM$^+$11] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *J. Cryptogr. Eng.*, 1(4):293–302, 2011.

[HPGM16] Annelie Heuser, Stjepan Picek, Sylvain Guilley, and Nele Mentens. Side-channel analysis of lightweight ciphers: Does lightweight equal easy? In Gerhard P. Hancke and Konstantinos Markantonakis, editors, *Radio Frequency Identification and IoT Security - 12th International Workshop, RFIDSec 2016, Hong Kong, China, November 30 - December 2, 2016, Revised Selected Papers*, volume 10155 of *Lecture Notes in Computer Science*, pages 91–104. Springer, 2016.

[HPGM20] Annelie Heuser, Stjepan Picek, Sylvain Guilley, and Nele Mentens. Lightweight ciphers and their side-channel resilience. *IEEE Transactions on Computers*, 69(10):1434–1448, 2020.

[JXLW20] Xin Jin, Yong Xiao, Shiqi Li, and Suying Wang. Deep learning-based side channel attack on HMAC SM3. *Int. J. Interact. Multim. Artif. Intell.*, 6(4):113–120, 2020.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[KFYF21] Kunihiro Kuroda, Yuta Fukuda, Kota Yoshida, and Takeshi Fujino. Practical aspects on non-profiled deep-learning side-channel attacks against AES software implementation with two types of masking countermeasures including RSM. In Chip-Hong Chang, Ulrich Rührmair, Stefan Katzenbeisser, and Debdeep Mukhopadhyay, editors, *ASHES@CCS 2021: Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security, Virtual Event, Republic of Korea, 19 November 2021*, pages 29–40. ACM, 2021.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA,* *August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[LJB+89] Yann LeCun, Lawrence D. Jackel, Bernhard E. Boser, John S. Denker, Hans Peter Graf, Isabelle Guyon, Don Henderson, Richard E. Howard, and Wayne E. Hubbard. Handwritten digit recognition: applications of neural network chips and automatic learning. *IEEE Commun. Mag.*, 27(11):41–46, 1989.

[Mag20] Houssem Maghrebi. Deep learning based side-channel attack: a new profiling methodology based on multi-label classification. Cryptology ePrint Archive, Report 2020/436, 2020. https://eprint.iacr.org/2020/436.

[MBA+23] Kamyar Mohajerani, Luke Beckwith, Abubakr Abdulgadir, Eduardo Ferrufino, Jens-Peter Kaps, and Kris Gaj. SCA evaluation and benchmarking of finalists in the NIST lightweight cryptography standardization process. *IACR Cryptol. ePrint Arch.*, page 484, 2023.

[MO23a] Thomas Marquet and Elisabeth Oswald. A comparison of multi-task learning and single-task learning approaches. *IACR Cryptol. ePrint Arch.*, page 611, 2023.

[MO23b] Thomas Marquet and Elisabeth Oswald. Exploring multi-task learning in the context of two masked AES implementations. *IACR Cryptol. ePrint Arch.*, page 6, 2023.

[Moc77] Jonas Mockus. On bayesian methods for seeking the extremum and their application. In Bruce Gilchrist, editor, *Information Processing, Proceedings of the 7th IFIP Congress 1977, Toronto, Canada, August 8-12, 1977*, pages 195–200. North-Holland, 1977.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 3–26. Springer, 2016.

[MS23] Loïc Masure and Rémi Strullu. Side-channel analysis against anssi's protected AES implementation on ARM: end-to-end attacks with multi-task learning. *J. Cryptogr. Eng.*, 13(2):129–147, 2023.

[MWM21] Thorben Moos, Felix Wegener, and Amir Moradi. Dl-la: Deep learning leakage assessment: A modern roadmap for sca evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):552–598, Jul. 2021.

[Ngu19] Vu Nguyen. Bayesian optimization for accelerating hyper-parameter tuning. In *2nd IEEE International Conference on Artificial Intelligence and Knowledge Engineering, AIKE 2019, Sardinia, Italy, June 3-5, 2019*, pages 302–305. IEEE, 2019.

[NTIY23] Yusuke Nozaki, Shu Takemoto, Yoshiya Ikezaki, and Masaya Yoshikawa. Deep learning based side-channel analysis for lightweight cipher PRESENT. In *15th International Conference on Computer and Automation Engineering, ICCAE 2023, Sydney, Australia, March 3-5, 2023*, pages 570–574. IEEE, 2023.

[PPM+23] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. Sok: Deep learning-based physical side-channel analysis. *ACM Comput. Surv.*, 55(11):227:1–227:35, 2023.

[PSB+18] Emmanuel Prouff, Remi Strullu, Ryad Benadjila, Eleonora Cagli, and Cecile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. Cryptology ePrint Archive, Report 2018/053, 2018. https://eprint.iacr.org/2018/053.

[PSK+18] Stjepan Picek, Ioannis Petros Samiotis, Jaehun Kim, Annelie Heuser, Shivam Bhasin, and Axel Legay. On the performance of convolutional neural networks for side-channel analysis. In Anupam Chattopadhyay, Chester Rebeiro, and Yuval Yarom, editors, *Security, Privacy, and Applied Cryptography Engineering - 8th International Conference, SPACE 2018, Kanpur, India, December 15-19, 2018, Proceedings*, volume 11348 of *Lecture Notes in Computer Science*, pages 157–176. Springer, 2018.

[PWP22] Guilherme Perin, Lichao Wu, and Stjepan Picek. Exploring feature selection scenarios for deep learning-based side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):828–861, Aug. 2022.

[RAD20] Keyvan Ramezanpour, Paul Ampadu, and William Diehl. SCARL: side-channel analysis with reinforcement learning on the ascon authenticated cipher. *CoRR*, abs/2006.03995, 2020.

[RK21] Tanu Shree Rastogi and Elif Bilge Kavun. Deep learning techniques for side-channel analysis on AES datasets collected from hardware and software platforms. In Alex Orailoglu, Matthias Jung, and Marc Reichenbach, editors, *Embedded Computer Systems: Architectures, Modeling, and Simulation - 21st International Conference, SAMOS 2021, Virtual Event, July 4-8, 2021, Proceedings*, volume 13227 of *Lecture Notes in Computer Science*, pages 300–316. Springer, 2021.

[RW06] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning.* Adaptive computation and machine learning. MIT Press, 2006.

[RWPP21] Jorai Rijsdijk, Lichao Wu, Guilherme Perin, and Stjepan Picek. Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):677–707, Jul. 2021.

[SD17] Niels Samwel and Joan Daemen. DPA on hardware implementations of ascon and keyak. In *Proceedings of the Computing Frontiers Conference, CF'17, Siena, Italy, May 15-17, 2017*, pages 415–424. ACM, 2017.

[SS23] Dillibabu Shanmugam and Patrick Schaumont. Improving side-channel leakage assessment using pre-silicon leakage models. In Elif Bilge Kavun and Michael Pehl, editors, *Constructive Side-Channel Analysis and Secure Design - 14th International Workshop, COSADE 2023, Munich, Germany, April 3-4, 2023, Proceedings*, volume 13979 of *Lecture Notes in Computer Science*, pages 105–124. Springer, 2023.

[Tea] ASCON Team. Ascon C repository.

[TEM+21] Ryan Turner, David Eriksson, Michael McCourt, Juha Kiili, Eero Laaksonen, Zhen Xu, and Isabelle Guyon. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. *CoRR*, abs/2104.10201, 2021.

[THM$^+$07] Michael Tunstall, Neil Hanley, Robert McEvoy, Claire Whelan, Colin Murphy, and William Marnane. Correlation power analysis of large word sizes. 09 2007.

[WNS21] Colin White, Willie Neiswanger, and Yash Savani. BANANAS: bayesian optimization with neural architectures for neural architecture search. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 10293–10301. AAAI Press, 2021.

[WPP20] Lichao Wu, Guilherme Perin, and Stjepan Picek. I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. *IACR Cryptol. ePrint Arch.*, page 1293, 2020.