cryptography

Article

Side-Channel Attacks on Masked Bitsliced Implementations of AES

Anca Rădulescu and Marios O. Choudary *

Faculty of Automatic Control and Computer Science, University Politehnica Bucharest, 060042 Bucharest, Romania; anca.radulescu2402@stud.acs.upb.ro
* Correspondence: marios.choudary@cs.pub.ro

Abstract: In this paper, we provide a detailed analysis of CPA and Template Attacks on masked implementations of bitsliced AES, targeting a 32-bit platform through the ChipWhisperer side-channel acquisition tool. Our results show that Template Attacks can recover the full AES key successfully within 300 attack traces even on the masked implementation when using a first-order attack (no pre-processing). Furthermore, we confirm that the SubBytes operation is overall a better target for Template Attacks due to its non-linearity, even in the case of bitsliced implementations, where we can only use two bits per key byte target. However, we also show that targeting the AddRoundKey can be used to attack bitsliced implementations and that, in some cases, it can be more efficient than the SubBytes attack.

Keywords: Template Attacks; bitsliced AES; masking

check for updates

Citation: Rădulescu, A.; Choudary, M.O. Side-Channel Attacks on Masked Bitsliced Implementations of AES. *Cryptography* **2022**, *6*, 31. https://doi.org/10.3390/cryptography6030031

Academic Editor: Jim Plusquellic

Received: 11 May 2022
Accepted: 17 June 2022
Published: 28 June 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Cryptographic algorithms, such as AES [1], are widely used nowadays to protect our data, for network communications, cloud data storage, or local computers. In all of these applications, we desire secure and fast computations. To speed up the AES computation, an implementation known as bitsliced has been proposed [2], which allows specialised hardware to perform hundreds of encryptions in parallel [3]. Furthermore, the bitsliced implementation of AES can protect this algorithm against cache-based timing attacks [4].

However, besides the cache-based timing attacks thwarted by the bitsliced implementation of AES, we still have other side-channel threats, such as power [5] and electromagnetic analyses [6]. These attacks have become critical threats to electronic devices, ranging from simple 8-bit and 16-bit microcontrollers (as used in smart cards) up to complex, full-fledged 32-bit and 64-bit system-on-a-chip devices, as used in the gateway devices in smart homes, smart cards, smartphones, etc.

To protect against these power or electromagnetic analysis attacks, several countermeasures have been introduced, among which, masking is very popular, due to its potential to break the correlation between the power consumption and the secret values processed by the target device through the use of random shares [7]. Hence, masking implementations have been developed as well for the bitsliced version of AES [8–10].

In past years, there have been several publications regarding attacks on masked implementations of bitsliced AES [9,11–15], using various attack and evaluation methods, primarily correlation power analyses [16], Template Attacks [17], and mutual information [18].

However, there are some shortcomings of these evaluations, such as the common use of only one or two key bits with CPA. Furthermore, existing evaluations do not provide successful results when using CPA with more bits or Template Attacks without preprocessing the traces.

Hence, in this paper, we take a closer look at the efficacy of side-channel attacks on bitsliced implementations of AES, both with and without masking protections, when targeting a 32-bit architecture. To provide a wider picture of the success of these attacks, we used Template Attacks as well as two flavors of CPA, targeting the input of the S-BOX operation (AddRoundKey) and the output of the S-BOX. Furthermore, for CPA, we used both a direct attack as well as a pre-processing step to defeat the masking protection.

We performed our evaluations of CPA and Template Attacks on real traces obtained using a ChipWhisperer device. We obtained interesting and unforeseen results using both of these attacks. Concerning Template Attacks, we were able to successfully determine the full AES key within 20 attack traces without countermeasures and within 300 attack traces with first-order masking, without any pre-processing of the traces. As for CPA, contrary to the expectations, we observed cases where attacking the AddRoundKey could lead to a successful key recovery, while the common attacks on the SubBytes operation did not reveal the key, even with a large number of attack traces.

In the following section, we will provide some background on side-channel attacks, masking, and bitsliced implementations. In Section 3, we detail our attack methods. Our experimental setup is presented in Section 4. Afterward, in Section 5, we present our results.

2. Background

2.1. Side-Channel Attacks

Side-channel attacks exploit various physical channels that may carry information about the data processed by an electronic device (e.g., microcontroller). Among these, the running time, power consumption, and electromagnetic emissions are very popular. In this paper, we focus on power consumption attacks, also known as power analysis attacks.

Power analysis attacks use the power consumption leakages of devices while running encryption algorithms. The power consumed is dependent on the processed data, as well as on the performed operations [5]. Based on the monitored power leakages for a set of plaintext/ciphertext encryptions, the attacker may recover the key by performing statistical analyses over hypothetical values of subsets of the key (in the case of AES, a common approach is to iterate over each byte of the key).

Two main classes of power analysis attacks can be distinguished: unprofiled (model-based) and profiled. Unprofiled attacks use a leakage model to determine the connection between the traces and the encryption key. Profiled attacks use a learned model of the target device in order to determine the secret key bytes as quickly and accurately as possible. These profiled attacks are split into two stages. First, it is assumed that the attacker has access to a clone device, which is used to acquire the profiling traces and derive an accurate leakage model. Afterward, this learned model is used on the traces collected from the target device to extract the encryption key. Although profiled attacks require access to a copy of the target device (in contrast to model-based attacks), these attacks are reportedly more powerful than model-based attacks.

Among the unprofiled attacks, correlation power analysis (CPA) [16] is one of the most common and powerful attacks, if we use a suitable leakage model, such as the Hamming weight or Hamming distance, which usually work well on many target devices. Among profiled attacks, Template Attacks [17,19] are known to be very efficient, since they use all of the information from a side-channel trace, both signal and noise in the detection of the target key. Hence, in this paper, we used these two attacks for our evaluations.

2.1.1. Correlation Power Analysis (CPA)

This kind of attack is part of the unprofiled group of side-channel attacks and relies on assuming a leakage model, such as the commonly observed Hamming weight or Hamming distance [16]. The procedure is composed of the following steps:

1. Traces acquisition: Power consumption traces are acquired for a large set of input plaintexts, encrypted with the same key, generating a set of output ciphertext. Suppose we have n traces available $t_1, \ldots, t_n$, corresponding to plaintext $p_1, \ldots, p_n$ and ciphertexts $c_1, \ldots, c_n$. All of these are associated with the input key k. Each trace is a

vector composed of a number of samples m. We will later refer to sample j of trace i as t_{i_j}.

2. Intermediate value selection and computation: Sensitive variables v depend on both the key and the plaintext/ciphertext. The selection is dependent on the encryption algorithm as well. In the current analysis, two scenarios were used: the output of the first key addition (ARK) and the output of the first S-BOX.
3. Attack leakage model selection and running: In this paper, we focus on correlation power analysis (CPA) attacks, which rely on generating a hypothetical power consumption. A typical leakage model is based on the Hamming weight, defined as follows:

$$HW(x) = \sum_{i=1}^{n} x[i], \tag{1}$$

where $x \in \mathcal{F}_2^n$ [20] (i.e., the total number of bits set to 1 of the value x).
A second model relies on the Hamming distance between two variables, defined as:

$$HD(x,y) = HW(x \oplus y). \tag{2}$$

We are interested in the correlation between the actual power consumption (recorded in the trace) and the hypothetical power values, which are computed considering all sample points of interest (where the intermediate values should occur). The correlation is computed using Pearson's correlation coefficient:

$$\rho_{x,y} = \frac{cov(x,y)}{\sigma_x \sigma_y}. \tag{3}$$

Essentially, considering that the *sensitive* variable leaks between samples j_1 and j_2 ($j_1 \leq j_2$) and considering a hypothetical consumption h of the variable, then, for each sample j ($j_1 \leq j \leq j_2$), we compute the correlation between h and the vector comprised of each trace sample j: $\rho_{v_j,h}$, where v_j=(t_{1_j}, t_{1_j} ..., t_{n_j}). However, h does depend on the key.
In particular, if we define the subkey $k[byte]$ ($byte \in (0,16)$) as a byte of the entire key (k), then we would compute the correlations for each $k[byte] = value$, where $value \in (0,256)$.
4. Identify the correct subkey: the correct key byte ($k[byte]$) value should correspond to the highest correlation value.

2.1.2. Template Attack

This kind of attack is representative of the profiled attacks and is known as one of the strongest side-channel attacks. It includes two stages: A profiling stage and the actual attack stage [17]. These are outlined in the following steps:

1. Traces acquisition on training device: The attacker acquires large datasets of power traces using different plaintext inputs and keys to build a sound model of the power consumption for the target computations [21]. Typical target computations are those operations that combine known input, such as plaintext bytes, with the unknown target key bytes of cryptographic algorithms. A canonical example is the output of the S-BOX in AES, where the power consumption depends on one plaintext byte and one key byte.
2. Template generation: The datasets are used to derive a statistical model of the device, eventually applying a compression (or sample selection) [19]. This typically results in a set of mean vectors (one per candidate key byte value) and covariance matrices, known as the template parameters.
3. Traces acquisition on victim device: The attacker acquires a relatively small number of power traces on the target device, using different input plaintexts. For the attack to be effective, the training and victim devices should be of the same kind. In a practical

attack scenario, the attacker will perform the profiling step in one device (where he can run extensive data acquisition and experimentation), and the attack step on the victim device. However, for evaluation purposes, it is also possible to use the same device in order to attest to the leakage of a particular implementation or to compare different methods of attack or protection.

4. Attack setup: The model is applied to the attack traces. The output consists of a list of scores or probabilities for each possible candidate value in the subkey space.
5. Identify the correct subkey: The correct subkey value is considered as the one corresponding to the highest probability in the list of scores (or probabilities).

2.2. Masking

A typical countermeasure to thwart side-channel attacks is to mask the sensitive variables [5,9]. Essentially, masking implies generating random values for each sensitive variable, which is then concealed by applying an operation between these values. This implies the splitting of the sensitive variables (v) into $(d+1)$ shares $v_m, m_1, ..., m_d$, such that [5]:

$$v_m = v * m_1 * m_2 * ... * m_d, \tag{4}$$

where v_m is the resulting masked value and $*$ is dependent on the operations employed in the cryptographic algorithm. In particular, three operations are extensively discussed: XOR function ($\oplus$), modular addition ($+$), and multiplication ($\times$). The first one lies at the core of Boolean masking, while the last two define the arithmetic masking. These operations are central to several masking algorithms. The splitting into $(d+1)$ shares ensures a d-th order masking construction.

The d masks are generated randomly, while one of the shares (v_m) is computed using relation 4. The subsequent computations to be performed are done separately for each share.

The concept of masking was first introduced in [7,22]. Ever since, this topic has been subject to extensive research and various schemes have been proposed. The differences between the masking schemes concern mostly the methods to perform the splitting. The encryption algorithms themselves play a key role, as well, since the operations used within masking are compatible with the operations performed in the algorithms. For further reference, we point to [23–27].

Attacks on Masking

Attacks on the unprotected implementations are called first-order attacks. In principle, if implemented efficiently (i.e., if the shares are indeed statistically independent), d-th order masked implementations should thwart a d-order attack but are still vulnerable to higher-order attacks.

These latter types of attacks exploit joint leakages of the intermediate variables and require a preprocessing of the traces, using a combining function over distinct pairs of samples (t_{i_j}, t_{j_k}) along the leakage trace:

$$t'_{i_{j,k}} = f(t_{i_j}, t_{i_k}), \tag{5}$$

where $i < n$ (number of traces), $j, k \leq m$, and m is the total number of samples. The preprocessing goal is to catch the pairs where both the masks and the masked variables leak. Two main preprocessing methods have been proposed and analysed over time [7,28]:

- Absolute difference:
$$f(x,y) = |x-y| \tag{6}$$
- Product combining:
$$f(x,y) = x \times y; \tag{7}$$

In our study, we used the classic absolute difference as the preprocessing function, which is a commonly used method. Nevertheless, as shown in some publications [20,29], certain flavours of the product-combining function may also lead to good results.

2.3. Bitsliced AES

Bitsliced implementations involve decompositions of the algorithms into bit-level instructions, using logic gate operations. These implementation types offer resistance against cached timing attacks while failing to withstand other side-channel attacks (such as power analysis), without proper countermeasures.

We recall the classical 4×4 byte array representation of the AES state:

$$s = \begin{pmatrix} s_{1,1} & s_{1,2} & s_{1,3} & s_{1,4} \\ s_{2,1} & s_{2,2} & s_{2,3} & s_{2,4} \\ s_{3,1} & s_{3,2} & s_{3,3} & s_{3,4} \\ s_{4,1} & s_{4,2} & s_{4,3} & s_{4,4} \end{pmatrix} \tag{8}$$

In this particular case, we followed the bitsliced representation in [9], which remaps two states s and s' (associated with two plaintexts) into eight 32-bit registers (R_i, $1 \leq i \leq 8$). Each register contains the i_{th} bit from each of the 16 bytes of the two plaintexts.

$$R_i = \begin{pmatrix} s_{1,1_i} & s'_{1,1_i} & s_{1,2_i} & s'_{1,2_i} & \cdots & s_{4,4_i} & s'_{4,4_i} \end{pmatrix} \tag{9}$$

This new representation (see Figure 1) brings forth the following tailoring of the AES operations:

- AddRoundKey: The key is also converted into the bitsliced representation, considering Equation (9). At this point, in the standard implementation, the state is XORed with the round key $s \leftarrow s \oplus k_r$, where $r \in (1, 10)$ is the current round index. Considering a bitsliced representation, the XOR can be done bit-level-wise, between the corresponding bits.
- ByteSub: This is the only non-linear step of the algorithm. Whereas in the standard implementations, each byte of the state is replaced with the corresponding value mapped by a LUT (S-BOX table), in the bitsliced variant, logic gate operations (XOR, XNOR, AND) are used to implement the S-BOX. As pointed out in both [4,9], several solutions have been proposed for implementing the S-BOX, considering aspects such as performance, throughput, and the architecture targeted. Both articles pinpoint the compact S-BOX implementation described by Canright [30], which is also the one considered in the current study, employing multi-level arithmetic representation.
- ShiftRows: In a standard implementation, each row of the 4×4 byte array is shifted to the left with a displacement equal to the row index—1.

  $$ShiftRows(s) = \begin{pmatrix} s_{1,1} & s_{1,2} & s_{1,3} & s_{1,4} \\ s_{2,2} & s_{2,3} & s_{2,4} & s_{2,1} \\ s_{3,3} & s_{3,4} & s_{3,1} & s_{3,2} \\ s_{4,4} & s_{4,1} & s_{4,2} & s_{4,3} \end{pmatrix}$$

  The same convention is valid in the bitsliced approach, by adjusting accordingly the bits in every register.
- MixColumns: This step is applied at the column level and can be seen as a matrix multiplication in $GF(2^8)$. The logic gate described in [4] was implemented in the current setup.

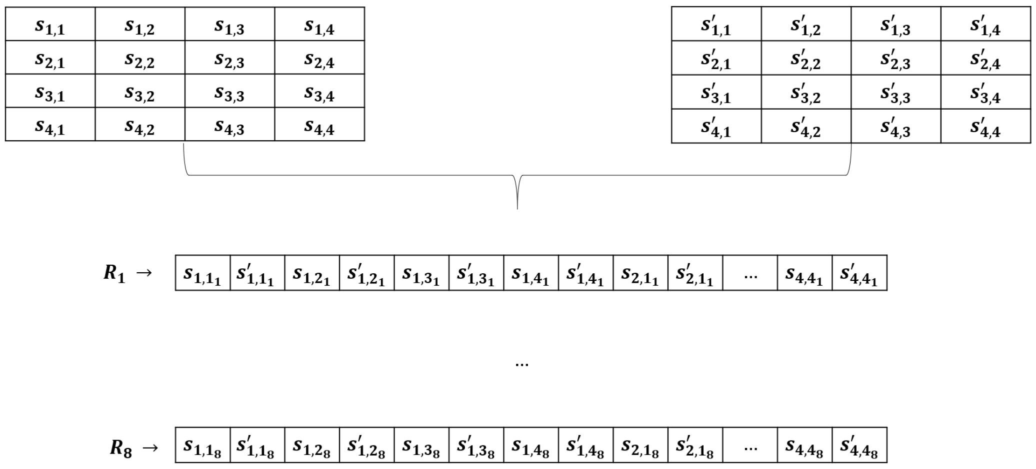

$s_{1,1}$ $s_{1,2}$ $s_{1,3}$ $s_{1,4}$
$s_{2,1}$ $s_{2,2}$ $s_{2,3}$ $s_{2,4}$
$s_{3,1}$ $s_{3,2}$ $s_{3,3}$ $s_{3,4}$
$s_{4,1}$ $s_{4,2}$ $s_{4,3}$ $s_{4,4}$
$s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$ $s'_{1,4}$
$s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$ $s'_{2,4}$
$s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$ $s'_{3,4}$
$s'_{4,1}$ $s'_{4,2}$ $s'_{4,3}$ $s'_{4,4}$
$R_1 \rightarrow$ $s_{1,1_1}$ $s'_{1,1_1}$ $s_{1,2_1}$ $s'_{1,2_1}$ $s_{1,3_1}$ $s'_{1,3_1}$ $s_{1,4_1}$ $s'_{1,4_1}$ $s_{2,1_1}$ $s'_{2,1_1}$ ... $s_{4,4_1}$ $s'_{4,4_1}$
...
$R_8 \rightarrow$ $s_{1,1_8}$ $s'_{1,1_8}$ $s_{1,2_8}$ $s'_{1,2_8}$ $s_{1,3_8}$ $s'_{1,3_8}$ $s_{1,4_8}$ $s'_{1,4_8}$ $s_{2,1_8}$ $s'_{2,1_8}$ ... $s_{4,4_8}$ $s'_{4,4_8}$

Figure 1. Bitsliced AES state representation.

We followed the observation from [9] and wrote the implementation using software macros for the elementary operations: XOR, AND, NOT, MOV, ROTL. This facilitated the integration of a protected/masked implementation using shares, requiring only a slight refactoring of the macros (i.e., to include the second share for straightforward management of the masks for the masked implementation analyses).

Masked Bitsliced AES

For our evaluations on masked bitsliced AES, we considered the first-order Boolean masking scheme. In this case, the algorithm structure was slightly adapted to include mask management.

As mentioned above, the key and input plaintext pair were mapped to the bitsliced representation, each spread over the eight 32-bit registers. As an additional step, the key and the plaintext masks were randomly generated, once again, each of them being spread over the eight 32-bit registers. Next, the masked bitsliced state and the masked bitsliced key were computed (using the *XOR* operation).

From this point onward, all AES operations were performed over both the masks and the masked bitsliced state. AddRoundKey, ShiftRows, and MixColumns could be straightforwardly applied for each of them independently. However, *ByteSub* is not linear and uses AND gates. In this case, the AND scheme in [31] was implemented. Our resulting implementation is equivalent to that used by Balasch et al. [9].

The attacks on bitsliced implementations (masked or not) follow the same principles outlined above, with the attack model taking into account the bits remapping table.

3. Attacks

We analysed the implementation resistance against both CPA and Template Attacks, considering two target operations. In this section, we highlight the key aspects of each attack.

3.1. Reference CPA Attacks on Bitsliced AES

Each byte of the plaintext was propagated across the eight registers (R_1, R_2, ..., R_8) and two plaintexts were encrypted with the same key k; thus, each register contained a pair of bits corresponding to the same key byte encryption. For example, considering Figures 1 and 2, the first two bits in R_1 ($s_{1,1_1}$ and $s'_{1,1_1}$) were encrypted with the same key byte ($k_{1,1}$). As proposed in [9], the attack can therefore be mounted using classic CPA, based on the sum of the two bits affected by the same byte of the key. This becomes the leakage model applied in the attack:

$$\begin{aligned}
\text{(byte 1)}hw &= s_{1,1R_{idx}} + s'_{1,1R_{idx}} \\
\text{(byte 2)}hw &= s_{1,2R_{idx}} + s'_{1,2R_{idx}} \\
&\dots \\
\text{(byte 16)}hw &= s_{4,4R_{idx}} + s'_{4,4R_{idx}}
\end{aligned} \tag{10}$$

The attack is summarised in Algorithm 1.

Algorithm 1 CPA attack on first S-BOX.

```
Input:
    measured/simulated n traces $t_1, \dots, t_n$, each trace having $m$ samples
    $n$ number of pairs of 16-byte plaintext $(p_{1,1}, p_{1,2}) \dots, (p_{n,1}, p_{n,2})$ generated
  with the same key ($k$)
    the register number $R_x$ which we want to target ($x \in [1, 8]$)
    standard2bitslice is the LUT mapping a byte index (in the standard
  representation) to a bit index in a bitsliced register
Output:
    16 byte-sized encryption key ($k$)—of the first round
Steps:
for $byte = 1$ to 16 do
    $max_{corr} = 0$
    $k^* = 0$
    for $value = 0$ to 255 do
        $k[byte] = value$
        $bitsliced_k \leftarrow$ bitslice representation of $k$
        for $i = 1$ to $n$ do
            $state \leftarrow$ bitslice representation of $(p_{i,1}, p_{i,2})$; see Figure 1
            $ark = state \oplus bitsliced_k$
            $sbox = \text{S-BOX}(ark)$
            $idx = standard2bitslice(byte)$
            $h_{k[byte],i} = R_x[idx] + R_x[idx + 1]$
        end
        for $i = 1$ to $m$ do
            $v_i = (t_{1_i}, ..., t_{n_i})$
            $correlation(k[byte]) = max(\rho(h_{k[byte],1:n}, v_i))$
        end
        if $correlation(k[byte]) \geq max_{corr}$ then
            $max_{corr} = correlation(k[byte])$
            $k^* = k[byte]$
        end
    end
    $k[byte] = k^*$.
end
```

For the second-order attack (for the masked implementation), the same attack scheme can be applied, except that the traces are replaced by the preprocessed traces (using the absolute difference).

3.2. Alternative CPA Attack

Although the CPA attack presented in [9] bears successful results, it limits the attack to using only two bits. We investigated an alternative attack scheme, which harvests the information from all bits of a plaintext byte. Instead of attacking the first round S-BOX output, we considered the output of the first round key addition (ARK), which is the input to the aforementioned S-BOX. Instead of targeting a specific register, we defined new subkeys for which we could target eight bits at a time. Specifically, as suggested in Figure 2, we identified a subkey (K_i) as the sequence of eight consecutive bits in one register, associated with one plaintext (remember that two consecutive bits are associated with the same key byte, but are derived from the two different plaintext bytes). Therefore, we focused solely on one plaintext and ignored the contribution from the other one. It would also be possible to consider the contributions from both plaintexts by using a Hamming weight on 16 bits, which resulted in slightly better results in our experiments. However, for simplicity, and a more clear presentation of our findings, we omit their presentation in this paper.

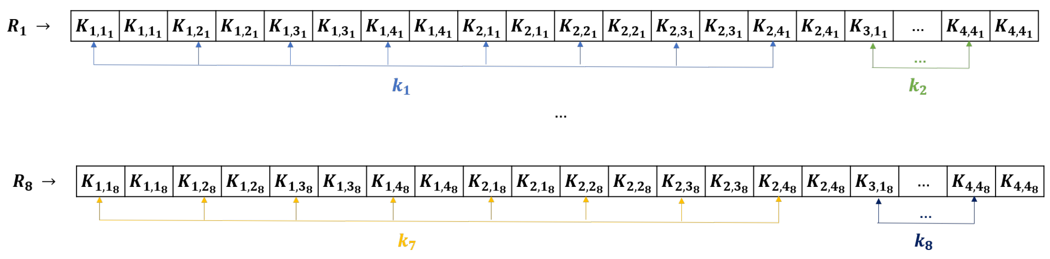

$R_1 \rightarrow$
$K_{1,1_1}$ $K_{1,1_1}$ $K_{1,2_1}$ $K_{1,2_1}$ $K_{1,3_1}$ $K_{1,3_1}$ $K_{1,4_1}$ $K_{1,4_1}$ $K_{2,1_1}$ $K_{2,1_1}$ $K_{2,2_1}$ $K_{2,2_1}$ $K_{2,3_1}$ $K_{2,3_1}$ $K_{2,4_1}$ $K_{2,4_1}$ $K_{3,1_1}$ ... $K_{4,4_1}$ $K_{4,4_1}$
k_1
k_2
...
$R_8 \rightarrow$
$K_{1,1_8}$ $K_{1,1_8}$ $K_{1,2_8}$ $K_{1,2_8}$ $K_{1,3_8}$ $K_{1,3_8}$ $K_{1,4_8}$ $K_{1,4_8}$ $K_{2,1_8}$ $K_{2,1_8}$ $K_{2,2_8}$ $K_{2,2_8}$ $K_{2,3_8}$ $K_{2,3_8}$ $K_{2,4_8}$ $K_{2,4_8}$ $K_{3,1_8}$ ... $K_{4,4_8}$ $K_{4,4_8}$
k_7
k_8

Figure 2. Alternative attack: using eight bits.

Considering Figure 1, the leakage model for each key byte can be summarised below:

$$
\begin{aligned}
(\text{byte } 1) hw &= \sum_{i=1}^{4} s_{1,i_{R_1}} + \sum_{i=1}^{4} s_{2,i_{R_1}} \\
(\text{byte } 2) hw &= \sum_{i=1}^{4} s_{3,i_{R_1}} + \sum_{i=1}^{4} s_{4,i_{R_1}} \\
(\text{byte } 3) hw &= \sum_{i=1}^{4} s_{1,i_{R_2}} + \sum_{i=1}^{4} s_{2,i_{R_2}} \\
&\cdots \\
(\text{byte } 16) hw &= \sum_{i=1}^{4} s_{1,i_{R_8}} + \sum_{i=1}^{4} s_{2,i_{R_8}}
\end{aligned}
\tag{11}
$$

This new scheme shifts the focus from deriving the standard representation key byte values $(k_{i,j}, i, j \leq 4)$ to deriving the key byte values in the bitsliced representation $(K_i, i \leq 8)$, which will require the additional step of converting the guessed subkey values to the standard representation. We mounted this attack on both unprotected implementations (on the acquired traces) and the first-order masked implementation (on the preprocessed traces). The updated Algorithm 2 is written in pseudocode below.

Algorithm 2 CPA attack on First ARK.

```
Input:
    measured/simulated n traces $t_1, \ldots, t_n$, each trace having $m$ samples
    $n$ number of pairs of 16-byte plaintext $(p_{1,1}, p_{1,2}), \ldots, (p_{n,1}, p_{n,2})$ generated
  with the same key ($k$)
    standard2bitslice is the LUT mapping a byte index (in the standard
  representation) to a bit index in a bitsliced register
Output:
    16 byte-sized encryption key ($k$)—of the first round
Steps:
for $byte = 1$ to $8$ do
    $max_{corr} = 0$
    $K^* = 0$
    for $value = 0$ to $255$ do
        $K[byte] = value$
        for $i = 1$ to $n$ do
            $state \leftarrow$ bitsliced representation of $(p_{i,1}, p_{i,2})$
            $ark = state \oplus K[byte]$
            $h_{K[byte],i} = HW(ark)$
        end
        for $i = 0$ to $m$ do
            $v_i = (t_{1_i}, \ldots, t_{n_i})$
            $correlation(K_{byte}) = max(\rho(h_{K[byte],1:n}, v_i))$
        end
        if $correlation(K[byte]) \geq max_{corr}$ then
            $max_{corr} = correlation(K[byte])$
            $K^* = K[byte]$
        end
    end
    $K[byte] = k^*$.
end
Convert $K$ to standard implementation to extract the key $k$.
```

3.3. Template Attacks on Bitsliced AES

We implemented and applied Template Attacks [17,19] to our unprotected and masked datasets. For the implementation, we followed the general approach described in Section 2.1. To improve the profiling step, we used stochastic models [32], which we combined with linear discriminant analysis (LDA) for compression [33]. This helped us to reduce the size of each trace from 5000 samples (which were carefully selected across our regions of interest, see more details below in Section 4) to eight samples per trace (in the projected space of LDA). We chose to keep only eight LDA dimensions based on the analysis of the eigenvalues.

In order to attack the S-BOX, we used the classic attack strategy described in Section 3.1. That is, we created model parameters for each possible output of the S-BOX when profiling the device using different plaintexts. While this ignores the bitsliced representation, it turns out to be very effective since the S-BOX output is computed at some point even in the bitsliced representation and the Template Attack can capture this.

As for the AddRoundKey context presented in Section 3.2, we targeted one byte at a time from a particular bitsliced register (see Figure 1), using a single plaintext, ignoring the second plaintext in each register. That is, we created Template Attack model parameters for consecutive bytes of the bitsliced representation corresponding to the first plaintext in each bitsliced register. Hence, the byte chunks we targeted were:

Byte 1: bit 1 of the first 8 bytes of the first plaintext (first bitsliced reg).
Byte 2: bit 1 of the last 8 bytes of the first plaintext (first bitsliced reg).
Byte 3: bit 2 of the first 8 bytes of the first plaintext (second bitsliced reg).
...
Byte 16: bit 8 of the last 8 bytes of the first plaintext (last bitsliced reg).

To present the success of these attacks, we used the empirical guessing entropy [34]; for the full-key, we employed the estimation algorithm of Glowacz et al. [35].

3.4. Prior Work

Several publications, such as [36–38], have already shown that problems remain with first-order masking and that it is possible to attack such implementations without any special preprocessing, due to unexpected interactions in hardware, glitches, transitions, etc., even when dealing with bitsliced implementations [9,12,15]. However, as we detail below, some interesting results were not presented in prior publications, particularly when dealing with bitsliced implementations, as in this present work.

Mangard et al. [36] have shown that first-order DPA attacks can work on first-order masked AES hardware implementations, without any special preprocessing, if they have good leakage models. However, in the same paper, they showed that a version of our AddRoundKey attack did not work, i.e., when they were targeting the AES state stored in registers. They were only able to successfully attack their implementation when targeting the logic gates that implemented some of the AES operations, such as the SubBytes operation. In contrast, we show here that the AddRoundKey operation can be successfully attacked in bitsliced masked implementations, at least in software, if we carefully group the bits of the target values.

Afterward, at CT-RSA 2007, Oswald and Mangard [37] analysed several implementations of Template Attacks against a software masked implementation of AES. They considered attacks with and without preprocessing of the traces to break a first-order masking implementation. However, even for the attacks without preprocessing, they still performed some additional work by computing the templates of each target value for each mask value. In contrast, we have shown that standard Template Attacks, without modeling the mask at all, can also be effective against first-order masked implementations.

As for bitsliced masked implementations, Balasch et al. [9] have shown successful CPA attacks on a software bitsliced masked implementation of AES, using a large number of traces when targeting the SubBytes operation. Similarly, Groot et al. [12] attacked a software bitsliced masked implementation of PRESENT using second-order CPA attacks. More recently, Han et al. [15] also analysed several bitsliced implementations of cryptographic algorithms, including AES, but still used CPA on a single bit. In all of these cases, they only used CPA attacks and a sub-optimal leakage model for the SubBytes state, considering only two bits of the key at a time. In contrast, we have considered more enhanced attacks (both against the SubBytes and the AddRoundKey operations) and have evaluated the success of Template Attacks against both target operations. In a different approach, Journault and Standaert [13] used leakage-detection tools based on the T-test to verify the existence of leakage and then the mutual information to quantify the possible leakage available in the traces. However, they did not perform our kind of attacks. Similarly, Azouaoui et al. [14] compared several bitsliced masking implementations using the mutual information tool to quantify the potential leakage available. In contrast, a CPA or Template Attack reveals the actual leakage extracted by an attacker.

In summary, prior work has indeed already investigated the efficiency of CPA and Template Attacks against various masking implementations of cryptographic algorithms. However, there have been several aspects that, to our knowledge, were not researched at all or not enough. These include: (i) The use of standard Template Attacks to break first-order masking, without any special preprocessing nor additional modeling of masks; (ii) The use of more efficient models for attacking bitsliced implementations, targeting both the AddRoundKey and SubBytes operations. Therefore, in this paper, we performed these additional experiments, obtaining interesting results, as we present them in the following sections.

An overview of these various evaluations, including our work, is given in Table 1. We identify with a "✓" the successful key recovery; with "X" the analysed implementations that withstood the attacks; and with "-" the analyses that were not performed in a specific study.

Table 1. Various attack implementations: successful key recovery.

	CPA S-BOX/1-2 Bits	CPA ARK/8 Bits	TA Simple	TA Preprocessing	T-Test or MI
Mangard et al. [36]	✓	X	-	-	-
Oswald et al. [37]	-	-	X	✓	-
Balasch et al. [9]	✓	-	-	-	-
Groot et al. [12]	✓	-	-	-	-
Han et al. [15]	✓	-	-	-	-
Journault et al. [13]	-	-	-	-	✓
Azouaoui et al. [14]	-	-	-	-	✓
Current paper	✓	✓	✓	-	-

4. Experimental Setup

ChipWhisperer [39] provides hardware and software solutions dedicated to side-channel analyses. In particular, ChipWhisperer-Lite integrates two components: a STM32F303 32-bit ARM target board and an oscilloscope.

The overall setup is straightforward; the board can be connected directly to the laptop using a micro-USB cable (USB controller). On the software side, a dedicated API is available to the user [40] facilitating the interface to the hardware (both the target and the oscilloscope).

The acquisition (see Figure 3) was configured to capture a sufficient, yet practical number of samples per trace, starting from the first AES round key addition to the end of the first AES round S-BOX operation (including samples from the subsequent ShiftRows and MixColumns operations, as a margin). The identification of the operations processing (in terms of samples) was performed by switching on/off different portions (rounds) of the algorithm. Considering that ChipWhisperer-Lite can record a maximum of 24,400 samples [40], we used a decimation factor to ensure that the entire range of samples of interest was captured. Furthermore, as written below, we used sampling rates of ≈3 Msamples/s, recalling that ChipWhisperer relies on the synchronous sampling technique [39], thus being optimised for such small sampling rates. The setup was tailored for each of the two implementations, as follows:

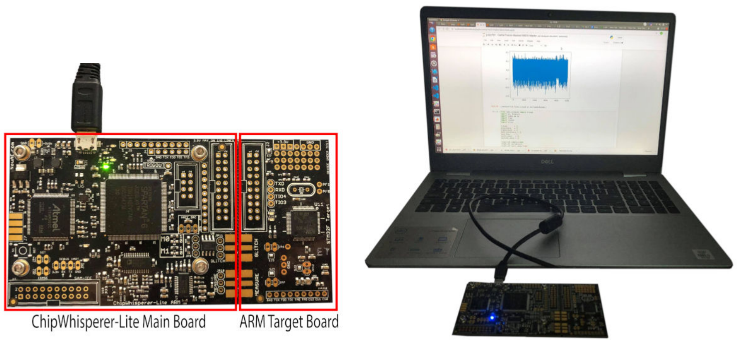


Figure 3. ChipWhisperer-Lite (CW1173) Setup.

4.1. Unprotected Bitsliced Implementation

The total number of samples for encryption is 54,848; the first round S-BOX processing ended at samples 3968. We decided to store the first 5000 samples, for each encryption, covering the operation of interest with a good margin. However, we highlight that we used a decimation factor of 10 (i.e., we stored every 1 in 10 samples). In this way, the first S-BOX should leak at samples around 400. Although we could have stored the full traces (not decimated), we decided to preserve the acquisition setup between the unprotected implementation and the masked implementation (see the subsection below). The ADC clock frequency is 29.5 MHz and the sampling rate ≈2.95 Msamples/s. Regarding the total number of traces, 100,000 traces were acquired, which were stored in the dataset and associated with the corresponding plaintext and ciphertext. Three files composed the dataset: a binary file with pairs of 16-byte plaintexts, a binary file with pairs of 16-bytes of ciphertext, and a binary file with sequences of 5000 32-bit floating-point values.

4.2. Masked Bitsliced Implementation

Due to the extra processing of the masks, the total number of samples for an encryption increases to 430,340 and the first round of S-BOX processing ends at sample number 41,984. Therefore, storing all samples up to the first S-BOX is rather infeasible, especially considering that ChipWhisperer Lite can record up to 24,400 samples per capture [40]. We decided to limit the capturing to store every 1 in 10 samples, by setting the decimation factor to 10. Furthermore, we limited the number of samples to 5000. Given that the S-BOX finishes at ≈45,000 samples (for a decimation factor of 1), we are confident that the S-BOX processing was captured within the first 5000 samples recorded with a decimation factor of 10 (should have finished at ≈4500 samples, in this case). Similar to the unprotected setup, the ADC clock frequency was 29.5 MHz and the sampling rate was ≈2.95 Msamples/s. Regarding the number of datasets, eight data batches were stored, each corresponding to 100,000 encryptions. Each data batch contained: a binary file with pairs of 16 bytes of plaintext, a binary file with pairs of 16 bytes of ciphertext, a binary file with sequences of 5000 32-bit floating-point values (power traces), a binary file with 32-byte masks for the plaintext pair (initial masks), 32-byte masks for the keys (initial masks). We also captured and stored the randomly generated mask after the S-BOX operation, as well as the corresponding masked S-BOX values, for checking purposes and for facilitating the second-order attacks mounting (to faster detect the leaking samples).

The clock cycles for the two implementations are outlined in Table 2. We note that the target clock frequency is 10 MHz.

Table 2. Clock cycles.

Implementation	**Clock cycles**
Unmasked implementation	360,934.3
Masked implementation	479,084.8

4.3. Evaluator Attack Scenario

For our experiments, we considered the scenario of an evaluator (evaluation lab), i.e., we used the same device for all experiments. This includes the Template Attacks, where the same Chip Whisperer device was used for both profiling and attack steps. While this might provide better results than what a real adversary can obtain in practice, it allows us to determine the feasibility of our attacks.

5. Results

5.1. Attacks on Unprotected Bitsliced Implementations of AES

The unprotected implementation is effectively vulnerable to first-order attacks, requiring but a few tens of traces for achieving confident CPA attack success as well as for successful Template Attacks.

5.1.1. Classic CPA on SubBytes (Targeting 2 Key Bits)

Figure 4 reveals the successful attack results for the first byte of the key, as an example, considering two perspectives. First, in Figure 4a, the correlation values associated with each sample and with each possible byte value (considering the first byte of the key) are depicted. The green values matching the highest correlation key byte guess value (43) correspond to the correct value. We further notice that the highest correlation value corresponds to the sample range where we have identified the S-BOX operation (around samples 4000). Second, Figure 4b depicts the highest correlation values for each possible value of the first byte of the key, outlined against an increasing number of traces used in the attack. The first byte is the easiest to attack and can be recovered confidently starting with a couple of ten traces.

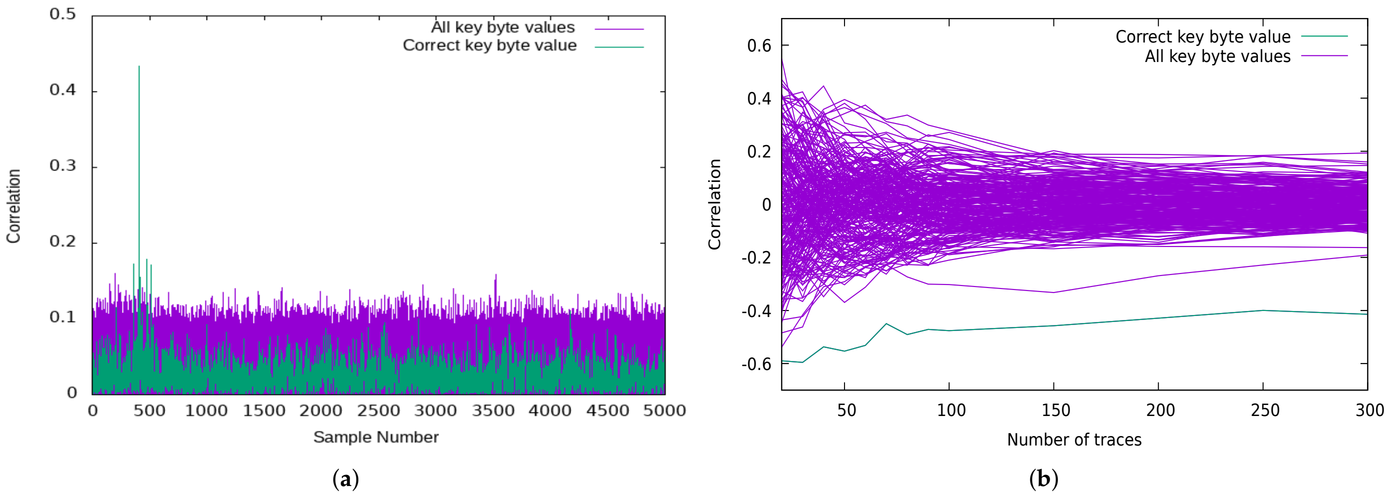


Figure 4. CPA 2-bit (SBOX) attack results for byte 1. (**a**) CPA correlation vs. sample. (**b**) CPA correlation vs. number of traces.

5.1.2. CPA Attacks on ARK (Targeting 8 Key Bits)

On a similar note, the attack on the ARK operation was successful; however, it required more traces than the classic attack on the SubBytes operation (Figure 5b); furthermore, the correlation gap between the correct and the other byte values was smaller in this case.

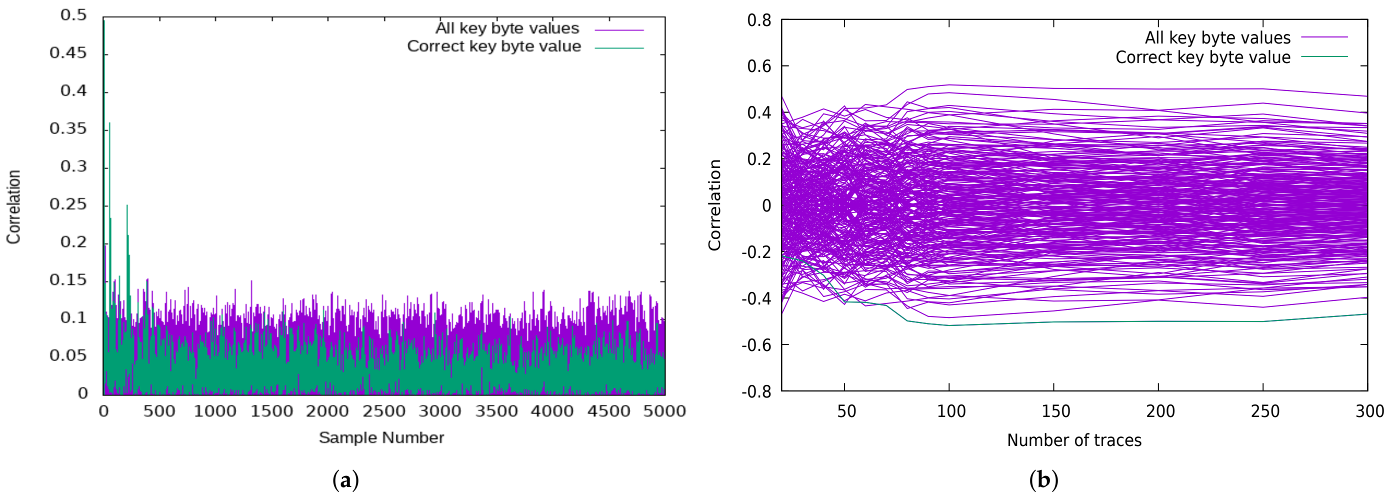


Figure 5. CPA 8-bit (ARK) attack results for byte 1. (**a**) CPA correlation vs. sample. (**b**) CPA correlation vs. number of traces.

5.1.3. Template Attacks

Template attacks require around 20 traces to recover one key byte when targeting the SubBytes operation and around 100 traces in the case of ARK (Figure 6a). For the full key, we see that we only need below 100 traces when targeting the SubBytes operation and around 1000 traces in the case of ARK (Figure 6b). Furthermore, taking a closer look at Figure 6a, we notice the attack using one trace is more effective for ARK than the S-BOX.

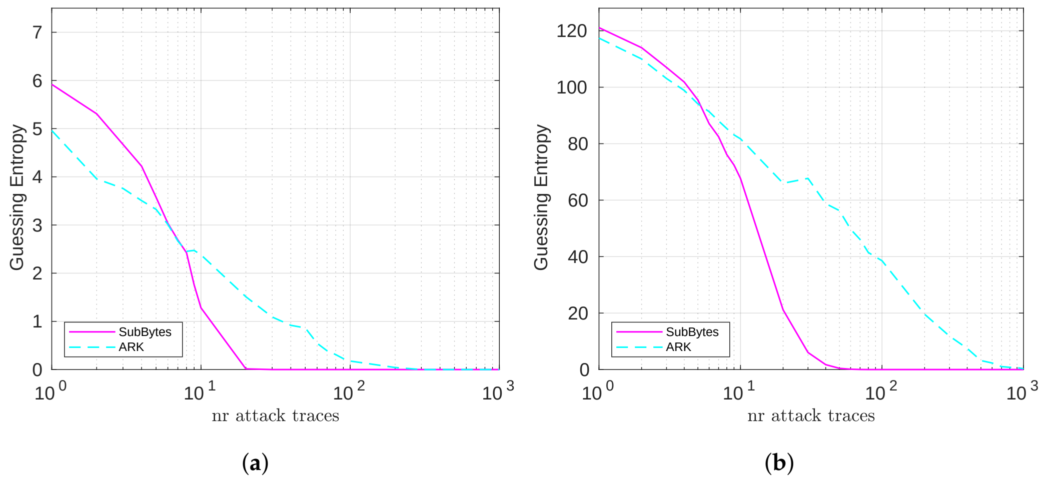


Figure 6. Template attack results (unprotected implementation). (**a**) GE on one byte. (**b**) Full-key evaluation.

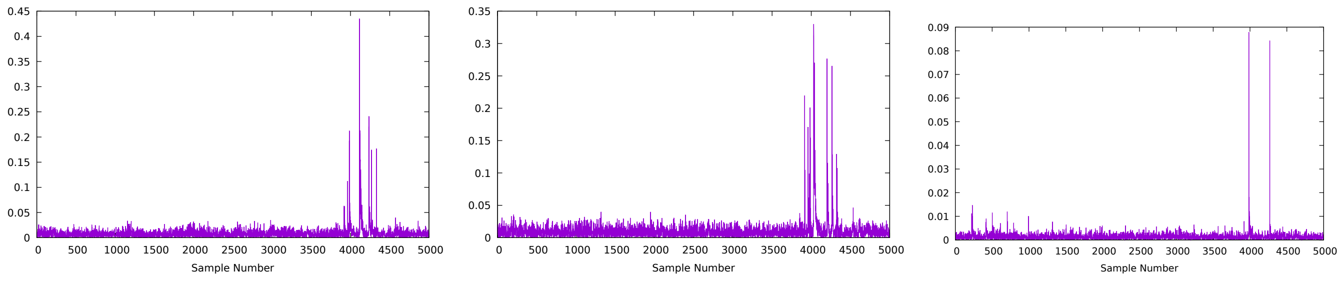
Both the CPA and Template Attack results show that both the ARK and the SubBytes can be good targets for conducting the attacks on the unmasked bitsliced implementation. Nevertheless, for complete key recovery, attacking the SubBytes operation requires undeniably less attack traces. This is mainly due to fact that the SubBytes operation is non-linear and helps discriminate the correct key much faster than with the linear ARK operation. Hence, while these results are expected for Template Attacks, which use the full information on the trace and intermediate variables, it may not have been obvious in the case of CPA, where exploiting only two bits of the SubBytes output for the Hamming weight has been enough to successfully extract the correct key, more efficiently than using eight bits but on the linear ARK operation.

5.2. Attacks on Masked Bitsliced Implementations of AES

The share splitting of the internal variables in masking bitsliced implementations of AES breaks the dependency between the data and the power traces, leaving first-order attacks less efficient. For second-order attacks, the traces were preprocessed using the differential approach (Equation (6)). For computational expenses reasons, we first set out to determine the sample points where the mask and the masked values were leaked after the first ARK and the first S-BOX operations. It was at these points that we later mounted the second-order attacks.

5.2.1. CPA Attack on Masked SubBytes (Targeting 2 Key Bits)

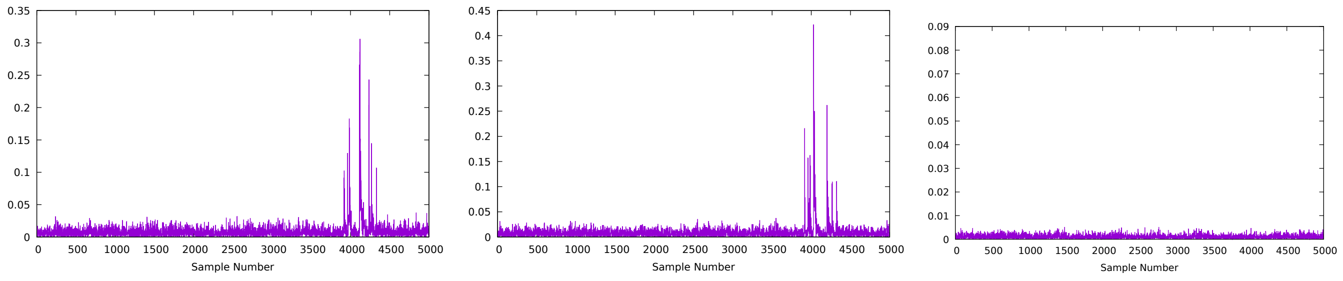
Regarding the first S-BOX operation, Figure 7a indicates the leakage interval (4000, 4200) for the updated mask, with the highest leakage points around 4128. A similar extensive leakage range is reported for the masked variables (Figure 7b), the highest leakage points being at $\approx$ 4040. We limited the preprocessing ranges to 50 points on each axis, and the preprocessing results outlined in this section refer to the ranges (4102, 4152) and (4020, 4070).



(**a**) Mask (**b**) Masked SubBytes (**c**) Unmasked SubBytes

Figure 7. Correlations for the correct key on masked SubBytes (Byte 1).

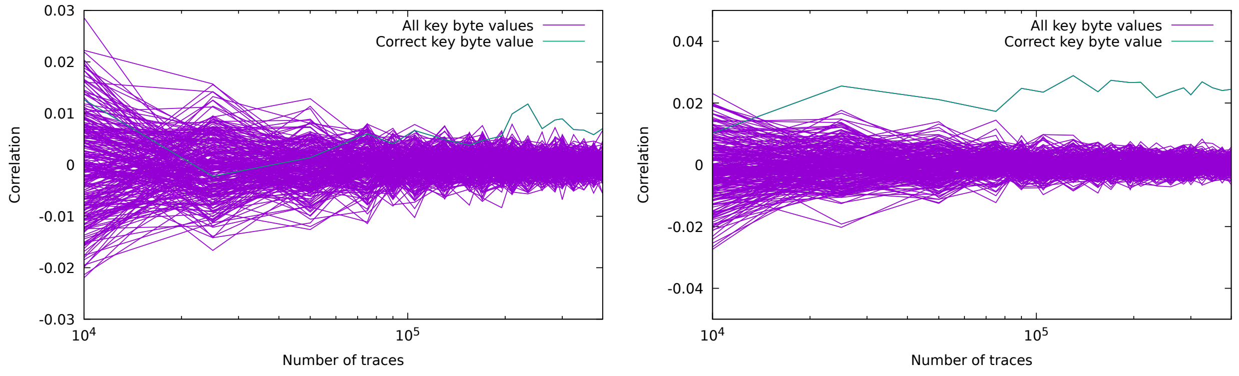
As seen in Figure 7c, the first byte reveals a strong correlation concerning the unmasked value. Nonetheless, the remaining 15 bytes reveal very weak correlation values (we show for example the second byte—Figure 8c, and we note that the subsequent bytes follow the same pattern), although the mask and the masked variables can be spotted in the traces. The images are aligned with the results obtained after applying the CPA attacks on each byte (Figures 9 and 10).



(**a**) Mask (**b**) Masked SubBytes (**c**) Unmasked SubBytes

Figure 8. Correlations for correct key on the masked SubBytes (Byte 2).

The masked implementation significantly increases the algorithm resistance against CPA attacks, from a few thousand traces to hundreds of thousands of traces. We outline in Figure 9a the first-order attack correlations of all values against the correct value, for the first byte, which is the easiest to recover. In contrast, the second-order attack requires much fewer traces to succeed (Figure 9b). The figures are generated for the best sample, respectively, the best pair of samples (i.e., the most leaking sample number(s) of the correct key byte value). For the first byte, for the first-order attack, the sample number is 4273 (see Figure 7c), while for the second-order attack, the pair of samples is (4030, 4114).



(**a**) First order attack (**b**) Second order attack

Figure 9. First S-BOX CPA attack on byte 1 (masked implementation).

However, as expected, Figures 8c, 10, and 11 further confirm that the second byte is very difficult to attack (either with a first-order attack or second-order attack), even when using 400,000 traces (as well as the other 14 bytes). The results suggest that the SubBytes target is rather difficult to attack, except for the very first byte.

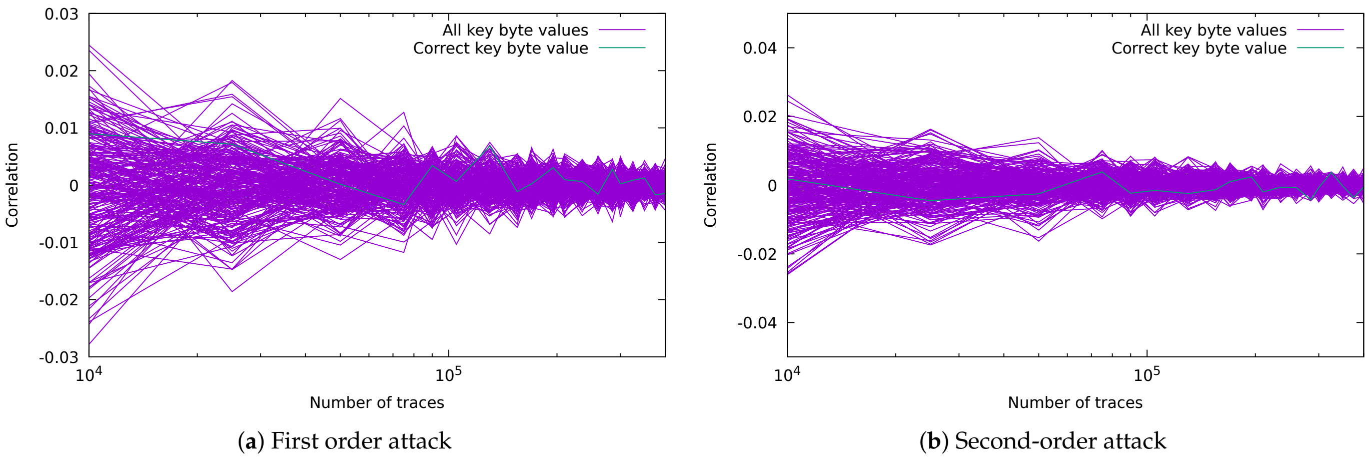


Figure 10. First SubBytes CPA Attack on Byte 2 (Masked implementation).

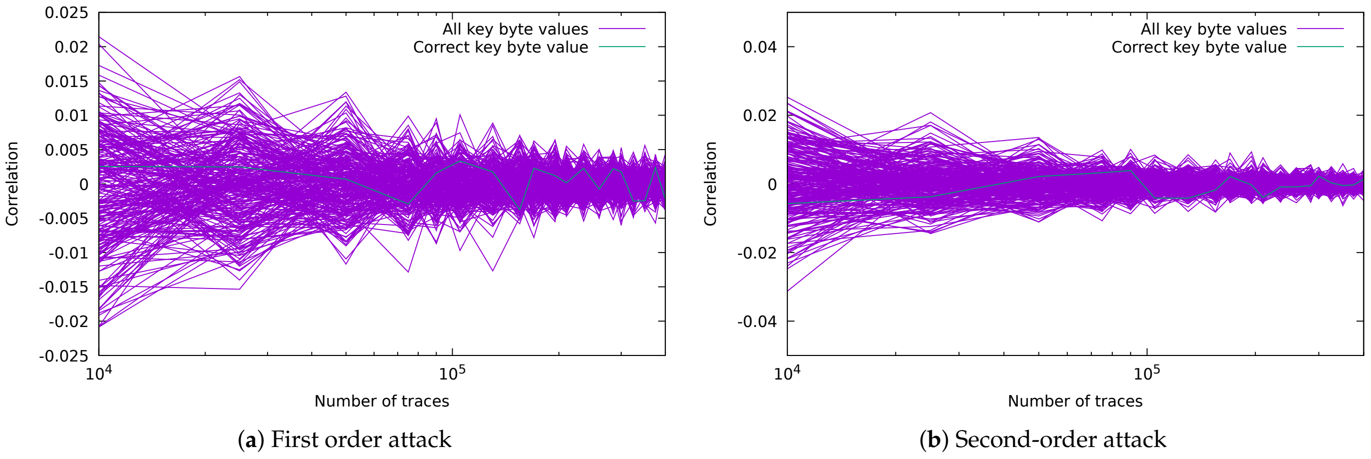


Figure 11. First SubBytes CPA attack on byte 3 (masked implementation).

5.2.2. CPA Attacks on Masked ARK (Targeting 8 Key Bits)

In the case of the first round key addition (ARK), both the mask and the masked variable leakage intervals reveal a pronounced leakage placed within the first 100 samples (see Figure 12a,b), confirming the acquisition setup explained in the previous section.

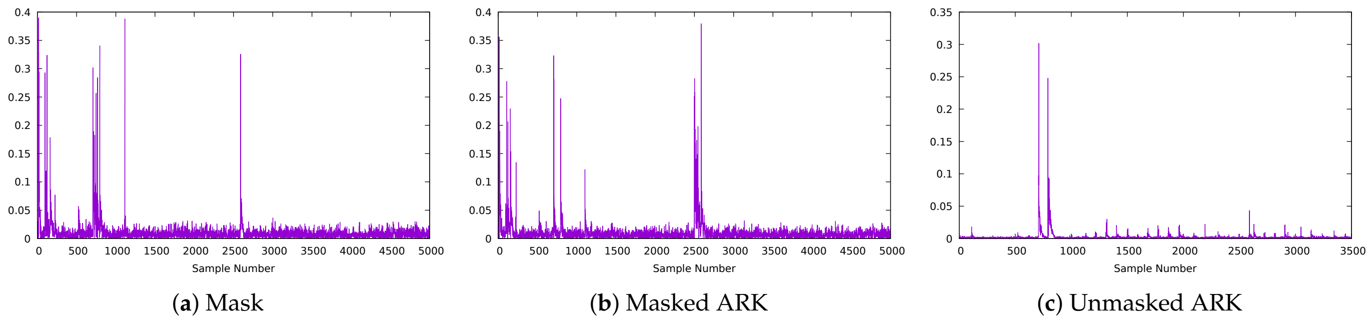
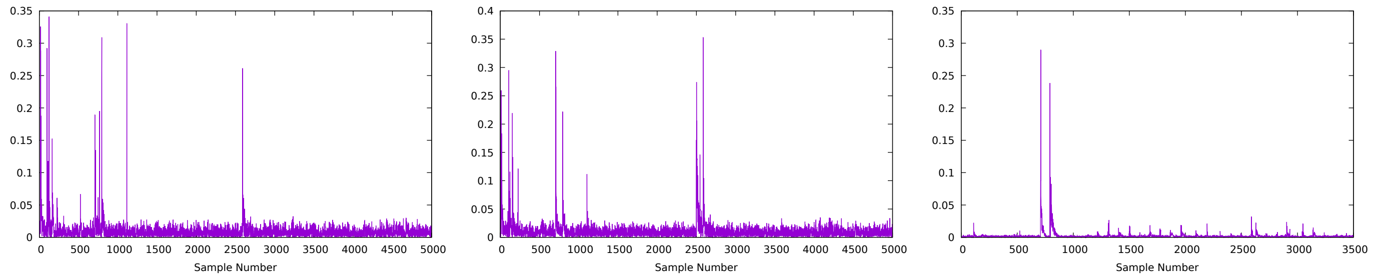


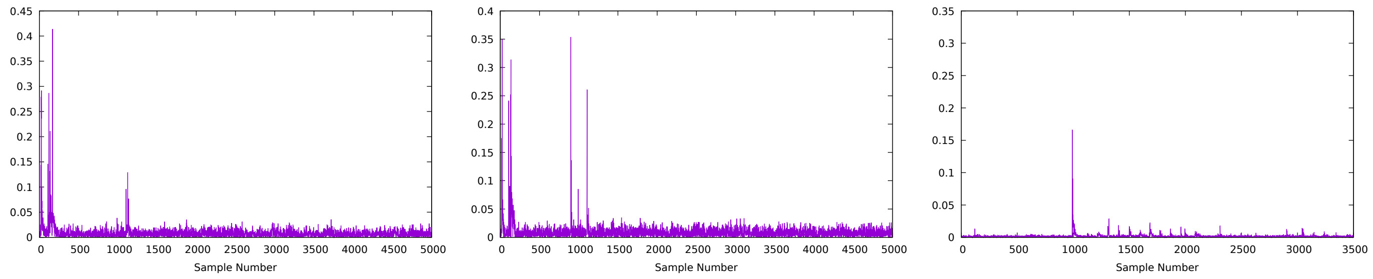
Figure 12. Correlations for the correct key on the masked ARK implementation (Byte 1).

Nevertheless, a quick glance at the following bytes correlations reveals a powerful leakage for the first four bytes (Figures 13c–16c). In fact, bytes 1 and 2 have the strongest correlations, which are double the ones recorded for bytes 3 and 4.



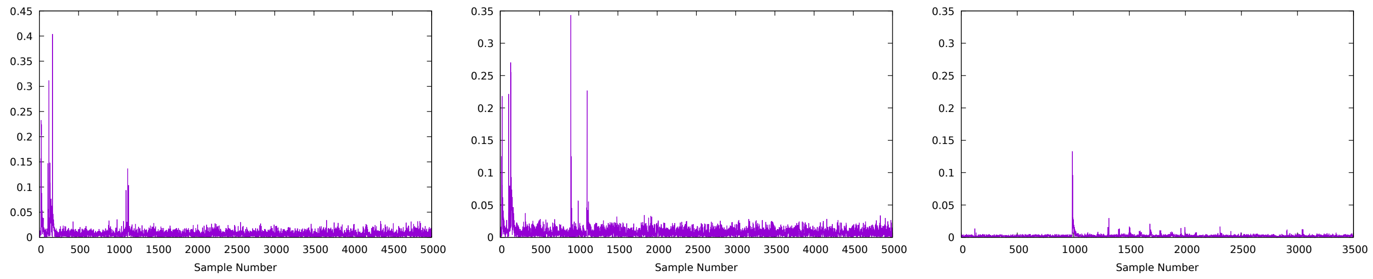
(**a**) Mask (**b**) Masked ARK (**c**) Unmasked ARK

Figure 13. Correlations for the correct key on the masked ARK implementation (Byte 2).



(**a**) Mask (**b**) Masked ARK (**c**) Unmasked ARK

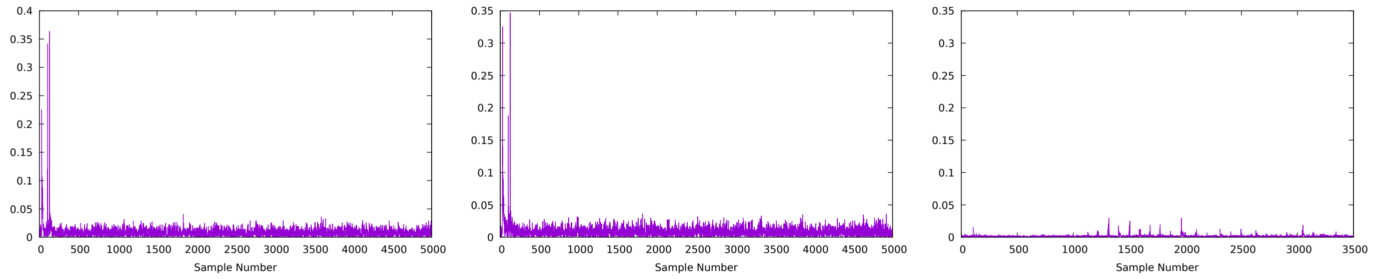
Figure 14. Correlations for the correct key on the masked ARK implementation (Byte 3).



(**a**) Mask (**b**) Masked ARK (**c**) Unmasked ARK

Figure 15. Correlations for the correct key on the masked ARK implementation (Byte 4).

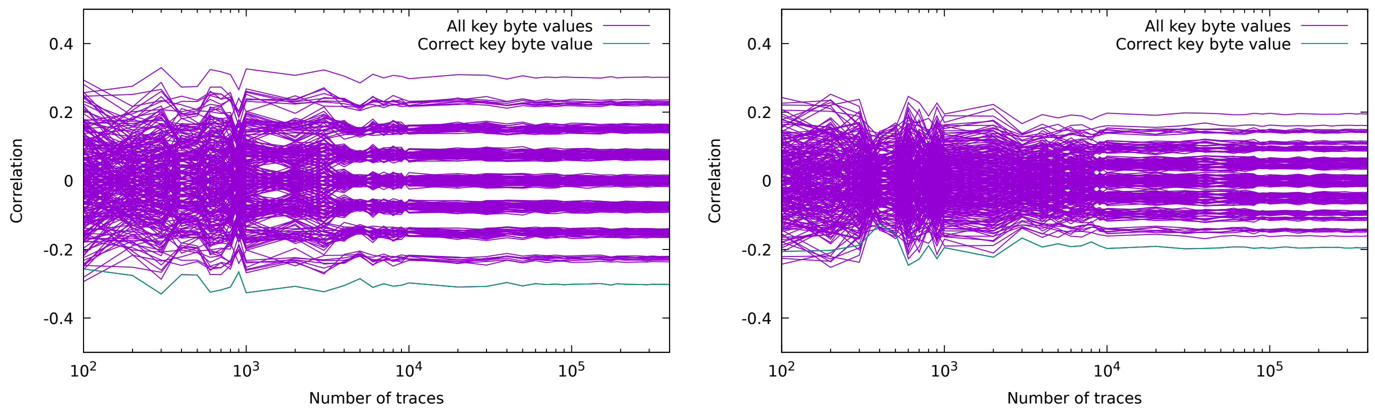
We do not show the remaining bytes; however, we should point out that the pattern recorded for the fifth byte (Figure 16) is applicable to the remaining 11 bytes. Thus, we can expect an increased difficulty in retrieving these bytes when applying the attacks.



(**a**) Mask (**b**) Masked ARK (**c**) Unmasked ARK

Figure 16. Correlations for correct key on masked ARK implementation (Byte 5).

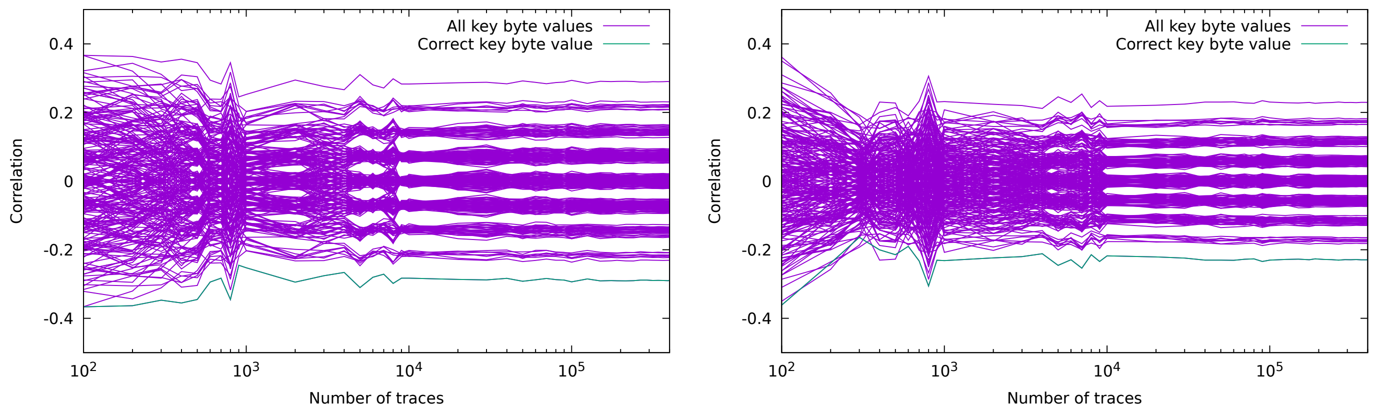
In Figure 17, we show the CPA attack results for the first eight key bits (in bitsliced format) in the first register, both for the first-order (no preprocessing) attack (left) and the second-order (absolute-difference) attack (right). We can see in this case that the first-order attack (Figure 17a) is successful with a very small number of traces (about 200), as confirmed by the high correlation in Figure 12c, while for the second-order attack, we need more than 500 traces, indicating that perhaps we were not able to find the best combination of points to obtain a better attack than in the first-order case. Furthermore, these results show that for this first register we could determine eight bits of the key (in bitsliced format) much faster than targeting the SubBytes operation, where we needed more than 200,000 traces.



(**a**) First-order attack (**b**) Second-order attack

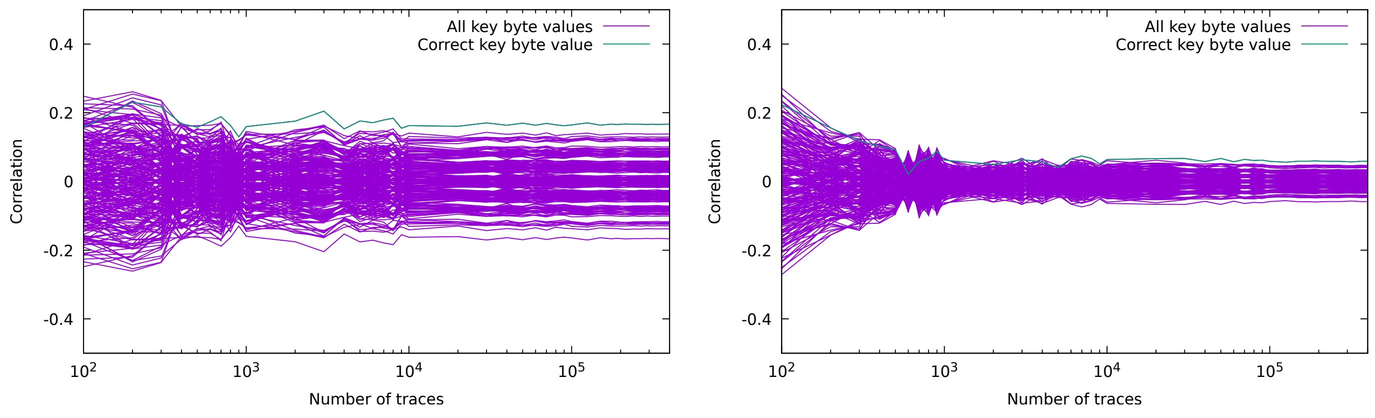
Figure 17. First ARK CPA attack on byte 1 (masked implementation).

Comparing Figure 18 to Figure 10 and Figure 19 to Figure 11, we notice the masked implementation ARK attack manages to recover some of the bytes, whereas the SubBytes attack fails. This observation stands for both the first-order and the second-order variants of the attacks. Finally, as an additional example, looking at the third byte (Figure 19), we notice the number of traces required for a successful attack increases (as expected from the correlation drop in Figure 14c).



(**a**) First-order attack (**b**) Second-order attack

Figure 18. First ARK CPA attack on byte 2 (masked implementation).

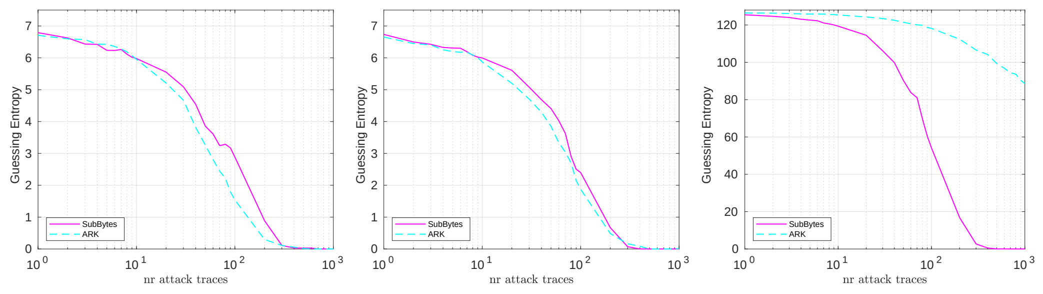


(**a**) First-order attack (**b**) Second-order attack

Figure 19. First ARK CPA attack on byte 3 (masked implementation).

5.2.3. Template Attacks

In Figure 20, we show the results of Template Attacks on the masked bitsliced implementation, without using any pre-processing; that is, performing the first-order attack. We can see (perhaps not too surprisingly, given the previous CPA results) that the Template Attacks can extract one key byte (on the first register) with about 200 traces when targeting either the SubBytes or ARK operations. For the full key, we need about 300 attack traces when targeting the SubBytes operation: we see in this case that the SubBytes operation is overall a better target than the ARK operation, due to its non-linearity.

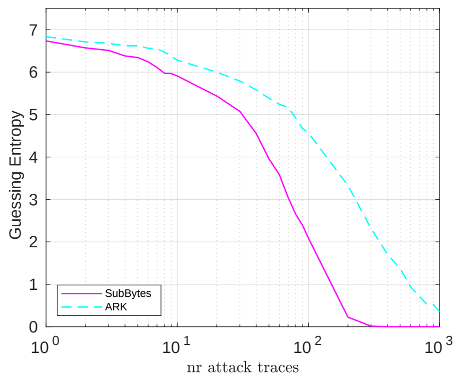


(**a**) GE on byte 1 (**b**) GE on byte 2 (**c**) Full-key evaluation

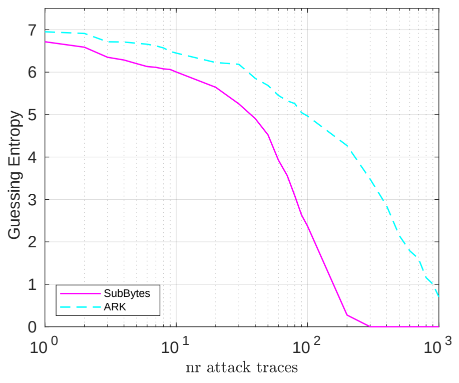
Figure 20. Template attack results (masked implementation).

In Figure 21, we take a closer look at the transitions occurring over the next three bytes. For the first two bytes, the ARK and the SubBytes results using Template Attacks are very close to each other (Figures 20a,b), although Figures 7c indicate much stronger correlation values for ARK than SubBytes (three times more). Unsurprisingly, the drop in the correlation values reported in Figures 14–16c brought forth a decrease in the chances to attack the following bytes, using the first ARK as the target.

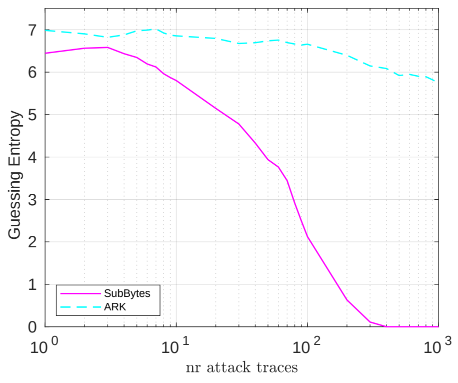
These results show that even a masked implementation of bitsliced AES is vulnerable to Template Attacks. This is mainly due to the fact that a considerable degree of correlation still exists between the leakage samples corresponding to the masked variables and the masks and (unfortunately) also between the power traces and the unmasked target value, as shown by the LDA eigenvectors of the attacks on both the SubBytes and ARK operations (see Figure 22). Both figures show that the LDA eigenvectors detect the peaks corresponding to the processing of the target variables, i.e., to the maximal peaks observed in Figures 7 and 12, respectively. The comparison between the CPA results and the Template Attack results reconfirms the higher efficiency of Template Attacks, explained by the extensive processing involved (considering the entire trace, performing a correlation between traces using LDA and PCA, including a profiling step).



(**a**) GE on byte 3



(**b**) GE on byte 4



(**c**) GE on byte 5

Figure 21. Template attack results for different bytes (masked implementation).

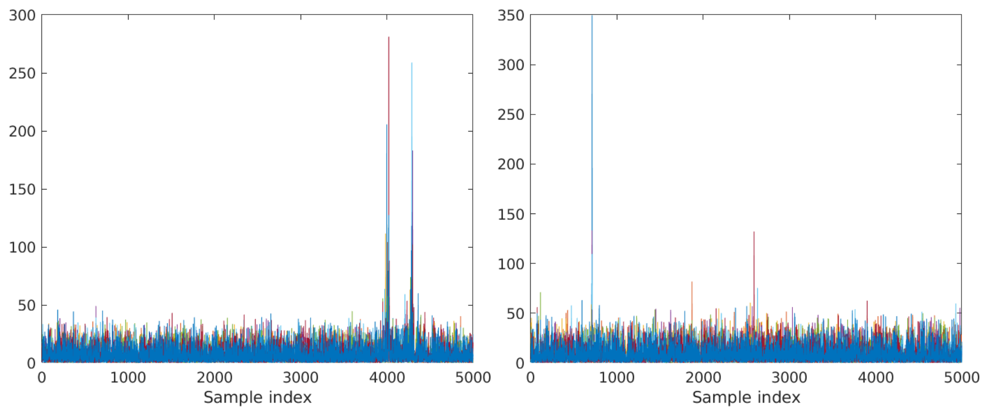


Figure 22. First eight LDA eigenvectors of Template Attacks (masked implementation). **Left**: SubBytes. **Right**: ARK.

Nevertheless, analysing both CPA and Template Attack results, we may conclude that the ARK operation can be more difficult to protect (at least for some registers) with first-order masking, but overall, the SubBytes operation allows a successful first-order attack on the full AES key even for the masked implementation. Hence, to increase the security of our target bitsliced AES implementation we might need higher-order masking and possibly higher noise, as noted by Standaert et al. [29].

6. Conclusions

In this paper, we presented an extensive analysis of CPA and Template Attacks on unprotected and protected versions of a bitsliced implementation of AES, targeting a 32-bit device through the ChipWhisperer side-channel attack platform.

We confirm the vulnerability of bitsliced implementations against power analysis attacks. In particular, our results show that Template attacks can determine the full AES key within 20 attack traces when targeting an implementation without countermeasures and within 300 attack traces when targeting first-order masking, without using any pre-processing of the traces.

Furthermore, we observed that although the SubBytes operation is overall the best target, there are a few cases where attacking the AddRoundKey operation is better: (i) Template attacks using a single trace; (ii) CPA against the masked implementation.

In summary, we can state that, depending on the chosen attack method (CPA or Template Attacks) and protection level of the implementation, we should choose between the SubBytes (CPA on unprotected implementations, Template Attacks on many attack traces) and AddRoundKey (CPA on protected implementations and Template Attacks using a single trace) operations.

Our study could be extended by developing more tailored CPA attacks against the masked implementation, exploiting the analytical equation of the bitsliced S-BOX implementation.

Author Contributions: The authors specific contributions to this paper are as follows: Conceptualization, A.R. and M.O.C.; Methodology, A.R. and M.O.C.; Software, A.R. and M.O.C.; Validation, A.R. and M.O.C.; Formal Analysis, M.O.C.; Investigation, A.R. and M.O.C.; Resources, M.O.C.; Data curation, A.R. and M.O.C.; Writing—original draft prepararion, A.R. and M.O.C.; Writing—review and editing, A.R. and M.O.C.; Visualisation, A.R. and M.O.C.; Supervision, M.O.C.; Project Administration, M.O.C.; Funding acquisition, M.O.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by a grant from the Romanian Ministry of Education and Research, CNCS–UEFISCDI, project number PN-III-P1-1.1-TE-2019-2245.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Rijmen, V.; Joan, D. Advanced Encryption Standard. Federal Information Processing Standards Publication 197. 2001; pp. 1–47. Available online: https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf (accessed on 3 March 2022).
2. Rebeiro, C.; Selvakumar, D.; Devi, A. Bitslice Implementation of AES. In Proceedings of the International Conference on Cryptology and Network Security, Suzhou, China, 8–10 December 2006; pp. 203–212.
3. Hajihassani, O.; Khalaj Monfared, S.; Khasteh, S.H.; Gorgin, S. Fast AES Implementation: A High-Throughput Bitsliced Approach. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *30*, 2211–2222. [CrossRef]
4. Könighofer, R. A Fast and Cache-Timing Resistant Implementation of the AES. In Proceedings of the Cryptographers' Track at the RSA Conference (CT-RSA 2008), San Francisco, CA, USA, 8–11 April 2008; pp. 187–202.
5. Mangard, S.; Oswald, E.; Popp, T. Power Analysis Attacks: Revealing the Secrets of Smart Cards; Springer: New York, NY, USA, 2007.
6. Agrawal, D.; Archambeault, B.; Rao, J.R.; Rohatgi, P. The EM Side—Channel(s). In *Cryptographic Hardware and Embedded Systems—CHES 2002*; Lecture Notes in Computer Science; Kaliski, B.S., Koç, Ç.K., Paar, C., Eds.; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2523, pp. 29–45.
7. Chari, S.; Jutla, C.; Rao, J.; Rohatgi, P. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *Advances in Cryptology—CRYPTO' 99. CRYPTO 1999. Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 1999; Volume 1666, pp. 398–412.
8. Grosso, V.; Leurent, G.; Standaert, F.X.; Varıcı, K. LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations. In *Fast Software Encryption. FSE 2014*; Lecture Notes in Computer Science; Cid, C., Rechberger, C., Eds.; Springer: Berlin/Heidelberg, Germany, 2015; Volume 8540, pp. 18–37.
9. Balasch, J.; Gierlichs, B.; Reparaz, O.; Verbauwhede, I. DPA, bitslicing and masking at 1 GHZ. In *Cryptographic Hardware and Embedded Systems—CHES 2015*; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9293, pp. 599–619.
10. Goudarzi, D.; Rivain, M. How Fast Can Higher-Order Masking Be in Software? In *Advances in Cryptology—EUROCRYPT 2017*; Lecture Notes in Computer Science; Coron, J.S., Nielsen, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2017; Volume 10210, pp. 567–597.
11. Longo, J.; De Mulder, E.; Page, D.; Tunstall, M. SoC It to EM: ElectroMagnetic Side-Channel Attacks on a Complex System-on-Chip. In *Cryptographic Hardware and Embedded Systems—CHES 2015*; Lecture Notes in Computer Science; Güneysu, T., Handschuh, H., Eds.; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9293, pp. 620–640.
12. de Groot, W.; Papagiannopoulos, K.; de La Piedra, A.; Schneider, E.; Batina, L. Bitsliced Masking and ARM: Friends or Foes? In *Lightweight Cryptography for Security and Privacy. LightSec 2016*; Lecture Notes in Computer Science; Bogdanov, A., Ed.; Springer: Cham, Switzerland, 2017; Volume 10098, pp. 91–109.
13. Journault, A.; Standaert, F.X. Very High Order Masking: Efficient Implementation and Security Evaluation. In *Cryptographic Hardware and Embedded Systems—CHES 2017*; Lecture Notes in Computer Science; Fischer, W., Homma, N., Eds.; Springer: Cham, Switzerland, 2017; Volume 10529, pp. 623–643.
14. Azouaoui, M.; Bronchain, O.; Grosso, V.; Papagiannopoulos, K.; Standaert, F.X. Bitslice Masking and Improved Shuffling: How and When to Mix Them in Software? 2021. Available online: hal.archives-ouvertes.fr (accessed on 22 February 2022).
15. Han, J.; Kim, Y.J.; Kim, S.J.; Sim, B.Y.; Han, D.G. Improved Correlation Power Analysis on Bitslice Block Ciphers. *IEEE Access* **2022**, *10*, 39387–39396. [CrossRef]
16. Brier, E.; Clavier, C.; Olivier, F. Correlation Power Analysis with a Leakage Model. In Proceedings of the Cryptographic Hardware and Embedded Systems, Cambridge, MA, USA, 11–13 August 2004; Volume 3156, pp. 16–29.

17. Chari, S.; Rao, J.R.; Rohatgi, P. Template Attacks. In *Cryptographic Hardware and Embedded Systems—CHES 2002*; Lecture Notes in Computer Science; Kaliski, B.S., Koç, Ç.K., Paar, C., Eds.; Springer: Berlin/Heidelberg, Germany, 2002; Volume 2523, pp. 13–28.
18. Gierlichs, B.; Batina, L.; Tuyls, P.; Preneel, B. Mutual information analysis. In Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems, Washington, DC, USA, 10–13 August 2008; pp. 426–442.
19. Choudary, M.; Kuhn, M. Efficient, Portable Template Attacks. *IEEE Trans. Inf. Forensics Secur.* **2017**, *13*, 490–501. [CrossRef]
20. Prouff, E.; Rivain, M.; Bevan, R. Statistical Analysis of Second Order Differential Power Analysis. *IACR Cryptol. ePrint Arch.* **2010**, *646*, 799–811. [CrossRef]
21. Durvaux, F.; Standaert, F.X.; Veyrat-Charvillon, N. How to Certify the Leakage of a Chip? In *Advances in Cryptology—EUROCRYPT 2014*; Lecture Notes in Computer Science; Nguyen, P.Q., Oswald, E., Eds.; Springer: Berlin/Heidelberg, Germany, 2014; Volume 8441, pp. 459–476.
22. Goubin, L.; Patarin, J. DES and Differential Power Analysis The "Duplication" Method. In *Cryptographic Hardware and Embedded Systems. CHES 1999*; Lecture Notes in Computer Science; Koç, Ç.K., Paar, C., Eds.; Springer: Berlin/Heidelberg, Germany, 1999; Volume 1717, pp. 158–172.
23. Schramm, K.; Paar, C. Higher Order Masking of the AES. In *Topics in Cryptology—CT-RSA 2006*; Lecture Notes in Computer Science; Pointcheval, D., Ed.; Springer: Berlin/Heidelberg, Germany, 2006; Volume 3860, pp. 208–225.
24. Coron, J.S.; Prouff, E.; Rivain, M.; Roche, T. Higher-order side channel security and mask refreshing. In Proceedings of the International Workshop on Fast Software Encryption, Singapore, 11–13 March 2013; Volume 8424, pp. 410–424.
25. Coron, J.S. Higher Order Masking of Look-Up Tables. In *Advances in Cryptology—EUROCRYPT 2014*; Lecture Notes in Computer Science; Nguyen, P.Q., Oswald, E., Eds.; Springer: Berlin/Heidelberg, Germany, 2014; Volume 8441, pp. 441–458.
26. Rivain, M.; Prouff, E. Provably Secure Higher-Order Masking of AES. In *Cryptographic Hardware and Embedded Systems, CHES 2010*; Lecture Notes in Computer Science; Mangard, S., Standaert, F.X., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6225, pp. 413–427.
27. Reparaz, O.; Bilgin, B.; Nikova, S.; Gierlichs, B.; Verbauwhede, I. Consolidating Masking Schemes. In *Advances in Cryptology—CRYPTO 2015*; Lecture Notes in Computer Science; Gennaro, R., Robshaw, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9215, pp. 764–783.
28. Messerges, T.S. Using Second-Order Power Analysis to Attack DPA Resistant Software. In *Cryptographic Hardware and Embedded Systems—CHES 2000*; Lecture Notes in Computer Science; Koç, Ç.K., Paar, C., Eds.; Springer: Berlin/Heidelberg, Germany, 2000; Volume 1965, pp. 238–251.
29. 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 *Advances in Cryptology—ASIACRYPT 2010*; Lecture Notes in Computer Science; Abe, M., Ed.; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6477, pp. 112–129.
30. Canright, D. *A Very Compact Rijndael S-Box*; Naval Postgraduate School, Department of Mathematics: Monterey CA, USA, 2004.
31. Trichina, E. Combinational Logic Design for AES SubByte Transformation on Masked Data. *IACR Cryptol. ePrint Arch.* **2003**, *236*, 1–13.
32. Schindler, W.; Lemke, K.; Paar, C. A Stochastic Model for Differential Side Channel Cryptanalysis. In *Cryptographic Hardware and Embedded Systems—CHES 2005*; Lecture Notes in Computer Science; Rao, J.R., Sunar, B., Eds.; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3659, pp. 30–46.
33. Choudary, M.O.; Kuhn, M.G. Efficient Stochastic Methods: Profiled Attacks Beyond 8 Bits. In *Smart Card Research and Advanced Applications. CARDIS 2014*; Lecture Notes in Computer Science; Joye, M., Moradi, A., Eds.; Springer: Cham, Switzerland, 2015; Volume 8968, pp. 85–103.
34. Standaert, F.X.; Malkin, T.G.; Yung, M. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, 30 May–3 June 2009; pp. 443–461.
35. Glowacz, C.; Grosso, V.; Poussier, R.; Schüth, J.; Standaert, F.X. Simpler and More Efficient Rank Estimation for Side-Channel Security Assessment. In *International Workshop on Fast Software Encryption. FSE 2015*; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9054, pp. 117–129.
36. Mangard, S.; Pramstaller, N.; Oswald, E. Successfully Attacking Masked AES Hardware Implementations. In Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems, Edinburgh, UK, 29 August–1 September 2005; pp. 157–171.
37. Oswald, E.; Mangard, S. Template Attacks on Masking—Resistance Is Futile. In Proceedings of the Cryptographers' Track at the RSA Conference (CT-RSA 2007), San Francisco, CA, USA, 5–9 February 2007; pp. 243–256.
38. Papagiannopoulos, K.; Veshchikov, N. Mind the Gap: Towards Secure 1st-Order Masking in Software. In *Constructive Side-Channel Analysis and Secure Design. COSADE 2017*; Lecture Notes in Computer Science; Guilley, S., Ed.; Springer: Cham, Switzerland, 2017; Volume 10348, pp. 282–297.
39. O'Flynn, C.; Chen, Z.D. Chipwhisperer: An open-source platform for hardware embedded security research. In Proceedings of the International Workshop on Constructive Side-Channel Analysis and Secure Design, Paris, France, 13–15 April 2014; pp. 243–260.
40. ChipWhispererLite 32-Bit API. Available online: https://chipwhisperer.readthedocs.io/en/latest/api.html (accessed on 10 January 2021).