Hybrid Homomorphic Encryption Resistance to Side-channel Attacks

Pierugo Pace[1,2], Hervé Pelletier[2], and Serge Vaudenay[1]

[1] EPFL, Lausanne, Switzerland
pierugo.pace@gmail.com, serge.vaudenay@epfl.ch
[2] Nagra Kudelski Group, Cheseaux-sur-Lausanne, Switzerland
pierugo.pace@gmail.com, herve.pelletier@nagra.com

Abstract. This work performs a side-channel analysis on the Hybrid Homomorphic Encryption cipher `Elisabeth-b4`. In particular, a Correlation Power Analysis allows to recover the 2048-bit key with 35,000 traces. Mounting template attacks or using Machine Learning decreases this number to 1,000. We then implement 2-share masking and shuffling, which completely eliminates the leakage measure – a Test Vector Leakage Assessment (TVLA) – and mitigates the Correlation Power Analysis and template attacks. Using a Divide and Conquer Deep Learning approach, we manage to bypass them but the number of required traces increases to 250,000.

Keywords: Hybrid homomorphic encryption · Side-channel analysis · Correlation power analysis · Template attacks · Deep machine learning · Countermeasures

1 Introduction

1.1 Context

We currently live in a world where the smallest Internet of Things (IoT) devices are able to collect useful data on the field at low cost and are used in combination with the largest cloud computers, that have the computing capabilities to process this data. This motivating process immediately raises privacy concerns if the data sent by the IoT contains sensitive information. A discovery from 2009 showed that *Fully Homomorphic Encryption* (FHE) could be used to tackle this problem [15]. FHE provides confidentiality and arbitrary computation on encrypted data, which allow cloud servers to process sensitive data that stays encrypted. While having practical realisations, Homomorphic Encryption is a costly operation that still remains a challenge for IoT devices.

Recent work thus came up with the Hybrid Homomorphic Encryption (HHE) framework, providing the same guarantees as FHE but shifting the overhead from the IoT device to the server [30]. The framework consists in the IoT device sending symmetrically-encrypted data and the server needing to *transcipher* it to get the equivalent data homomorphically encrypted and process it. The overhead induced by this transciphering heavily depends on the underlying symmetric cipher used by the parties. In particular, not all operations have the same cost and multiple symmetric schemes have been designed for HHE with different applications in mind [2,5,19,28]. In particular, as Deep Machine Learning inference is getting widely popularized as a cloud service, researchers designed `Elisabeth-4` [9] to optimize for this use case.

1.2 Motivation

When designing a cryptographic cipher, researchers always provide theoretical security guarantees that are mathematically sound but do not consider any implementation details, to be as generic as possible. In reality though, practical implementations on an electronic device often lead to a loss of security. Indeed, information about cryptographic secrets, like keys, can be obtained by observing the environment of the device. This process is called side-channel analysis and examples of the environment are timing [22], current consumption [23], or electromagnetic radiation [12]. It is very powerful but often requires some sort of physical access to the target device to observe the leakage.

`Elisabeth-b4` is a good candidate for the emerging HHE but IoT devices intrinsically suffer from physical access and it is thus essential to provide a secure implementation of the algorithms running on those devices. Our work aims to analyze the complexity of side-channel attacks on an IoT device and study the efficiency of countermeasures.

1.3 Related Work

One year after its release, an algebraic attack compromised `Elisabeth-4`, demonstrating its vulnerability to such methods [17]. In response, the original authors proposed multiple different alternatives, including an adjusted cipher named `Elisabeth-b4` [20], specifically designed to resist the attack in [17]. Our work, however, focuses on analyzing and attacking `Elisabeth-b4` using a distinct approach – side-channel analysis – and demonstrates that, despite the modifications, the newer version remains vulnerable to this class of attacks.

To the best of our knowledge, we have not seen any research done on the side-channel analysis of hybrid homomorphic ciphers and our work is innovative in that sense. Another hybrid homomorphic cipher called `FRAST` [8], beating `Elisabeth-b4` in terms of both latency and throughput, has been designed after the start of our work, but it still does not consider the threat of side-channel analysis in its design, and our work could probably be extended to this cipher as well. A differential fault attack was recently mounted on Elisabeth-4 [33] but they did not study side-channel analysis either.

1.4 Contributions

Our work provides results of side-channel attacks using electromagnetic radiation traces from `Elisabeth-b4` executions. We show how to recover the 2048-bit key with 35,000 traces with Correlation Power Analysis and with 1,000 traces using a template attack.

By integrating known protections like masking [6] and shuffling [32], the previously mounted Correlation Power Analysis becomes impossible and the template attack is much more complex. By using multiple Deep Learning models instead of templates, we still manage to recover the key by predicting the mask shares and shuffling. Though, it increases the number of traces on the target device to 250,000.

1.5 Outline

After a background on `Elisabeth-b4` and side-channel tools in Section 2, we describe our attack setup in Section 3 and show how we recover the entire key from the algorithm using different attacks in Section 4. In Section 5, we describe the countermeasures we implemented to increase the attacks' complexity and analyze their effectiveness in Section 6. We then present the use of Deep Learning as a new attack method in Section 7. Finally, we conclude our work in Section 8.

2 Background

2.1 Homomorphic Encryption

A homormophic encryption scheme includes a plaintext space, a ciphertext space, a pair of public-private key space and a circuit space. It provides the following algorithms:

– The key generation **KeyGen** $\left(1^{\lambda}\right)$ for security parameter λ returns a public-private key pair $\left(HK^{pub}, HK^{priv}\right)$ satisfying λ bits of security.
– The encryption **HEnc** $\left(p, HK^{pub}\right)$ for plaintext p and public key HK^{pub} returns the ciphertext c corresponding to the encryption of p with HK^{pub}.
– The decryption **HDec** $\left(c, HK^{priv}\right)$ for ciphertext c and private key HK^{priv} returns the plaintext p corresponding to the decryption of c with HK^{priv}. For correctness, $Pr\left(\mathbf{HDec}\left(\mathbf{HEnc}\left(p, HK^{pub}\right), HK^{priv}\right) = p\right) = 1$ must be satisfied.
– The evaluation **HEval** $\left(HK^{pub}, C, c_1, ..., c_n\right)$ returns the encryption of circuit C evaluated at the decrypted inputs, i.e., if $c_i = \mathbf{HEnc}\left(p_i, HK^{pub}\right)$, then $\mathbf{HEval}\left(HK^{pub}, C, c_1, ..., c_n\right) = \mathbf{HEnc}\left(C\left(p_i, ..., p_n\right), HK^{pub}\right)$.

The second and third provide confidentiality on the plaintexts and the last provides certain computation on encrypted data. If the circuit C can be arbitrary, then we call such a scheme a *Fully* Homomormphic Encryption (FHE) one and it provides arbitrary computation on encrypted data.

Hybrid Homomorphic Encrypytion (HHE) A HHE scheme consists of a FHE scheme with operations (**HEnc**, **HDec**, **HEval**) and keys $\left(HK^{pub}, HK^{priv}\right)$ combined with a symmetric encryption scheme with operations (**SEnc**, **SDec**) and key SK. The client sends $SK^{\mathtt{HE}} := \mathbf{HEnc}\left(SK, HK^{pub}\right)$ once at the very start and then sends the symmetric encryption $p^{\mathtt{SE}} := \mathbf{SEnc}\left(p, SK\right)$ of input p (instead of $p^{\mathtt{HE}} := \mathbf{HEnc}\left(p, HK^{pub}\right)$ for FHE). The server can then retrieve $p^{\mathtt{HE}}$ by homomorphically evaluating the symmetric decryption circuit $C^{\mathtt{SDec}}$ on inputs $p^{\mathtt{SE}^{\mathtt{HE}}} := \mathbf{HEnc}\left(p^{\mathtt{SE}}, HK^{pub}\right)$ and $SK^{\mathtt{HE}}$: $\mathbf{HEnc}\left(C^{\mathtt{SDec}}\left(p^{\mathtt{SE}}, SK\right), HK^{pub}\right) = p^{\mathtt{HE}}$, an operation called *transciphering.*

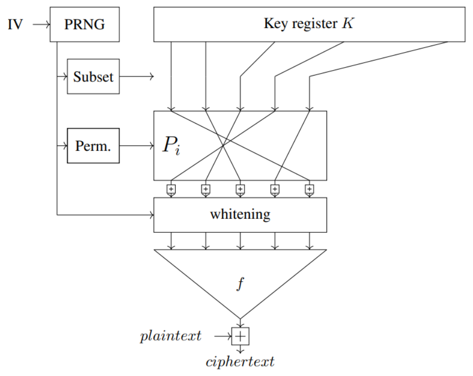
The HHE scheme provides the same guarantees as FHE but removes the overhead of performing any homomorphic encryptions on the client except one on SK, which is ideal for IoT devices. The challenge lies in finding a symmetric cipher whose decryption circuit is efficient homomorphically to reduce the overhead from transciphering as much as possible. `Elisabeth-4` and `Elisabeth-b4` are examples of such an optimized cipher.

2.2 `Elisabeth-4` & `Elisabeth-b4`

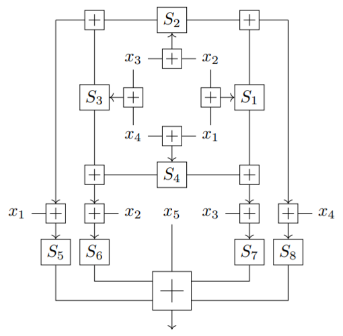
`Elisabeth-4` [9] is a symmetric stream cipher operating in $\mathbb{Z}_{16}$. It encrypts one plaintext element of $\mathbb{Z}_{16}$ (i.e. 4 bits) at a time by summing it with the keystream generated by the algorithm depicted in Figure 1a. The latter consists of the following two parts.

- We refer to the first part as **Random Whitened Subset (RWS)**. A public initialization vector (IV), is used as a seed to a foward secure PRNG that selects a random subset of 60 elements of $\mathbb{Z}_{16}$ from the 256 of the key, permutes them and adds a uniformly distributed whitening to each element. This results in a vector of 60 elements of $\mathbb{Z}_{16}$ that we will call *keyround* throughout the rest of the document.
- The keyround is then divided into 12 blocks of 5 elements called x_1, x_2, x_3, x_4, x_5 where each block is fed into the filtering function f depicted in Figure 1b. The outputs of the 12 independent evaluations of f are summed to get the keystream. All S-boxes are instances of Negacyclic Look-Up Tables (NLUTs), i.e. $S\left[i+8\right] = -S\left[i\right] \mod 16\ \forall i \in \{0, \ldots, 7\}$ for a NLUT S of length 16. Their first halves were generated taking the output of the SHA-256 hash of a string chosen by the authors.

The decryption algorithm consists in generating the same keystream using the initialization vector and subtracting it to the ciphertext to get the plaintext. `Elisabeth-4`'s key has a length of 256 elements of $\mathbb{Z}_{16}$, so 256 nibbles or 1,024 bits.
`Elisabeth-b4` [20] is designed to mitigate the attacks against `Elisabeth-4` discovered by [17]. The design is very similar to `Elisabeth-4`. The differences lie in the size of the key, which now has 512 elements, or 2,048 bits, and the size of the keyround which has a length of 98 elements. The latter is now divided into 14 blocks of length 7 that are fed to its adapted filtering function shown in Algorithm 1. The S-boxes are also NLUTs and their first halves were also generated taking the output of the SHA-256 hash of a string chosen by the authors. The initial copy of the inputs on line 3 of the algorithm corresponds to



(a) `Elisabeth-4`'s encryption algorithm



(b) `Elisabeth-4`'s filtering function

Fig. 1: `Elisabeth-4` description

their manipulation when copying them in the function's stack frame. The copy is necessary because the function would modify the inputs in place otherwise.

2.3 Side-channel Analysis Tools

Points of Interests Faced with the very large number of samples present in the traces, it is essential to identify and select the points that carry the useful information for our attacks. Numerous methods have been studied [13] to detect leakage regions, also called Points of Interests (PoIs). Examples of these include *Difference Of Means based method* [7] (DOM), *Signal-to-Noise Ratios based method* [25] (SNR), *Correlation Power Analysis based method* [25] (CPA), *Principal Component Analysis based method* [3] (PCA), and *Sum Of Squared pairwise T-differences based method* [16] (SOST). In our experiments, the most effective one was the last. Its formula reads:

$$\mathbf{f}[t] = \sum_{s_1 \neq s_2} \left(\frac{\mathbf{M_{s_1}}[t] - \mathbf{M_{s_2}}[t]}{\sqrt{\mathbf{S^2_{s_1}}[t]/|G_{s_1}| + \mathbf{S^2_{s_2}}[t]/|G_{s_2}|}} \right)^2 \qquad (1)$$

where s_1 and s_2 are all the possible critical values, G_s is the set of traces associated to s, $\mathbf{M_s} = \frac{1}{|G_s|} \sum_{\mathbf{t_i} \in G_s} \mathbf{t_i}$, $\mathbf{S^2_s} = \frac{1}{|G_s|} \sum_{\mathbf{t_i} \in G_s} (\mathbf{t_i} - \mathbf{M_s}) \odot (\mathbf{t_i} - \mathbf{M_s})$

2.4 Evaluation Metrics

Rank The rank of a subkey hypothesis is the position of the correct value in the corresponding vector of prediction probabilities sorted in decreasing order. If the rank is 0, then the correct value is predicted with highest probability and the prediction is correct. If the number of possible values the subkey can take is K, a trivial attack consisting in uniformly selecting one of the K values yields an expected rank of $(K-1)/2$.

Algorithm 1: `Elisabeth-b4`'s filtering function

```
input  : $(x_0, x_1, x_2, x_3, x_4, x_5, x_6) \in \mathbb{Z}_{16}^7$
output: $z \in \mathbb{Z}_{16}$
begin
    for $j$ in range(7) do
        $x'_j \leftarrow x_j$
    for $j$ in range(3) do
        $x'_{2j+1} \leftarrow x'_{2j+1} + x'_{2j}$
    for $j$ in range(6) do
        $y_j \leftarrow S_j\left(x'_j\right)$
    for $j$ in range(3) do
        $z_{2j} \leftarrow y_{2j+5 \mod 6} + y_{2j}$
        $z_{2j+1} \leftarrow y_{2j+4 \mod 6} + y_{2j+1}$
    for $j$ in range(6) do
        $z_j \leftarrow z_j + x'_{j+2 \mod 6}$
        $z_j \leftarrow S_{j+6}\left(z_j\right)$
    for $j$ in range(2) do
        $t_{3j} \leftarrow z_{3j} + z_{3j+1} + z_{3j+2}$
        $t_{3j+1} \leftarrow z_{3j+1} + z_{3j+3 \mod 6}$
        $t_{3j+2} \leftarrow z_{3j+2} + z_{3j+3 \mod 6} + y_{3j}$
    $x\_perm \leftarrow [5, 4, 3, 1, 0, 2]$
    for $j$ in range(6) do
        $t_j \leftarrow t_j + x'_{x\_perm[j]}$
    $z \leftarrow x'_6$
    for $j$ in range(6) do
        $u_j \leftarrow S_{j+12}\left(t_j\right)$
        $z \leftarrow z + u_j$
    return $z$
```

Success Rate The success rate is the ratio of subkeys correctly retrieved by the attacker. The same trivial attack for each element of the key yields an expected success rate of $1/K$. In the case of `Elisabeth-b4`, this is $1/16 = 6.25\%$.

3 Threat Model and Attack Setup

The very long key length – 2048 bits compared to the usual 256 bits – for this stream cipher represents a new challenge in terms of side-channel analysis. After considering a first threat model consisting of collecting traces directly on the target device, we then assume a different one in which the attacker can access a cloned component, where they can run any code. In both cases, the attacker gathers thousands of traces, each encrypted with the same key, while seeds (IVs) vary for each encryption.

We took inspiration from the authors' implementation of `Elisabeth-4` in Rust[1] to implement it in C. Adapting it for `Elisabeth-b4` was straight-forward. The PRNG was implemented by generating the byte stream from the output of the ChaCha20 cipher, using the seed as the cipher's key. The stream was produced by encrypting null plaintexts with null initialization vectors and subsequently used to produce numbers for the subsetting, permutation, and whitening operations of `Elisabeth-b4`'s key. We embedded the compiled code on an *Arduino DUE*, featuring an ARM Cortex-M3 32-bit processor operating at 84MHz. In addition to the previous work on stream ciphers in [24], we show that it is possible to attack 4-bit data words on a 32-bit processor.

By using the electromagnetic radiation with a dedicated probe instead of the power, it is possible to focus on the CPU part in charge of code execution only and eliminate other sources of noise (USB stack, analog part). Moreover, as shown in [26], glitches in CMOS circuitry have a high-frequency component which is more easily measurable in the EM field (a shunt resistor, for power, acts as a combination of resistance, inductance, and capacitance at high frequency). Finally, in modern processors, since nowadays dynamic power continues to dominate static power [29], it is in the interest of collecting the signal in the high-frequency domain with an electromagnetic probe.

To improve the quality of the leakage signal, we also thinned the chip by the back side, until it was about 100 µm long and moved the probe until we got the best signal. We finally select the sampling frequency of the oscilloscope at the value of 500 MS/s, by quantifying the collected values over 8 bits.

Indeed, measurements with a lower sampling rate showed that the amplitude and the number of leaks in the signal decreased. In contrast, a greater sampling rate produced new brief leaks, but this required the collection of a very large number of points per trace and led to a dataset which was too large to manage.

Each time we collected a trace, we executed the target 10 times and average them to reduce the noise. The collected trace corresponds to the algorithm executed to generate one keystream element of 4 bits as described in Section 2.2.

To evaluate the proposed methods, we collected four distinct sets of traces, referred to as Datasets A, B, C, and D, referenced throughout this document:

- **Dataset A:** Collected without countermeasures. It is divided into two populations of 125,000 traces each: one uses a fixed seed, while the other varies the seed. Both populations use the same fixed key.
- **Dataset B:** Collected without countermeasures. It contains 256,000 traces, each generated with a different seed but under the same fixed key.
- **Dataset C:** Collected with countermeasures and structured similarly to Dataset A, with two populations of 125,000 traces each: one with a fixed seed and the other with varying seeds. The seeds and keys are different from those in Dataset A.
- **Dataset D:** Collected with countermeasures. It comprises 1,000,000 traces, each generated with a different seed but under the same fixed key.

[1] Available on GitHub

The unprotected algorithm (Datasets A and B) runs in 160 µs, resulting in traces of 80,000 time points, while the protected one (Datasets C and D) runs in 400 µs, producing traces of 200,000 time points.

The machine mounting the CPA, template and ML-based attacks consists of an *AMD Ryzen Threadripper 7970X* processor with 256 GB RAM, using Python and NumPy, SciPy, scikit-learn. Our neural networks were trained on two *NVIDIA RTX 4500 Ada Generation* graphics cards and six *NVIDIA GeForce RTX 2080 Ti* graphics cards, using TensorFlow.

4 Attacks on an Unprotected Implementation

4.1 Trace Analysis

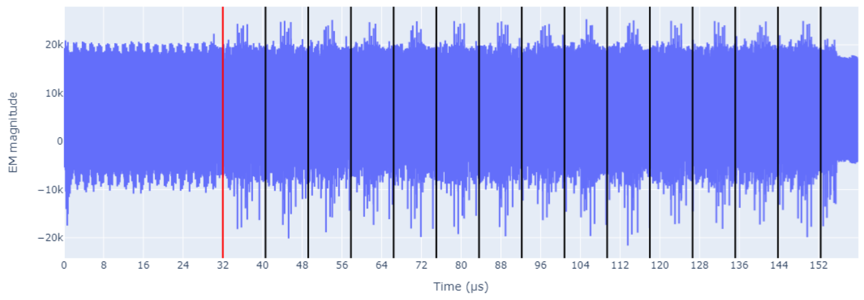


Fig. 2: Example of trace of `Elisabeth-b4`'s keystream generation

Figure 2 displays an example of a trace from Dataset B (unprotected). The scale of the electromagnetic field amplitude correspond to mV within a multiplicative factor: collected samples are encoded on 16 bits, ranging from −32,768 to 32,767. We can already identify the **RWS** from the start to the red line, as well as each of the 14 sequential executions – or *rounds* – of the filtering function, separated by the black lines.

4.2 Attack Strategy

The adversary's goal could either be to recover the plaintext or the symmetric key. The key is more often targeted since it can be used to recover not only the current plaintext but also future plaintexts. Also, as opposed to the plaintext, the same key is used in multiple messages, giving the adversary lots of traces to recover it. Therefore, the targeted critical values are the 512 key elements in our case. `Elisabeth-b4`'s keystream generation leaks in two distinctive parts:

Random Whitened Subset (RWS) The key elements are directly manipulated by the device to shuffle and whiten them. Even though the key is 512 elements long, only 98 are selected – and thus, leaked – during this procedure.
Filtering function The key elements are manipulated but are shuffled and whitened. Since both operations are deterministically derived from the public seed, they are reversible. The method to revert them is described below. Each of the 14 different executions of the filtering function leaks information about 7 elements of the key, totalling in the same 98 key elements as during the **RWS** from the previous paragraph.

We use $\mathbf{S_i} \in \{A \subseteq \{0, \ldots, 511\} \mid |A| = 98\}$ and $\mathbf{W_i} \in \mathbb{Z}_{16}^{98}$ to respectively refer to the sample/permutation and whitening during **RWS** of trace $\mathbf{t_i}$. To get prediction probabilities for the original key $\mathbf{K} \in \mathbb{Z}_{16}^{512}$ from prediction probabilities for the keyrounds $\mathbf{k_i} \in \mathbb{Z}_{16}^{98}$, we revert the random sample, shuffling and whitening as follows:

$$p\left(\mathbf{K}[j] = \dot{k} \;\middle|\; \mathbf{t_i}\right) = \begin{cases} p\left(\mathbf{k_i}\left[\mathbf{S_i^{-1}}[j]\right] = \dot{k} + \mathbf{W_i}\left[\mathbf{S_i^{-1}}[j]\right] \mod 16 \;\middle|\; \mathbf{t_i}\right), & \text{if } j \in \mathbf{S_i}. \\ \frac{1}{16}, & \text{otherwise.} \end{cases}$$

Indeed, the key element is uniformly distributed if it has not been sampled (i.e., $j \notin \mathbf{S_i}$), as no information can be gathered from the trace. Also, note that $p(\mathbf{k_i}[\cdot] = \cdot \mid \mathbf{t_i}) \propto p(\mathbf{t_i} \mid \mathbf{k_i}[\cdot] = \cdot)$ because $p(\mathbf{k_i}[\cdot] = \cdot) = 1/16$, a useful property to compute probabilities of keyround elements from the templates. In both cases, and even if we target both parts simultaneously, only 98 out of the 512 elements of the key are used. This implies that, to recover the entire key, its 512 elements must appear at least once. A strict minimum of $\lceil \frac{512}{98} \rceil = 6$ traces are thus needed by the pigeonhole principle, but 33 in expectation.[2]

4.3 Test Vector Leakage Assessment (TVLA)

We perform a TVLA on Dataset A. It is shown in Figure 3, zoomed on the last round of the algorithm – the other rounds are similar.

The red horizontal lines highlight a t-value of 4.5, which is the threshold corresponding to a confidence level of 99.999% in our setting, with nearly 250,000 degrees of freedom at each time point. By disassembling the code and applying a Pearson correlation calculation between the manipulated data and the trace, we identify the leakage points associated with different colors in Figure 3. These colors correspond to operations that we found leaked the most, such as the output of the first round of S-Boxes for the red regions (Algorithm 1, line 7).

4.4 Correlation Power Analysis

Previous results showed that the **Hamming Weight** was a good model to assess leakage. This model was thus retained for the attack. We targeted non-linear operations, i.e. the S-boxes. S-boxes S_0, S_2, and S_4 caught our attention

[2] See Appendix A for more details.

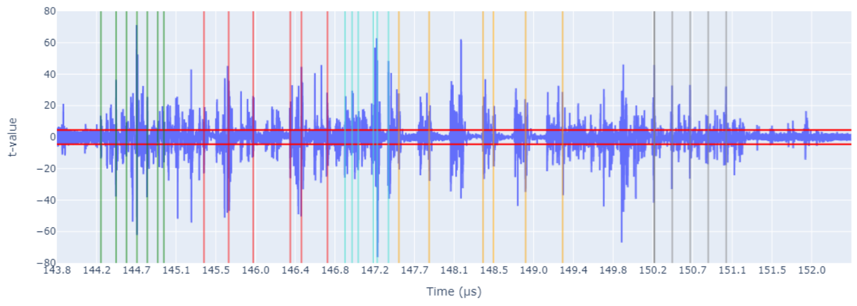


Fig. 3: TVLA on last round of `Elisabeth-b4`

because they depend only on a single keyround element which corresponds to a search space of 4 bits. Other S-boxes depend on up to six elements, corresponding to a larger search space of 24 bits. Examples of correlations with the correct key are shown in Figure 4. This step was performed on Dataset B.

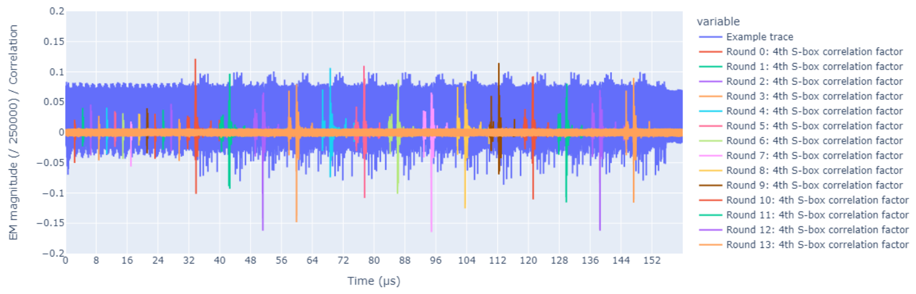


Fig. 4: Correlation factor of the Hamming Weight of the output of S-box S_4

An example of a graph showing the maximum correlation factor throughout the trace for different hypotheses of a keyround element, targeting the output of S-box S_4 is shown in Figure 5. An interesting observation is that there are always two distinctive correlation peaks: one for the correct key hypothesis $\dot{k}$ and one for $\dot{k}+8 \mod 16$. This increases the number of traces to distinguish the two and thus increases the complexity of the attack. The phenomenon is explained because the S-boxes are NLUTs, i.e., $S_4\left[k+8 \mod 16\right] = -S_4\left[k\right] \mod 16$ and the Hamming Weights of the respective outputs are thus heavily correlated.

We only used the output of S_4 because it gave the best results. We only considered the last two rounds for simplicity. Concretely, we correlate the second

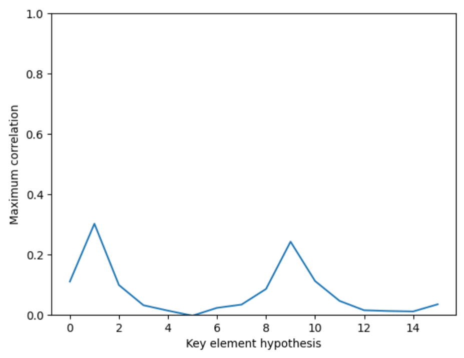


Fig. 5: Example of maximum correlation factors across the entire trace for different hypotheses. Correct key element is 1.

to last round's traces with values $\mathbf{HW}(S_4[x])$ and the last round's traces with values $\mathbf{HW}(S_4[x'])$ with $x, x' \in \mathbb{Z}_{16}$ hypotheses for $\mathbf{K}[\mathbf{S_i}[88]] + \mathbf{W_i}[88] \mod 16$ and $\mathbf{K}[\mathbf{S_i}[95]] + \mathbf{W_i}[95] \mod 16$ respectively. By removing the whitening, we thus have a hypothesis for a key element.

As two different elements of the key are leaked in each trace ($\mathbf{K}[\mathbf{S_i}[88]]$ and $\mathbf{K}[\mathbf{S_i}[95]]$), we collected 256,000 traces so that we have about 1,000 traces for each key element (Dataset B) and successfully recovered the entire key. We expect that the number of traces required is divided by 7 when targeting all 14 rounds instead of the last two, involving about 35,000 traces. This number could probably be even further reduced by also exploiting more information than simply the output of S_4, such as other S-boxes or the **HW** of the inputs at the beginning of the rounds. We observe that the attack is feasible with a model leakage based on 4 bits only (the **HW** of an S-box) even though the processor is 32-bit. In our case, it is difficult to find a model on more bits, since every operation is in $\mathbb{Z}_{16}$, but doing so could also reduce the complexity of the attack.

4.5 Template Attack

The objective of our template attack is to predict the values of the 98 keyround elements. Compared to other PoI detection methods, SOST gave us the clearest ones and were in accordance with the previously computed TVLA. We thus compute 98 SOSTs, one for each of the 98 targeted keyround elements. Examples of such SOSTs are shown in Figure 6. We notice in particular that keyround values leak once during **RWS** and once during its respective round, validating the claim in 4.2.

We build $98 \cdot 16 = 1{,}568$ templates during the profiling phase – one for every possible value of every keyround element. Each template is defined over $N = 40$ points from the trace, chosen as the largest N values of the SOST corresponding to the target element. The precise values of N was 5-fold cross-validated based on the success rate.

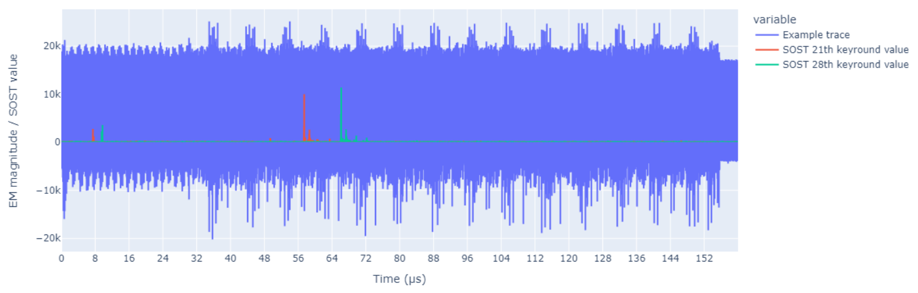


Fig. 6: Example of SOSTs on top of a trace

During the extraction phase, we compute the 16 probabilities of each key-round value for each of the 98 elements, follow Section 4.2 to revert the **RWS** and select the Maximum-Likelihood Estimator for every j:

$$\hat{\mathbf{K}}[j] = \underset{\dot{k}\in\mathbb{Z}_{16}}{\arg\max} \prod_{\mathbf{t_i}} p\left(\mathbf{t_i} \;\middle|\; \mathbf{K}[j] = \dot{k}\right) = \underset{\dot{k}\in\mathbb{Z}_{16}}{\arg\max} \sum_{\mathbf{t_i}} \log p\left(\mathbf{K}[j] = \dot{k} \;\middle|\; \mathbf{t_i}\right) \quad (2)$$

where we used that $p\left(\mathbf{K}[j] = \dot{k}\right) = 1/16\ \forall \dot{k}$ and applied the logarithm to tackle numerical instabilities that arise when multiplying small probabilities together. This method preserves the order of probabilities when selecting the maximum due to the logarithm's monotonic increase while staying numerically stable.

We also used Dataset B for this attack. When using 100,000 profiling traces, this attack gives an average accuracy of 13.33 ± 0.20 % for keyround elements. After applying the Maximum-Likelihood Estimator, we recover the complete key with 1,000 extraction traces. The average rank of a key element is 0.005 at 1,000 extraction traces. To practically compute this number, we computed each key element's average rank across 100 sets of size 1,000 sampled from 10,000 traces not seen during the training. We then averaged the ranks from all elements.

4.6 Machine Learning-Based Attack

Using Machine Learning is almost identical as with templates: For each one of the 98 targeted elements, we first select a subset of the trace using SOST, and then train an ML model predicting its value between 0 and 15. Each model outputs probabilities for each possible keyround value, we revert the **RWS** following Section 4.2, add all the logarithms of probabilities and choose the highest one as the final prediction. We tried different configurations of ML models:
SVMs [10]: We rapidly abandoned the use of SVMs due to their long training times, independently of the kernel or of the regularization parameter.
Random Forests [4]: Using 5-fold cross-validation, the best choice of hyperparameters were to choose a subset of length $N = 40$ from the SOST, train 300 trees, let them grow indefinitely but have at least 6 samples left at each leaf.

Gradient Boosting [14]: Using 5-fold cross-validation, the best choice of hyperparameters were to choose a subset of length $N = 60$ from the SOST, iterate for at most 300 iterations with the multinomial deviance loss, let each tree grow indefinitely but have at least 6 samples left at each leaf. This method performed best consistently.

Also using Dataset B and choosing Gradient Boosting with 100,000 profiling traces results in 11.14 ± 0.18 % accuracy for keyround elements. We recover the entire key with 1,500 extraction traces, performing slightly worse than template attacks. We did not try any more complex models as our results were already satisfactory. All our attack results are summarized in Table 1. The *Baseline* row indicates the accuracy one would get with a trivial estimator that would select a uniform random value for every keyround element.

Table 1: Overall attack results without countermeasures

Method	# Traces		Test element accuracy (%)	Success rate (%)
	Profiling	Extraction		
CPA	0	~35,000	-	100
Template	100,000	1,000	13.33 ± 0.20	100
Random Forest	100,000	2,500	10.93 ± 0.17	100
Gradient Boosting	100,000	1,500	11.14 ± 0.18	100
Baseline	0	0	6.25 ± 0.00	6.25

5 Countermeasures Implementation

5.1 Masking

Masking is a countermeasure that has already been proven to exponentially increase the attack complexity with respect to the number of shares used [6]. To protect a sensitive value s with M-share masking, we uniformly sample M shares such that they all sum to s. This leads to the masking vector $\mathbf{m} \in \mathbb{Z}_{16}^{M}$, where $\sum_{i=0}^{M-1} \mathbf{m}[i] \mod 16 = s$. Then, the algorithm uses $\mathbf{m}$ instead of s and the value of s is never manipulated directly.

Implementing this countermeasure means redefining all operations that use sensitive values into ones that use masking vectors. In our case, the sensitive values are the 98 keyround elements and the operations that use them are additions and S-box evaluations.

The addition of two masked values $\mathbf{m_1}$ and $\mathbf{m_2}$ that respectively correspond to sensitive s_1 and s_2 is $\mathbf{m_3}$, where $\mathbf{m_3}[i] = \mathbf{m_1}[i] + \mathbf{m_2}[i] \mod 16$, such that the corresponding s_3 is $s_1 + s_2 \mod 16$. The time complexity of the addition is multiplied by M.

To evaluate S-box $\mathbf{S} \in \mathbb{Z}_{16}^{16}$ on sensitive s, we pre-compute a table $\mathbf{S}'$ corresponding to the evaluation of S-box $\mathbf{S}$ for every possible share combination, inspired by the implementation of [1]. What follows applies only for $M = 2$, protecting against first-order attacks. For higher order attacks where $M > 2$, we refer the reader to [31]. The new table is $\mathbf{S}' \in \mathbb{Z}_{16}^{16\times16}$, where $\mathbf{S}'[i_1][i_2] = \mathbf{S}[i_1 + i_2 \mod 16] - i_1 \mod 16$ and we store $\mathbf{S}'$ in memory instead of $\mathbf{S}$. When we want to evaluate it with masked $\mathbf{m}$, we compute $\left[\mathbf{m}[0]\ \mathbf{S}'[\mathbf{m}[0]][\mathbf{m}[1]]\right] = \left[\mathbf{m}[0]\ \mathbf{S}[\mathbf{m}[0] + \mathbf{m}[1] \mod 16] - \mathbf{m}[0] \mod 16\right]$, a vector whose sum is the expected $\mathbf{S}[s]$. The memory complexity is multiplied by 16 to store $\mathbf{S}'$.

We chose $M = 2$ for performance reasons. An important improvement we realized is that we could pack the 2 shares inside the same 32-bit word in memory since they are 4 bits long, as elements of $\mathbb{Z}_{16}$. This has the advantage of being able to add two masked values in a single addition instead of two and with less memory fetches. Concretely, our packed number is a single word in memory instead of a vector: $m' = \left(2^8 \cdot \mathbf{m}[0]\right) \vee \mathbf{m}[1]$. We use the notation $[\cdot : \cdot]$ for bit indexing. We thus have $m'[11:8] = \mathbf{m}[0]$ and $m'[3:0] = \mathbf{m}[1]$. Additions are defined as (the $\wedge$ is used here to prevent overflows):

$$\texttt{MaskedAdd}\left(m'_1, m'_2\right) := \left(m'_1 + m'_2\right) \wedge \texttt{0b000011110000111}, \tag{3}$$

and S-boxes evaluations as:

$$\texttt{MaskedSBox}\left(\mathbf{S}', m'\right) := \left(2^8 \cdot m'[11:8]\right) \vee \mathbf{S}'[m'[11:8]][m'[3:0]]. \tag{4}$$

We mask every keyround element. Since there is also some shuffling (see 5.2), the masks do not appear in the same order during **RWS** and during the rounds. We will thus refer to them respectively as $\mathbf{m}_\mathbf{j}^\mathbf{rws} \in \mathbb{Z}_{16}^2, j \in \{0, ..., 97\}$ and $\mathbf{m}_\mathbf{r,b}^\mathbf{rounds} \in \mathbb{Z}_{16}^2, (r, b) \in \{0, ..., 13\} \times \{0, ..., 6\}$ (the b^{th} mask of the r^{th} round), in order where they appear in the trace. There is thus a total of 392 variables for mask shares, all ranging between 0 and 15.

5.2 Shuffling

Shuffling is another countermeasure that was thoroughly studied previously [32] whose purpose is to randomize the execution order of independent operations. This does not alter the outcome of the operations and an attacker now cannot directly know which operation is executed at which point in time. We implemented it with Random Start Indices (RSI) [32], i.e., the operations are executed in order but starting from a random index, for performance reasons. We shuffled the following for `Elisabeth-b4`:

- The order in which the key elements are manipulated during **RWS** is indifferent. We will refer to this RSI as $p_{rws} \in \mathbb{Z}_{98}$.
- The order in which the 14 executions of the filtering function are executed is indifferent. We will refer to this RSI as $p_{round} \in \mathbb{Z}_{14}$.
- For each round $r \in \{0, \ldots, 13\}$, the order in which the block inputs are copied is indifferent (Line 3). We will refer to the corresponding r^{th} round's RSI as $p_{block_r} \in \mathbb{Z}_7$.

– For each of the 7 remaining loops of the 14 rounds, the order in which they are executed is indifferent. The RSIs take values respectively from $\mathbb{Z}_3$, $\mathbb{Z}_6$, $\mathbb{Z}_3$, $\mathbb{Z}_6$, $\mathbb{Z}_2$, $\mathbb{Z}_6$, $\mathbb{Z}_6$ (see Algorithm 1).

Our shuffling countermeasure thus contains 114 different RSIs, but we target only 16 of them $(p_{rws}, p_{round}, p_{block_r})$ to retrieve the key.

The pseudo-code containing this countermeasure as well as the 2-share masking can be found in Algorithms 2 and 3. Note: we use the vector notation for masks, but our implementation pack the shares in the same 32-bit word in memory, as described in 5.1.

The countermeasures bring a `x2.5` time overhead, as traces now last 400 µs.

Algorithm 2: Masked and shuffled `Elisabeth-b4`'s keystream generation

```
input : Masked key $(\mathbf{K_0}, \dots, \mathbf{K_{511}}) \in \mathbb{Z}_{16}^{512\times 2}$, Sampling/Permutation
        $\mathbf{S_i} \in \{A \subseteq \{0, \dots, 511\} \mid |A| = 98\}$ and Whitening $\mathbf{W_i} \in \mathbb{Z}_{16}^{98}$
output: Masked keystream $\mathbf{z} \in \mathbb{Z}_{16}^{2}$
1  Function Perm($max$):
2      $rsi \leftarrow rand() \mod max$
3      return
         $[rsi \mod max, rsi + 1 \mod max, \dots, rsi + (max - 1) \mod max]$
4  begin
5      // Masked RWS:
6      for $j$ in Perm(98) do // Targeted RSI $p_{rws}$
7          $\mathbf{m_j} \leftarrow$ MaskedAdd $\left(\mathbf{K}_{\mathbf{S_i}[\mathbf{j}]}, \mathbf{W_i}[j]\right)$ // Targeted masks $\mathbf{m_j^{rws}}$
8      // Masked rounds:
9      $\mathbf{z}[0] \leftarrow rand() \mod 16$
10     $\mathbf{z}[1] \leftarrow 0 - \mathbf{z}[0] \mod 16$
11     for $j$ in Perm(14) do // Targeted RSI $p_{round}$
12         $\mathbf{z} \leftarrow$ MaskedAdd $(\mathbf{z}, \textbf{Algorithm3}(\mathbf{m_{7*j}}, \dots, \mathbf{m_{7*j+6}}))$
13     return $\mathbf{z}$
```

6 Attacks on a Protected Implementation

6.1 TVLA and CPA

To assess the impact of these countermeasures, we enabled them and mounted the same attacks from Section 4. As a first step, we redo a TVLA, this time on Dataset C. It is shown in Figure 7, zoomed on the last round of the algorithm – the other rounds are similar. We observe that the first-order leakage of the key elements has in this case completely disappeared. This is explained because random mask shares are manipulated instead of the key elements. The leakage

Algorithm 3: Masked and shuffled `Elisabeth-b4`'s filtering function

```
input  : Mask shares $(\mathbf{m_0}, \mathbf{m_1}, \mathbf{m_2}, \mathbf{m_3}, \mathbf{m_4}, \mathbf{m_5}, \mathbf{m_6}) \in \mathbb{Z}_{16}^{7\times 2}$
output: Mask shares $\mathbf{z} \in \mathbb{Z}_{16}^{2}$
1  begin
2      for $j$ in Perm(7) do // Targeted RSI $p_{block_r}$
3          $\mathbf{m'_j} \leftarrow \mathbf{m_j}$ // Targeted masks $\mathbf{m^{rounds}_{r,b}}$
4      for $j$ in Perm(3) do
5          $\mathbf{m'_{2j+1}} \leftarrow$ MaskedAdd$(\mathbf{m'_{2j+1}}, \mathbf{m'_{2j}})$
6      for $j$ in Perm(6) do
7          $\mathbf{y_j} \leftarrow$ MaskedSBox$(\mathbf{S'_j}, \mathbf{m'_j})$
8      for $j$ in Perm(3) do
9          $\mathbf{z_{2j}} \leftarrow$ MaskedAdd$(\mathbf{y_{2j+5 \bmod 6}}, \mathbf{y_{2j}})$
10         $\mathbf{z_{2j+1}} \leftarrow$ MaskedAdd$(\mathbf{y_{2j+4 \bmod 6}}, \mathbf{y_{2j+1}})$
11     for $j$ in Perm(6) do
12         $\mathbf{z_j} \leftarrow$ MaskedAdd$(\mathbf{z_j}, \mathbf{m'_{j+2 \bmod 6}})$
13         $\mathbf{z_j} \leftarrow$ MaskedSBox$(\mathbf{S'_{j+6}}, \mathbf{z_j})$
14     for $j$ in Perm(2) do
15         $\mathbf{t_{3j}} \leftarrow$ MaskedAdd(MaskedAdd$(\mathbf{z_{3j}}, \mathbf{z_{3j+1}})$, $\mathbf{z_{3j+2}}$)
16         $\mathbf{t_{3j+1}} \leftarrow$ MaskedAdd$(\mathbf{z_{3j+1}}, \mathbf{z_{3j+3 \bmod 6}})$
17         $\mathbf{t_{3j+2}} \leftarrow$ MaskedAdd(MaskedAdd$(\mathbf{z_{3j+2}}, \mathbf{z_{3j+3 \bmod 6}})$, $\mathbf{y_{3j}}$)
18     $m\_perm \leftarrow [5, 4, 3, 1, 0, 2]$
19     for $j$ in Perm(6) do
20         $\mathbf{t_j} \leftarrow$ MaskedAdd$(\mathbf{t_j}, \mathbf{m'_{m\_perm[j]}})$
21     $\mathbf{z} \leftarrow \mathbf{m'_6}$
22     for $j$ in Perm(6) do
23         $\mathbf{u_j} \leftarrow$ MaskedSBox$(\mathbf{S'_{j+12}}, \mathbf{t_j})$
24         $\mathbf{z} \leftarrow$ MaskedAdd$(\mathbf{z}, \mathbf{u_j})$
25     return $\mathbf{z}$
```

is further reduced due to the order of operations depending on those elements that are consistently shuffled.

6.2 Template Attack on Countermeasures

In contrast, one can adapt the template attack to compute templates for the RSIs and the mask shares instead of the keyround elements.

For RSIs, we have 98 templates for p_{rws}, 14 templates for p_{round} and 7 templates for each of the 14 p_{block_r}. For mask shares, we have $16 \cdot 2$ templates for each of the $\mathbf{m^{rws}_j}$ and $\mathbf{m^{rounds}_{r,b}}$. The total number is thus increased to 6,482.

For trace $\mathbf{t_i}$, the probabilities of keyround element $\mathbf{k_i}[j]$, renamed here $\hat{k}_j$ for convenience, is computed from the ones of RSIs and mask shares as follows [27]:

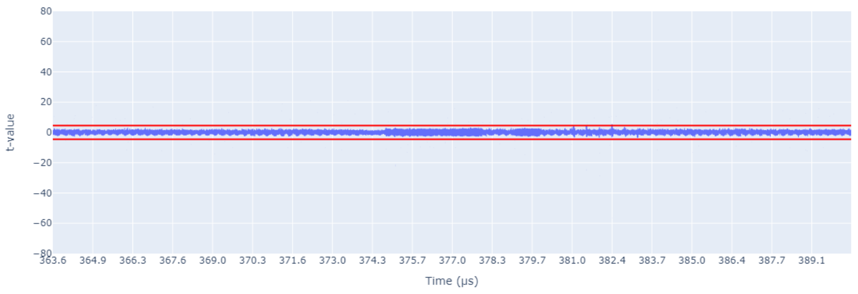


Fig. 7: TVLA on last round of `Elisabeth-b4` with 2-share masking and shuffling

$$p_{\hat{k}_j}\left(\dot{k}\;\middle|\;\mathbf{t_i}\right) = p_{\hat{k}_j^{rws}}\left(\dot{k}\;\middle|\;\mathbf{t_i}\right)\cdot p_{\hat{k}^{rounds}_{\lfloor j/7\rfloor,\, j \bmod 7}}\left(\dot{k}\;\middle|\;\mathbf{t_i}\right) \tag{5}$$

where

$$p_{\hat{k}_j^{rws}}\left(\dot{k}\;\middle|\;\mathbf{t_i}\right) = \sum_{p=0}^{97} p_{p_{rws}}\left(p \mid \mathbf{t_i}\right)\cdot\left(\sum_{m=0}^{15} p_{\mathbf{m_{j'}^{rws}}[0]}\left(m \mid \mathbf{t_i}\right)\cdot p_{\mathbf{m_{j'}^{rws}}[1]}\left(\dot{k}-m \mod 16\;\middle|\;\mathbf{t_i}\right)\right),$$

$$\begin{aligned}p_{\hat{k}^{rounds}_{r,b}}\left(\dot{k}\;\middle|\;\mathbf{t_i}\right) = \sum_{p_r=0}^{13} p_{p_{round}}\left(p_r \mid \mathbf{t_i}\right)\cdot\Bigg(\sum_{p_b=0}^{6} p_{p_{block_{r'}}}\left(p_b \mid \mathbf{t_i}\right)\\ \cdot\left(\sum_{m=0}^{15} p_{\mathbf{m^{rounds}_{r',b'}}[0]}\left(m \mid \mathbf{t_i}\right)\cdot p_{\mathbf{m^{rounds}_{r',b'}}[1]}\left(\dot{k}-m \mod 16\;\middle|\;\mathbf{t_i}\right)\right)\Bigg),\end{aligned}$$

with $j' = j-p \mod 98$, $r' = r-p_r \mod 14$, and $b' = b-p_b \mod 7$. Just like without the countermeasures, we then revert the **RWS**, add all the logarithms of probabilities and choose the highest one as the final prediction for each key element.

During the profiling phase, we compute the PoIs of every predicted variable (RSI and mask share) using SOST, select the N highest time points, and build the templates using those. The value of N is 5-fold cross-validated to be 1,000 for p_{rws}, 200 for p_{round} and p_{block_r}, and 20 for $\mathbf{m_j^{rws}}$ and $\mathbf{m^{rounds}_{r,b}}$. We used 675,000 profiling traces and 250,000 extraction traces from Dataset D and got the results displayed in Table 2.

Even though we see that mask shares have an accuracy slightly better than the baseline, it was not enough to reconstruct the complete key without a large number of extraction traces. Indeed, only 20.7% of the key was recovered with 250,000 extraction traces.

We also tried predicting both shares together instead of separately, i.e. compute 16^2 templates for every masked variable. Our validation accuracies in this case were worse and it required more computing power as more than 50,000 templates needed to be computed. Overall, predicting each share separately would give a more consistent performance.

We did not evaluate the ML-based attack with the countermeasures since it already performed worse than the template without them. The latter thus successfully mitigate the attacks that we were able to mount previously: CPA, template attack and ML-based attack from Section 4. We will now study the usage of Deep Learning as a more complex method to bypass them.

7 Deep Learning Application to Bypass Protections

7.1 Straightforward Monolithic Approach

Since our implemented countermeasures were very similar to the ones attacked by Masure and Strullu [27], we adapt their `MultiResNetSCA-1` to our use-case. Their architecture predicts 34 variables each between 0 and 255 by taking a trace of 15,000 time points and feeding it to a residual network, whose output is then fully connected to each output variable through 1024-node hidden layers, totalling in 137,396,064 trainable parameters.

In our case, we predict 408 variables, counting all RSIs and mask shares. This large number of output variables and the large trace size (200,000 time points) results in a neural network with 20,871,800,210 trainable parameters if we use the exact same architecture. Even if we wanted to, we do not have the computing capabilities to train such a gigantic model. To reduce the number of parameters, we modify the architecture in three ways:

- We compute the second order Haar wavelet that divides our trace size by 4 while keeping most information of the trace [11].
- We select the parts of the trace that leak the variables using SOST on the wavelets and concatenate them. Combined with the previous wavelet transform, this reduces the number of input points to 12,150, a number similar to the original architecture.
- We reduce the number of hidden nodes from 1,024 to 512 for RSIs and to 128 for mask shares.

These modifications result in an architecture with 188,046,738 parameters. A graphical representation is shown in Appendix B. We kept the same loss function – the categorical cross-entropy – and trained it during 30 epochs with a dataset of 675,000 training traces.

The mean test accuracies for all predicted variables using Dataset D are displayed in Table 2. It resulted in a success rate of 12.5 %, slightly better than the baseline of 6.25% but not even reaching the performance of the template attack from Section 6.2. We explain this poor performance due to the wavelet transform discarding useful information found in higher frequencies. Indeed, the PoIs identified by the SOST were less clear after the wavelet transform, compared to the original traces. Another explanation could be that the model architecture is not complex enough to capture the data structure and that the optimizer gets stuck in a local minimum – our model is underfitting.

7.2 A Better Modular Approach

Instead of prematurely using wavelets to reduce our input space, we only detect the PoIs of our variables using SOST and train multiple smaller – but with more capabilities – models on subsets of the trace, following the *Divide & Conquer* paradigm. Concretely, we train 20 models that have similar architectures but with different input and output sizes.

- `ResNetRWSPerm`, taking the entire **RWS** of 19,850 time points and predicting p_{rws}. This model has 11,497,410 trainable parameters.
- `ResNetRoundPerm`, taking the PoIs of p_{round} of 700 time points and predicting p_{round}. This model has 4,389,998 trainable parameters.
- $\{$`ResNetRWS-0`$, \dots,$ `ResNetRWS-3`$\}$ each taking one quarter of the **RWS** of 4,860 (5,270 for `ResNetRWS-3`) time points and each predicting 48 (52 for `ResNetRWS-3`) of the $\mathbf{m_j^{rws}}$. Splitting **RWS** in less than four led to models too complex to be trained. These models have 63,107,168 (74,680,480 for `ResNetRWS-3`) trainable parameters.
- $\{$`ResNetRound-0`$, \dots,$ `ResNetRound-13`$\}$ each taking round $r \in \{0, \dots, 13\}$ of 1,400 time points (identified with the SOSTs of involved variables) and predicting their respective p_{block_r} and $\mathbf{m_{r,b}^{rounds}}$. These models have 26,213,447 trainable parameters.

This sums up to a total of 646,877,650 trainable parameters but that can be trained on multiple GPUs in parallel. A graphical representation is shown in Appendix C. We used 675,000 traces for training, 75,000 for validation and chose the Adam optimizer with default values except an exponential decay β_1 of 0.99 to escape local minima [21]. We early stopped the training by monitoring the validation loss with a patience of 10 epochs.

`ResNetRWSPerm` converged after 3 epochs, while the rest after 10 to 15 epochs. Mean test accuracies for all predicted variables are displayed in Table 2, requiring 250,000 extraction traces to recover the entire key.

Figure 8 depicts the expected rank of each of the 512 elements of the key for different numbers of extraction traces. We practically computed the expected rank of a key element for a given number of extraction traces using the same method as in 4.5, sampling this time from 250,000 traces not seen during the training. The average rank of a key element is 0.003 at 250,000 extraction traces. We notice that few key elements still have a mean rank not steadily converging to 0. We do not have a rational explanation for this behaviour, even though there are only 4 of them out of 512 and that they are still below rank 1, meaning they are still ranked between first and second in average.

8 Conclusion

`Elisabeth-b4` offers a lightweight solution for preserving user privacy in cloud computing as an HHE scheme. Our side-channel analysis showed that a 2048-bit

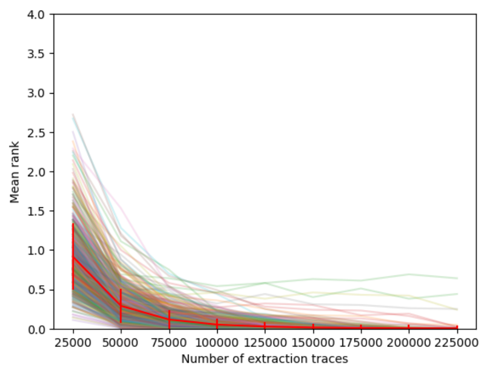


Fig. 8: 20 Neural Networks with 2-share masking and shuffling: Expected key ranks

Table 2: Overall attack results with countermeasures

Method	# Traces		Mean test accuracies (%)					Success rate (%)
	Profiling	Extraction	p_{rws}	p_{round}	p_{block_r}	$\mathbf{m_j^{rws}}$	$\mathbf{m_{r,b}^{rounds}}$	
CPA	mitigated	mitigated	-	-	-	-	-	0
Template	675,000	250,000	69.96	65.92	58.01	7.71	7.37	20.7
Single NN	675,000	250,000	74.12	58.93	37.71	6.85	6.98	12.5
20 NNs	675,000	250,000	99.35	75.11	60.23	9.58	8.44	100
Baseline	0	0	1.02	7.14	14.29	6.25	6.25	6.25

key could be extracted with a limited number of traces. In side-channel analysis, non-linear operations typically exhibit the highest leakage, which we observed as the CPA specifically targeted the S-boxes. These S-boxes, structured as NLUTs with a minimal output size of just 4 bits, introduced uncertainty regarding whether any exploitable leakage would be present. Nevertheless, our attack proved successful even on a 32-bit processor. Moreover, other classic techniques such as template attacks enabled key recovery with as few as 1,000 traces.

Using well-known mitigations like masking and shuffling, the previous attack is mitigated but a stronger attacker can still train 20 independent deep neural networks and acquire 250 times more traces on the target to retrieve the key.

As a direction for future work, we doubt that the same analysis could be transposed to power analysis as we suspect most of our leaking information to be found in higher frequencies of the leakage signal. With a lot of the leakage coming from the **RWS**, we wonder whether the latter could be modified to keep the security level while avoiding to unnecessarily manipulate all keyround elements for every plaintext nibble. Instead, it could be thought of precomputing the keyrounds in advance once per seed.

A Expected number of traces before every key element is observed

The key has $N = 512$ elements and $n = 98$ of them are uniformly sampled at each trace. We would like to compute the expected number of traces after which all N elements are chosen. This problem is a variant of the Coupon collector's problem but instead of choosing one element with replacement at a time, we choose n of them.

We model the problem as a discrete-time Markov chain where its states represent the number of key elements that were already selected. We are interested in the expected number of steps to go from state 0 to state N. The probability $p_{i,j}$ to go from state i to state j can be computed as:

$$p_{i,j} = \begin{cases} \frac{\binom{N-i}{j-i}\binom{i}{n-(j-i)}}{\binom{N}{n}}, & \text{if } i \leq j \leq i+n. \\ 0, & \text{otherwise.} \end{cases}$$

Indeed, we cannot decrease our number of selected elements so $p_{i,j} = 0$ if $i > j$. We can select a maximum of n elements at each step so $p_{i,j} = 0$ if $j > i + n$. In the remaining case, we count the number of ways of selecting $j - i$ new elements from the $N - i$ remaining to be selected, while also selecting the rest $(n - (j - i))$ from the i already selected. We then divide by the total number of ways of selecting n elements from N.

Our chain is an instance of an absorbing Markov chain, i.e., $\exists i, p_{i,i} = 1$, and its only absorbing state is N because we can always increase the number of already selected elements except if all of them have been so. There exists a useful theorem that allows us to compute the expected number of steps before the chain is absorbed starting from any state [18]. In our case, we start at state 0. Applying the theorem shows that the expected number of steps is the sum of the first row of $(\mathbf{I_N} - \mathbf{Q})^{-1}$ where $\mathbf{Q} = \begin{pmatrix} p_{0,0} & \cdots & p_{0,N-1} \\ \vdots & \ddots & \vdots \\ p_{N-1,0} & \cdots & p_{N-1,N-1} \end{pmatrix}$.

Replacing N with 512 and n with 98 leads to an expected number of steps of 32.61, rounding up to 33.

B Adapted `MultiResNetSCA-1` architecture

We adapted the `MultiResNetSCA-1`'s architecture, originally proposed by [27] into the one shown in Figure 9, with $f_{\cdot}(\cdot)$ being probability density functions.

C Our 20 neural network architectures

`ResNetRWSPerm`, `ResNetRoundPerm`, $\{$`ResNetRWS-0`$, \ldots,$ `ResNetRWS-3`$\}$ and $\{$`ResNetRound-0`$, \ldots,$ `ResNetRound-13`$\}$ are shown in Figure 10, with $f_{\cdot}(\cdot)$ being probability density functions.

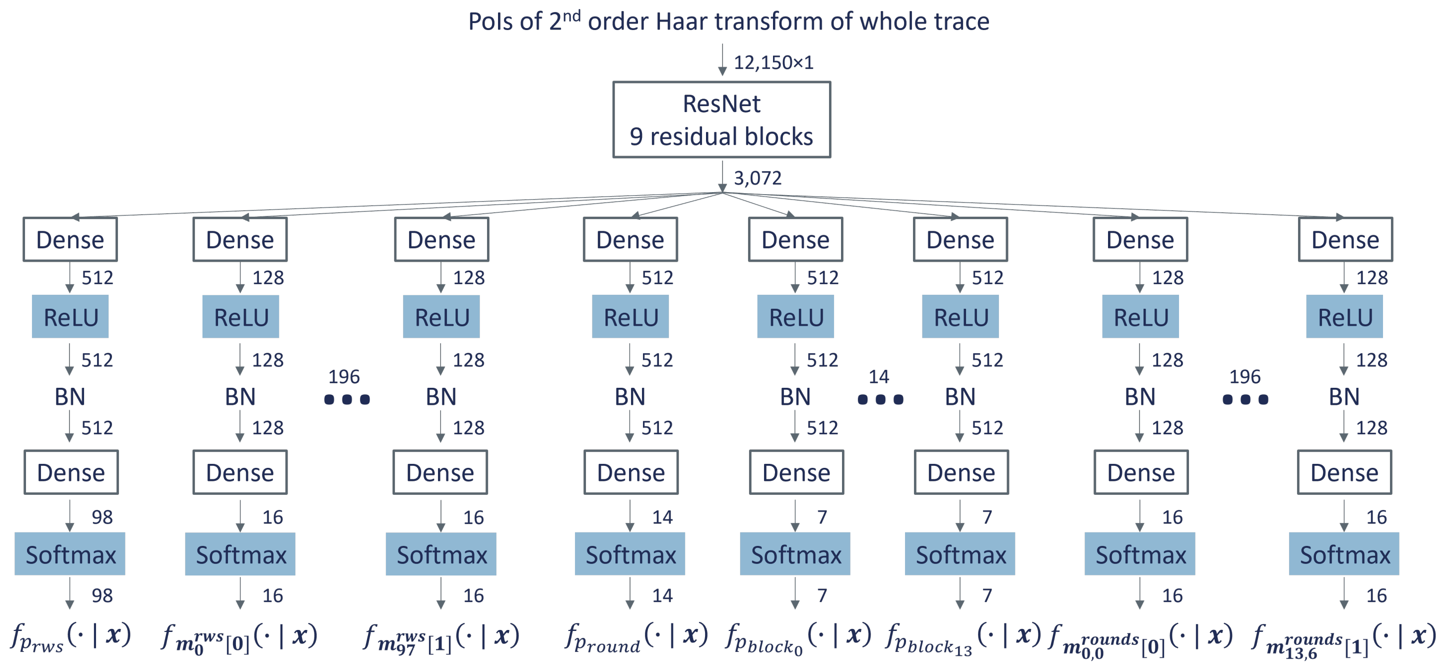
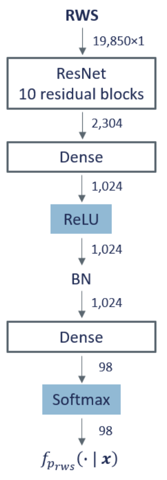
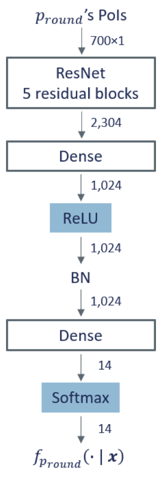

PoIs of 2nd order Haar transform of whole trace
12,150×1
ResNet
9 residual blocks
3,072
Dense
ReLU
BN
Softmax
512
128
98
16
14
7
196
14
$f_{p_{rws}}(\cdot \mid \boldsymbol{x})$
$f_{\boldsymbol{m}_0^{rws}[\boldsymbol{0}]}(\cdot \mid \boldsymbol{x})$
$f_{\boldsymbol{m}_{97}^{rws}[\boldsymbol{1}]}(\cdot \mid \boldsymbol{x})$
$f_{p_{round}}(\cdot \mid \boldsymbol{x})$
$f_{p_{block_0}}(\cdot \mid \boldsymbol{x})$
$f_{p_{block_{13}}}(\cdot \mid \boldsymbol{x})$
$f_{\boldsymbol{m}_{0,0}^{rounds}[\boldsymbol{0}]}(\cdot \mid \boldsymbol{x})$
$f_{\boldsymbol{m}_{13,6}^{rounds}[\boldsymbol{1}]}(\cdot \mid \boldsymbol{x})$

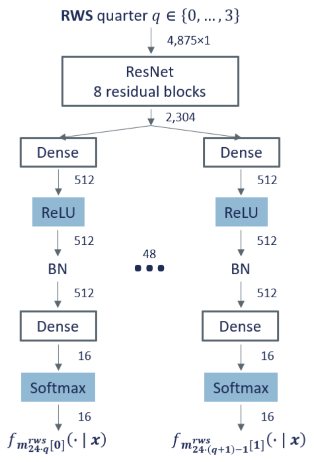
Fig. 9: Adapted `MultiResNetSCA-1` [27].



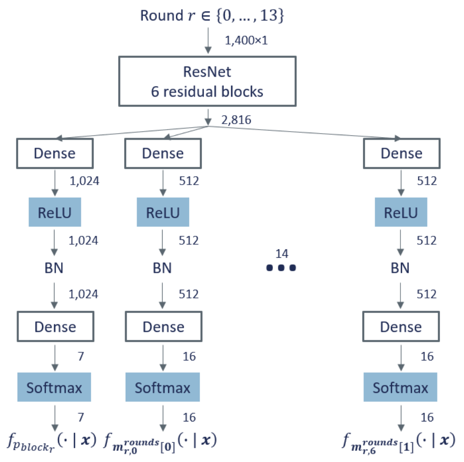
(a) `ResNetRWSPerm` architecture.



(b) `ResNetRoundPerm` architecture.


RWS quarter $q \in \{0, \dots, 3\}$
4,875×1
ResNet
8 residual blocks
2,304
Dense
512
ReLU
512
BN
512
Dense
16
Softmax
16
48
$f_{m^{rws}_{24 \cdot q}[0]}(\cdot \mid \boldsymbol{x})$
$f_{m^{rws}_{24 \cdot (q+1)-1}[1]}(\cdot \mid \boldsymbol{x})$

(c) `ResNetRWS-0/.../3` architectures.


Round $r \in \{0, \dots, 13\}$
1,400×1
ResNet
6 residual blocks
2,816
Dense
1,024
512
ReLU
BN
Dense
7
16
Softmax
14
$f_{p_{block_r}}(\cdot \mid \boldsymbol{x})$
$f_{m^{rounds}_{r,0}[0]}(\cdot \mid \boldsymbol{x})$
$f_{m^{rounds}_{r,6}[1]}(\cdot \mid \boldsymbol{x})$

(d) `ResNetRound-0/.../13` architectures.

Fig. 10: Our 20 neural networks architectures.

References

1. Akkar, M., Giraud, C.: An implementation of DES and aes, secure against some attacks. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2162, pp. 309–318. Springer (2001). https://doi.org/10.1007/3-540-44709-1_26, https://doi.org/10.1007/3-540-44709-1_26
2. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 430–454. Springer (2015). https://doi.org/10.1007/978-3-662-46800-5_17, https://doi.org/10.1007/978-3-662-46800-5_17
3. Archambeau, C., Peeters, E., Standaert, F., Quisquater, J.: Template attacks in principal subspaces. In: Goubin, L., Matsui, M. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4249, pp. 1–14. Springer (2006). https://doi.org/10.1007/11894063_1, https://doi.org/10.1007/11894063_1
4. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001). https://doi.org/10.1023/A:1010933404324, https://doi.org/10.1023/A:1010933404324
5. Canteaut, A., Carpov, S., Fontaine, C., Lepoint, T., Naya-Plasencia, M., Paillier, P., Sirdey, R.: Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. J. Cryptol. **31**(3), 885–916 (2018). https://doi.org/10.1007/S00145-017-9273-9, https://doi.org/10.1007/s00145-017-9273-9
6. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M.J. (ed.) Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 398–412. Springer (1999). https://doi.org/10.1007/3-540-48405-1_26, https://doi.org/10.1007/3-540-48405-1_26
7. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Jr., B.S.K., Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2523, pp. 13–28. Springer (2002). https://doi.org/10.1007/3-540-36400-5_3, https://doi.org/10.1007/3-540-36400-5_3
8. Cho, M., Chung, W., Ha, J., Lee, J., Oh, E., Son, M.: FRAST: tfhe-friendly cipher based on random s-boxes. IACR Cryptol. ePrint Arch. p. 745 (2024), https://eprint.iacr.org/2024/745
9. Cosseron, O., Hoffmann, C., Méaux, P., Standaert, F.: Towards globally optimized hybrid homomorphic encryption - featuring the elisabeth stream cipher. IACR Cryptol. ePrint Arch. p. 180 (2022), https://eprint.iacr.org/2022/180
10. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press (2000). https://doi.org/10.1017/CBO9780511801389, https://doi.org/10.1017/CBO9780511801389

11. Debande, N., Souissi, Y., Elaabid, M.A., Guilley, S., Danger, J.: Wavelet transform based pre-processing for side channel analysis. In: 45th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2012, Workshops Proceedings, Vancouver, BC, Canada, December 1-5, 2012. pp. 32–38. IEEE Computer Society (2012). https://doi.org/10.1109/MICROW.2012.15, https://doi.org/10.1109/MICROW.2012.15
12. van Eck, W.: Electromagnetic radiation from video display units: An eavesdropping risk? Comput. Secur. **4**(4), 269–286 (1985). https://doi.org/10.1016/0167-4048(85)90046-X, https://doi.org/10.1016/0167-4048(85)90046-X
13. Fan, G., Zhou, Y., Zhang, H., Feng, D.: How to choose interesting points for template attacks? IACR Cryptol. ePrint Arch. p. 332 (2014), http://eprint.iacr.org/2014/332
14. Friedman, J.H.: Greedy function approximation: A gradient boosting machine. The Annals of Statistics **29**(5), 1189 – 1232 (2001). https://doi.org/10.1214/aos/1013203451, https://doi.org/10.1214/aos/1013203451
15. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009. pp. 169–178. ACM (2009). https://doi.org/10.1145/1536414.1536440, https://doi.org/10.1145/1536414.1536440
16. Gierlichs, B., Lemke-Rust, K., Paar, C.: Templates vs. stochastic methods. In: Goubin, L., Matsui, M. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4249, pp. 15–29. Springer (2006). https://doi.org/10.1007/11894063_2, https://doi.org/10.1007/11894063_2
17. Gilbert, H., Boissier, R.H., Jean, J., Reinhard, J.: Cryptanalysis of elisabeth-4. In: Guo, J., Steinfeld, R. (eds.) Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part III. Lecture Notes in Computer Science, vol. 14440, pp. 256–284. Springer (2023). https://doi.org/10.1007/978-981-99-8727-6_9, https://doi.org/10.1007/978-981-99-8727-6_9
18. Grinstead, C.M., Snell, J.L.: Introduction to probability. American Mathematical Soc. (2012)
19. Hebborn, P., Leander, G.: Dasta - alternative linear layer for rasta. IACR Trans. Symmetric Cryptol. **2020**(3), 46–86 (2020). https://doi.org/10.13154/TOSC.V2020.I3.46-86, https://doi.org/10.13154/tosc.v2020.i3.46-86
20. Hoffmann, C., Méaux, P., Standaert, F.: The patching landscape of elisabeth-4 and the mixed filter permutator paradigm. In: Chattopadhyay, A., Bhasin, S., Picek, S., Rebeiro, C. (eds.) Progress in Cryptology - INDOCRYPT 2023 - 24th International Conference on Cryptology in India, Goa, India, December 10-13, 2023, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14459, pp. 134–156. Springer (2023). https://doi.org/10.1007/978-3-031-56232-7_7, https://doi.org/10.1007/978-3-031-56232-7_7
21. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015), http://arxiv.org/abs/1412.6980

22. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Koblitz, N. (ed.) Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings. Lecture Notes in Computer Science, vol. 1109, pp. 104–113. Springer (1996). https://doi.org/10.1007/3-540-68697-5_9, https://doi.org/10.1007/3-540-68697-5_9
23. Kocher, P.C., Jaffe, J., Jun, B., Rohatgi, P.: Introduction to differential power analysis. J. Cryptogr. Eng. **1**(1), 5–27 (2011). https://doi.org/10.1007/S13389-011-0006-Y, https://doi.org/10.1007/s13389-011-0006-y
24. Kumar, S., Dasu, V.A., Baksi, A., Sarkar, S., Jap, D., Breier, J., Bhasinl, S.: Side channel attack on stream ciphers: A three-step approach to state/key recovery. IACR Trans. Cryptogr. Hardw. Embed. Syst. p. 166 (2022), https://doi.org/10.46586/tches.v2022.i2.166-191
25. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks - revealing the secrets of smart cards. Springer (2007)
26. Mangard, S., Popp, T., Gammel, B.M.: Side-channel leakage of masked cmos gates. Topics in Cryptology – CT-RSA 2005. p. 351 (2005), https://matpack.de/personal/Mangard_Popp_Gammel_CTRSA2005.pdf
27. Masure, L., Strullu, R.: Side channel analysis against the anssi's protected AES implementation on ARM. IACR Cryptol. ePrint Arch. p. 592 (2021), https://eprint.iacr.org/2021/592
28. Méaux, P., Carlet, C., Journault, A., Standaert, F.: Improved filter permutators for efficient FHE: better instances and implementations. In: Hao, F., Ruj, S., Gupta, S.S. (eds.) Progress in Cryptology - INDOCRYPT 2019 - 20th International Conference on Cryptology in India, Hyderabad, India, December 15-18, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11898, pp. 68–91. Springer (2019). https://doi.org/10.1007/978-3-030-35423-7_4, https://doi.org/10.1007/978-3-030-35423-7_4
29. Moradi, A.: Side-channel leakage through static power – should we care about in practice? IACR Cryptol. ePrint Arch. p. 562 (2014), https://eprint.iacr.org/2014/025.pdf
30. Naehrig, M., Lauter, K.E., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: Cachin, C., Ristenpart, T. (eds.) Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011. pp. 113–124. ACM (2011), https://dl.acm.org/citation.cfm?id=2046682
31. Vadnala, P.K.: Provably Secure Countermeasures against Side-channel Attacks. Ph.D. thesis, University of Luxembourg (2015), http://orbilu.uni.lu/handle/10993/21653
32. Veyrat-Charvillon, N., Medwed, M., Kerckhof, S., Standaert, F.: Shuffling against side-channel attacks: A comprehensive study with cautionary note. In: Wang, X., Sako, K. (eds.) Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7658, pp. 740–757. Springer (2012). https://doi.org/10.1007/978-3-642-34961-4_44, https://doi.org/10.1007/978-3-642-34961-4_44
33. Wang, W., Tang, D.: Differential fault attack on he-friendly stream ciphers: Masta, pasta and elisabeth. IACR Cryptol. ePrint Arch. p. 1005 (2024), https://eprint.iacr.org/2024/1005