Another Look at Side-Channel Resistant Encoding Schemes

Xiaolu Hou[1], Jakub Breier[2] and Mladen Kovačević[3]

[1]Faculty of Informatics and Information Technologies, Slovak University of Technology, Slovakia. E-mail: houxiaolu.email@gmail.com.

[2]TTControl GmbH, Vienna, Austria. E-mail: jbreier@jbreier.com.

[3]Faculty of Technical Sciences, University of Novi Sad, Serbia. E-mail: kmladen@uns.ac.rs.

Abstract

The idea of balancing the side-channel leakage in software was proposed more than a decade ago. Just like with other hiding-based countermeasures, the goal is not to hide the leakage completely but to significantly increase the effort required for the attack. Previous approaches focused on two directions: either balancing the Hamming weight of the processed data or deriving the code by using stochastic leakage profiling.

In this brief, we build upon these results by proposing a novel approach that combines the two directions. We provide the theory behind our encoding scheme backed by experimental results on a 32-bit ARM Cortex-M4 microcontroller. Our results show that such a combination gives better side-channel resistance properties than each of the two methods separately.

1 Introduction

Side-channel analysis (SCA) attacks are a well-known attack vector affecting the security of cryptographic implementations that has been around for more than two decades [KJJ99]. The main working principle of SCA is observing the physical characteristics of the device during the encryption, for example in the form of power consumption or electromagnetic emanation, based on which the attacker can recover confidential information, such as encryption keys.

In the area of symmetric cryptography, two main countermeasure concepts have emerged: *masking* and *hiding*. Masking countermeasures use random numbers to "mask" the sensitive values so that there is no correlation between these values and the side-channel leakage. Currently, the most popular masking-based scheme is threshold implementation [NRR06]. Masking has been shown to be provably secure, given that the source of randomness is truly random [RP10]. Hiding countermeasures aim to decrease the signal-to-noise ratio to make side-channel attacks harder. This means that, while the attacker can eventually succeed in the key recovery, the number of leakage traces is significantly higher compared to attacking an unprotected implementation. In some scenarios, such as payment cards, this might mean preventing an attack if the number of required traces is higher than the limit on the number of transactions set by the card issuer. Some of the hiding-based schemes are built upon shuffling the cipher operation [VCMKS12] and utilizing complementary wires and computations in the form of so-called "dual-rail" [CZ06].

In this brief, we are interested in hiding countermeasures in software based on coding theory. Several schemes have been proposed to date, such as balancing the Hamming weight of the processed values by using look-up table operations [CESY15], software approach to imitate dual-rail [HDD11], or codes derived from stochastic profiling to take the separate bit leakages into the account [MSB16]. The last scheme was also extended to provide protection against fault injection attacks [BH17].

Our contribution. We revisit encoding schemes in software by combining two different approaches to provide better protection against SCA than the previously proposed encoding-based works. More specifically, we combine the balancing of Hamming weight with the stochastic leakage characteristic of the underlying device. We provide experimental results on a PRESENT-80 Sbox implementation running on a 32-bit ARM Cortex-M4 microcontroller. Our results show that even with 50k traces, the guessing entropy metric shows a very small reduction in the remaining brute force search that needs to be done by the attacker to recover the secret key. This is in line with our expectations that the proposed method performs better than the two previously proposed approaches.

2 Background and Related Work

2.1 Binary codes

We recall the definition of a binary code [LX04]. Let n be a positive integer. *A nonempty set $C \subseteq \{0,1\}^n$ is called a binary code of length n. Elements of C are called codewords. The number of codewords in C is called the size of C. A binary code of length n and size M is called an $(n, M)-$binary code.*

For example, $\{000, 111\} \subseteq \{0,1\}^3$ is a $(3,2)-$binary code. It is easy to see that if we wish to encode information with bit length k, we need to have a binary code of size 2^k. For example, suppose we want to encode bits 0 and 1, then we would look for binary codes of size 2. We can take the above mentioned $(3,2)-$binary code $\{000, 111\}$ and encode 0 as 000, 1 as 111 (the so-called repetition code). To decode, we simply decode 111 to 1 and 000 to 0.

We note that the information on which binary code is used in the countermeasure is considered public.

2.2 Related work

The initial idea to hide the information leakage by balancing the Hamming weight in software was proposed by Hoogvorst et al. in 2011 [HDD11]. Servant et al. [SDMB15] took the idea and implemented a constant Hamming weight[1] AES by using a binary code of length 6 whose all codewords have Hamming weight 3. This was followed by Rauzy et al. [RGN16] who translated dual-rail circuits into look-up tables in software (titled "DPL"), showing that if properly balanced, DPL implementation can resist up to 4810 traces. The authors used a 16-bit ATmega163 microcontroller for their experiments. Chen et al. [CESY15] proposed a different approach that took the original nibble of data $b_0b_1b_2b_3$ and translated it into $\bar{b_0}b_0\bar{b_1}b_1\bar{b_2}b_2\bar{b_3}b_3$, where $\bar{b_i}$ is a binary complement of b_i, thus always having a Hamming weight of 4. An experimental evaluation on an 8-bit AVR microcontroller showed no side-channel leakage until $\approx 50k$ traces. While all the previously listed works focused on the Hamming weight leakage model, Maghrebi et al. [MSB16] experimented with stochastic leakage, where the contribution of each bit is determined by device profiling. This approach allows tailoring the code according to the device and achieves better results, as the Hamming weight model is often inaccurate due to process variation and other factors influencing the physical computation.

Our work bridges the gap by combining the two approaches – utilizing the stochastic device profile together with Hamming weight-balanced code to improve on the previous results.

[1]Hamming weight of a non-negative integer is defined as the number of 1s in its binary representation.

3 Method

As has been shown before, the leakage at a single time sample can be profiled with a stochastic leakage model [DPRS11]. When a value $\mathbf{x} = x_{n-1}x_{m-2}\dots x_1x_0$ with bit length n is being processed in the device, the stochastic leakage model assumes that the leakage is related to each bit of $\mathbf{x}$ as follows:

$$\mathcal{L}(\mathbf{x}) = \sum_{s=0}^{n-1} \alpha_s x_s + N, \tag{1}$$

where $N \sim \mathcal{N}(0, \sigma^2)$ denotes the Gaussian noise with mean 0 and variance σ^2. Here α_s ($s = 0, 1, \dots, n-1$) are real numbers referred to as the *coefficients* of the stochastic leakage model.
The steps of our countermeasure are as follows:

① **Identify the target intermediate value and vulnerable instruction**. For the proposed countermeasure, same as in [MSB16], we focus on a single time sample, which we will refer to as point of interest (POI). While the whole cipher state can be encoded, it is not practical. Instead, we can choose which part of the cipher state to encode. The first step is to identify the *target intermediate value* and the vulnerable instruction(s) in the cryptographic implementation. The target intermediate value, denoted $\mathbf{v}$, is the value that we would like to protect. In particular, we would like to know the bit length of $\mathbf{v}$, m_v, and hence determine the code size of our binary code, which is given by 2^{m_v}. The vulnerable instruction is the instruction that would result in the most leakage while operating on the target intermediate value during the computation.

For example, in differential power analysis, the Sbox output is a common target [MOP08]. In such a scenario, we can choose $\mathbf{v}$ to be one Sbox output. For illustration, we will demonstrate the countermeasure on a PRESENT [BKL+07] Sbox implementation. As we have $m_v = 4$, we are interested in binary codes of size $2^4 = 16$. For the vulnerable instruction, we choose the `MOV` instruction, as memory operations tend to have the highest leakage and they are a common target of SCA.

② **Choose the code length** n_C. As mentioned before, we would like to combine the proposal of utilizing the stochastic device profile with the Hamming weight-balanced code. In particular, we look for binary codes of size 2^{m_v} such that each codeword has the same Hamming weight. We note that there are in total $\binom{n_C}{w_H}$ vectors in $\{0,1\}^{n_C}$ of Hamming weight w_H. Since $\binom{n_C}{w_H}$ is maximized when w_H is $\lfloor n_C/2 \rfloor$ or $\lceil n_C/2 \rceil$, we need to choose n_C such that

$$\binom{n_C}{\lfloor n_C/2 \rfloor} > 2^{m_v},$$

so that we can construct a binary code of size 2^{m_v}, length n_C, and with every codeword having the same Hamming weight.

It is easy to see that the longer the code length, the more choices we have for the Hamming weight values. On the other hand, longer code length results in bigger memory overhead. We need to find a compromise in the level of freedom of protection the code gives and the overhead it causes. For the illustration, we will choose $n_C = 8$ so that we can have a few choices for the fixed Hamming weight of each codeword.

③ **Identify the possible Hamming weight values**. We propose to analyze a few binary codes that have balanced Hamming weight and choose the best-performing one for the final countermeasure. With the chosen code length n_C, the number of possible codewords with fixed Hamming weight w_H is given by $\binom{n_C}{w_H}$. Thus, all the possible Hamming weight values

w_H are those for which

$$\binom{n_C}{w_H} > 2^{m_v}.$$

For the illustration, we have $n_C = 8$, $2^{m_v} = 2^4 = 16$. All the possible Hamming weight values are $2, 3, 4, 5, 6$.

④ **Experimental setup and trace measurement**. Since the goal is to profile the device and find the best code, we would take measurements with devices that we expect the potential attackers to have. Two sets of traces are to be collected, $\mathcal{T}_1$ and $\mathcal{T}_2$. Each trace in $\mathcal{T}_1$ corresponds to the computation of the vulnerable instruction with random values of $\mathbf{v}$ as input. This dataset will be used to identify the POI, which is the time of the computation that is supposed to be the most vulnerable. Each trace from $\mathcal{T}_2$ corresponds to the computation of the vulnerable instruction with random values of bit length n_C as input. $\mathcal{T}_2$ will be used to profile the stochastic leakage of the device, namely finding estimations for the coefficients α_s from Equation (1). We note that it is important for traces in those two datasets to be horizontally aligned so that the profiling of $\mathcal{T}_2$ is carried out with the correct POI. Suppose there are M_1 traces in $\mathcal{T}_1$ and M_2 traces in $\mathcal{T}_2$.

⑤ **POI identification**. With $\mathcal{T}_1$, we compute the SNR at each time sample and choose the time sample with the highest SNR value to be our POI. Recall that SNR is given by

$$\text{SNR} = \frac{\text{Var(signal)}}{\text{Var(noise)}}.$$

We follow a standard way for SNR calculation in the context of SCA (see e.g. [MOP08, Section 4.3.2]) – we first group the traces into different sets, $A_1, A_2, \ldots, A_{2^{m_v}}$, according to the corresponding value of $\mathbf{v}$, then for a given time sample t:

- We compute the averaged leakage for traces in each set, denoted $\mu_{1,t}, \mu_{2,t}, \ldots, \mu_{2^{m_v},t}$.
- The variance of the signal, Var(signal), is given by the variance of the values $\mu_{1,t}, \mu_{2,t}, \ldots, \mu_{2^{m_v},t}$.
- The noise for each trace $\boldsymbol{\ell}$ at time sample t is given by the leakage at t subtracted by the corresponding average $\mu_{i,t}$, where $\boldsymbol{\ell} \in A_i$.
- The variance of noise, Var(noise), is given by the variance of noises in each trace at time sample t.

⑥ **Estimate stochastic leakage model coefficients**. Let $\mathbf{x}_i$ denote the input corresponding to trace $\boldsymbol{\ell}_i$ in $\mathcal{T}_2$. Suppose the binary representation of $\mathbf{x}_i$ is given by

$$\mathbf{x}_i = x_{i(n_C-1)} \ldots x_{i1} x_{i0}, \quad i = 1, 2, \ldots, M_2.$$

Then we define matrix M as follows

$$M = \begin{pmatrix} x_{10} & x_{11} & \cdots & x_{1(n_C-1)} \\ x_{20} & x_{21} & \cdots & x_{2(n_C-1)} \\ \vdots & \vdots & \ddots & \vdots \\ x_{M_2 0} & x_{M_2 1} & \cdots & x_{M_2(n_C-1)} \end{pmatrix}.$$

Next, we take

$$\boldsymbol{\ell}^{pf} = (\boldsymbol{\ell}_{1,\text{POI}}, \boldsymbol{\ell}_{2,\text{POI}}, \ldots, \boldsymbol{\ell}_{M_2,\text{POI}})$$

to be the array of leakage values at the POI from each trace in $\mathcal{T}_2$. According to the *ordinary least square method* [DPRS11], the estimations of α_s values in Equation (1), denoted $\hat{\alpha_s}$, are given by

$$(\hat{\alpha}_0, \hat{\alpha}_1, \ldots, \hat{\alpha}_{n_C-1}) = \left(M^T M\right)^{-1} M^T \boldsymbol{\ell}^{pf}.$$

Then, for any binary string $\mathbf{x} = x_{n_C-1}x_{n_C-2}\ldots x_1 x_0$, we define its *estimated signal* to be

$$\mathrm{SG}(\mathbf{x}) \coloneqq \sum_{s=0}^{n_c} \hat{\alpha}_s x_s. \tag{2}$$

⑦ **Find optimal codes for each possible Hamming weight value**. For each possible Hamming weight identified in Step 3, we will find an optimal code using Algorithm 1. This algorithm is a modification of [MSB16, Algorithm 1], which does not preserve the constant Hamming weight.

For the algorithm's input, m_v is the bit length of the target intermediate value identified in Step 1; n_C is the code length chosen in Step 2; w_H is one of the possible Hamming weight values found in Step 3; $\hat{\alpha}_s$ are estimations of stochastic leakage model coefficients obtained in Step 6. In line 1, as the name suggests, code_size specifies the size of the binary code, and total_word is the total number of binary strings of length n_C and Hamming weight w_H, where those binary strings are found and stored in the array S in lines 6 – 8. Line 9 computes the estimated signal of each element of S with Equation (2) and stores it to the array T_{SG}. T_{sorted} is the sorted version of T_{SG} such that $T_{\mathrm{sorted}}[0]$ contains the lowest value from T_{SG} (line 10). Array I records the corresponding binary string in S for each estimated signal in T_{sorted} (lines 11 and 12). In the jth entry of array D, we put the difference between the value in $T_{\mathrm{sorted}}[j+\text{code\_size}-1]$ and $T_{\mathrm{sorted}}[j]$ (line 14). Then, we find the index of the smallest value in D (line 15). Finally, our optimal code consists of codewords that correspond to estimated signals in the interval $D[\text{ind}]$ and $D[\text{ind} + \text{code\_size} - 1]$ (lines 16 and 17).

Now, let C be any $(n_C, 2^{m_v})-$binary code and define

$$\mathrm{SG}(C) \coloneqq \{\, \mathrm{SG}(\mathbf{c}) \;\mid\; \mathbf{c} \in C \,\}$$

to be the set of estimated signals of codewords in C. When C is used for encoding the target intermediate value, the variance of the signal at POI is then given by $\mathrm{Var}(\mathrm{SG}(C))$. We note that the goal of Algorithm 1 is to find a C such that

$$\begin{aligned} d &\coloneqq \max\{\ |\,\mathrm{SG}(\mathbf{c}_i) - \mathrm{SG}(\mathbf{c}_j)| \mid \mathbf{c}_i, \mathbf{c}_j \in C\,\} \\ &= \max\{\ |\,a_i - a_j| \mid a_i, a_j \in \mathrm{SG}(C)\,\} \end{aligned}$$

is the minimum among all $(n_C, 2^{m_v})-$binary codes whose codewords have Hamming weight w_H. Let a_i $(i = 1, 2, \ldots, 2^{m_v})$ denote the elements of $\mathrm{SG}(C)$ and let $\overline{a}$ be the mean of those numbers. Then by definition

$$\mathrm{Var}(\mathrm{SG}(C)) = \frac{1}{2^{m_v}} \sum_{i=1}^{2^{m_v}} (a_i - \overline{a})^2 \leq d^2.$$

Thus, Algorithm 1 indeed finds a binary code with a relatively small Var(signal), resulting in a small SNR. Although it is possible to compare $\mathrm{Var}(\mathrm{SG}(C))$ for all binary codes with the same parameters $n_C, 2^{m_v}, w_H$, it is too slow, especially when n_C is big.

Algorithm 1: Finding the optimal code with given Hamming weight.

```
   Input: $m_v,\ n_C,\ w_H,\ \hat{\alpha}_s\ (s = 0, 1, \ldots, n_C - 1)$
   Output: An $(n_C, 2^{m_v})-$binary code with each codeword having Hamming weight $w_H$
1  code_size = $2^{m_v}$, total_word = $\binom{n_C}{w_H}$
2  empty array $S$, $T_{\mathrm{SG}}$
3  array of size total_word-code_size+1 $D$
4  array of size total_word $I$
5  array of size code_size $C$
6  for $i = 0,\ i < 2^{n_C},\ i{+}{+}$ do
7      if Hamming weight of $i == w_H$ then
8          append $i$ to $S$
9          append SG($i$) to $T_{\mathrm{SG}}$
10 $T_{\mathrm{sorted}} = T_{\mathrm{SG}}$ sorted in ascending order
11 for $j = 0,\ j <$ total_word$,\ j++$ do
12     $I[j] = S$ [index of $T_{\mathrm{sorted}}[j]$ in $T_{\mathrm{SG}}$]
13 for $j = 0,\ j \leq$ total_word $-$ code_size$,\ j++$ do
14     $D[j] = T_{\mathrm{sorted}}[j +$ code_size $- 1] - T_{\mathrm{sorted}}[j]$
15 ind = $\arg\min_j D[j]$
16 for $j = 0,\ j <$ code_size$,\ j++$ do
17     $C[j] = I[\mathrm{ind} + j]$
18 return $C$
```

4 Experimental Evaluation

As an illustration, we have chosen PRESENT Sbox output as the target intermediate value and `MOV` as the vulnerable instruction. The bit length of our target intermediate value is hence $m_v = 4$. As mentioned in Step 3, we have chosen $n_C = 8$ and all the possible Hamming weight values are $2, 3, 4, 5, 6$.

For the measurements, NewAE ChipWhisperer-Lite was used. The target device is a 32−bit ARM Cortex-M4 microcontroller (STM32F303RDT6) with a clock speed of ≈ 7.4 MHz. ADC was set to capture the samples at $4\times$ that speed, i.e. ≈ 29.6 MHz with a 10−bit resolution. We measured the computation of the operation

$$\texttt{MOV}\ \ \texttt{r0}\ \ \mathbf{x},$$

where `r0` represents a register. $M_1 = 10,000$ traces were measured with $\mathbf{x}$ being a random 4−bit value, giving us the dataset $\mathcal{T}_1$. Dataset $\mathcal{T}_2$ was obtained by taking $\mathbf{x}$ to be random 8−bit values and we also measured $M_2 = 10,000$ traces.

The SNR values computed with $\mathcal{T}_1$ are shown in Figure 1. The highest point corresponds to the time sample 430, which will be our POI. The estimations of α_s computed with our dataset $\mathcal{T}_2$ are as follows:

$$\begin{aligned} &\hat{\alpha}_0 \approx -0.0024576, && \hat{\alpha}_1 \approx -0.0013003, \\ &\hat{\alpha}_2 \approx -0.0013588, && \hat{\alpha}_3 \approx -0.0012280, \\ &\hat{\alpha}_4 \approx -0.0013157, && \hat{\alpha}_5 \approx -0.0021347, \\ &\hat{\alpha}_6 \approx -0.0020975, && \hat{\alpha}_7 \approx -0.0022129. \end{aligned}$$

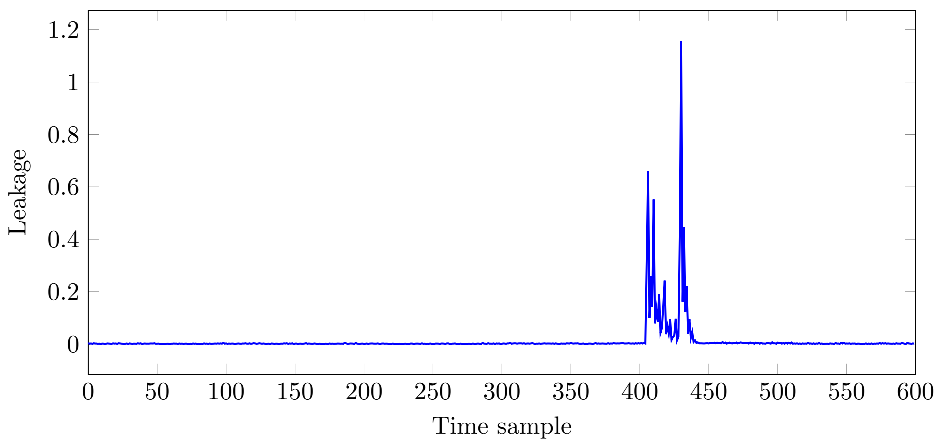

Figure 1: SNR values for each time sample computed with dataset $\mathcal{T}_1$. The highest point is our POI = 430.

Five $(8, 16)-$binary codes were found using Algorithm 1 with $w_H = 2, 3, 4, 5, 6$. For example, with $w_H = 3$, the code we have obtained is given by (in hexadecimal)

[`92, 34, 8C, 26, 54, 98, 46, 32, 8A, 2C, 52, 4C, 38, 2A, 58, 4A`].

To identify the best code and compare it with existing works, we implemented differential power analysis (DPA) attacks based on correlation coefficient analysis [Sta10] on the `MOV` instruction using the same POI = 430. Such an attack is also sometimes referred to as a correlation power analysis (CPA) attack. In particular, with each random plaintext nibble p and a fixed key nibble `9`, we compute the PRESENT Sbox output SB($p \oplus$`9`), where PRESENT Sbox is given by `C56B90AD3EF84712`. Then, we take measurements with `MOV` instruction. For the unprotected implementation, the Sbox output is the input operand, and for the protected implementation, the encoded Sbox output is the input operand. As an attacker, we do not have the knowledge of the real key `9`, thus we make a hypothesis and compute the correlation coefficients between the corresponding hypothetical leakages and real measurements. The key guesses are ranked according to the resulting absolute value of correlation coefficients. It is desirable for the correct key guess (i.e. `9`) to be ranked the 1st. Guessing entropy [SMY09] is a common criterion used in DPA attacks to demonstrate the effectiveness of the attack, which is defined to be the expected ranking of the correct key guess. The lower the guessing entropy, the better the attack. Empirically, we repeat the experiments many times, 100 in our case, to get an estimation of this value.

4.1 Results

The attack results are shown in Figure 2. As the baseline, we can see that the unprotected implementation reaches a guessing entropy of 1 very fast. The code where each codeword has Hamming weight 3 performs the best, especially outperforming the two existing proposals [CESY15] and [MSB16]. The key rank stabilizes around 12 and stays at that point until 50k traces. As there are only 16 possible key guesses for one key nibble, we can conclude that the protection is rather effective. As an attacker, if we would like to brute force the key with all the top 12 ranked key guesses, we will need to brute force 16 nibbles each with 12 possibilities, resulting in $12^{16} \approx 2^{57.4}$ remaining key space for one $64-$bit round key. To recover the master key, the remaining key space is then $\approx 2^{73.4}$ in case of PRESENT-80. The codes with Hamming weights 4 and 5 take the second and the third best rank, respectively. The attacks

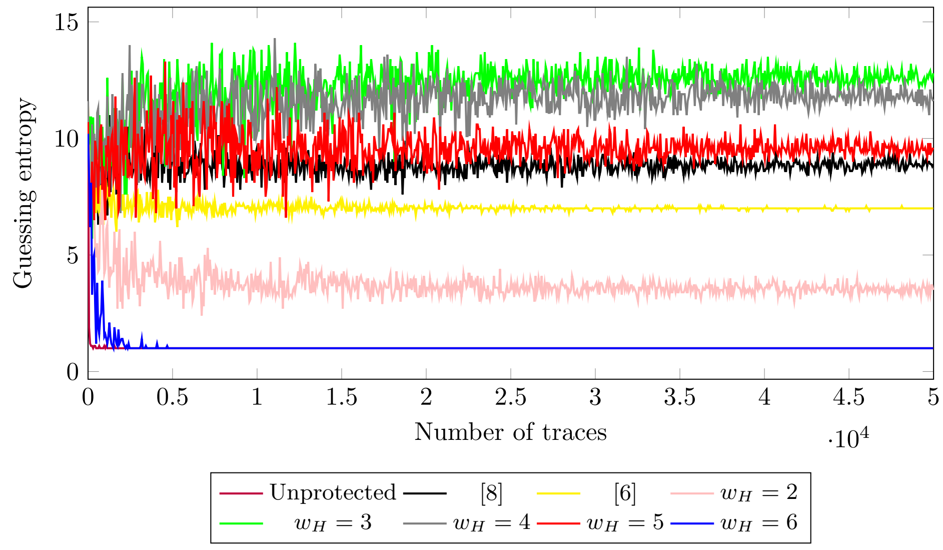

Figure 2: Guessing entropy of CPA attacks on `MOV` instruction with different $(8, 16)-$binary codes. [CESY15] is a balanced 8-bit Hamming weight code and [MSB16] is the stochastic encoding.

on the proposal from [MSB16] stabilize at key rank 9, resulting in $9^{16} \times 2^{16} \approx 2^{66.7}$ remaining key space. Attacks on the balanced encoding proposal from [CESY15] reach key rank 7, thus the key search space is reduced to $7^{16} \times 2^{16} \approx 2^{60.9}$. On the other hand, we can see that the codes for Hamming weights 2 and 6 do not perform well, for some reason. That shows that it is always important to perform an experimental evaluation with the target device as the leakage modeling is always just an estimation of the real leakage.

5 Discussion

Hiding vs. masking. As mentioned in the introduction, masking-based approaches are generally more effective as they utilize randomness to mask the source of leakage. However, hiding-based methods can make the attacker's task significantly harder by decreasing SNR and are often implemented on top of masking to enhance its potential. A noteworthy advantage of hiding over masking is that it does not require a source of randomness. This is especially important for small-scale low-power devices that do not contain random number generators (RNGs) or if they do, they can be biased [HSHC17].

Fault protection. Encoding countermeasures necessarily create memory overheads based on the chosen code. As codes can naturally provide error detection/correction, it makes sense to design a combined countermeasure against both SCA and fault attacks. A fault attack evaluation of proposals from [CESY15, MSB16, RGN16] was provided in [BJB17], essentially showing that even without a specific fault-tolerant design, these schemes can provide a decent security margin against fault attacks. Furthermore, the stochastic encoding scheme [MSB16] was taken as a basis for a combined countermeasure in [BH17] which shows an automated way to generate codes that can provide specified trade-offs between side-channel and fault resistance. We would argue that with minor adjustments, the same method can be used to tailor the codes used in this work.

Encoding the entire computation. Even though the presented attack focuses on one instruction, the whole cipher state can be encoded, giving protection to all the other instructions. We refer the readers to [BHL19] for a method of encoding the whole encryption computation. Different codes might work for different devices, but as evaluators, we would have access to the device we want to protect and can choose the best code that is suitable for the device.

Overhead. The overheads of the protected implementation are dependent on the used code length. More details on that topic can be found in [BHL19], where the authors compared various codes with the PRESENT-80 implementation. The reported execution time overhead was $\approx 82.5\%$, while the required memory for codes up to length 8 was $\approx$ 2MB. Additionally, some memory reduction tricks can be used to avoid implementing large tables, as mentioned in [SDMB15].

6 Conclusion

In this brief, we looked into a hiding-based countermeasure against side-channel attacks based on coding theory. More specifically, we analyzed the previously proposed software encoding schemes that used either balanced Hamming weights [CESY15] or stochastic encoding [MSB16]. Realizing that a combination of the two would likely result in even better leakage properties, we developed a method to generate such codes. We experimentally verified our proposal and found that, indeed, a constant Hamming weight code that takes the leakage characteristics of the device into account provides better resistance than each of these encoding countermeasures separately.

For future work, a natural extension would be to investigate fault resistance properties of the codes that can be generated by using our method.

Acknowledgement

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under the Programme SASPRO 2 COFUND Marie Sklodowska-Curie grant agreement No. 945478.

This work was supported by the Slovak Research and Development Agency under the Contract no. SK-SRB-21-0059.

References

[BH17] Jakub Breier and Xiaolu Hou. Feeding two cats with one bowl: On designing a fault and side-channel resistant software encoding scheme. In *Topics in Cryptology–CT-RSA 2017: The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14–17, 2017, Proceedings*, pages 77–94. Springer, 2017.

[BHL19] Jakub Breier, Xiaolu Hou, and Yang Liu. On evaluating fault resilient encoding schemes in software. *IEEE Transactions on Dependable and Secure Computing*, 18(3):1065–1079, 2019.

[BJB17] Jakub Breier, Dirmanto Jap, and Shivam Bhasin. A study on analyzing side-channel resistant encoding schemes with respect to fault attacks. *Journal of Cryptographic Engineering*, 7:311–320, 2017.

[BKL$^+$07] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. Present: An ultra-lightweight block cipher. In *Cryptographic Hardware and*

Embedded Systems-CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings 9, pages 450–466. Springer, 2007.

[CESY15] Cong Chen, Thomas Eisenbarth, Aria Shahverdi, and Xin Ye. Balanced encoding to mitigate power analysis: a case study. In *Smart Card Research and Advanced Applications: 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers 13*, pages 49–63. Springer, 2015.

[CZ06] Zhimin Chen and Yujie Zhou. Dual-rail random switching logic: a countermeasure to reduce side channel leakage. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 242–254. Springer, 2006.

[DPRS11] Julien Doget, Emmanuel Prouff, Matthieu Rivain, and François-Xavier Standaert. Univariate side channel attacks and leakage modeling. *Journal of Cryptographic Engineering*, 1:123–144, 2011.

[HDD11] Philippe Hoogvorst, Guillaume Duc, and Jean-Luc Danger. Software implementation of dual-rail representation. *COSADE, February*, pages 24–25, 2011.

[HSHC17] Darren Hurley-Smith and Julio Hernandez-Castro. Certifiably biased: An in-depth analysis of a common criteria eal4+ certified trng. *IEEE Transactions on Information Forensics and Security*, 13(4):1031–1041, 2017.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, pages 388–397. Springer, 1999.

[LX04] San Ling and Chaoping Xing. *Coding theory: a first course.* Cambridge University Press, 2004.

[MOP08] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*, volume 31. Springer Science & Business Media, 2008.

[MSB16] Houssem Maghrebi, Victor Servant, and Julien Bringer. There is wisdom in harnessing the strengths of your enemy: Customized encoding to thwart side-channel attacks. In *Fast Software Encryption: 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers 23*, pages 223–243. Springer, 2016.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In *International conference on information and communications security*, pages 529–545. Springer, 2006.

[RGN16] Pablo Rauzy, Sylvain Guilley, and Zakaria Najm. Formally proved security of assembly code against power analysis: A case study on balanced logic. *Journal of Cryptographic Engineering*, 6:201–216, 2016.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of aes. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 413–427. Springer, 2010.

[SDMB15] Victor Servant, Nicolas Debande, Houssem Maghrebi, and Julien Bringer. Study of a novel software constant weight implementation. In *Smart Card Research and Advanced Applications: 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers 13*, pages 35–48. Springer, 2015.

[SMY09] François-Xavier Standaert, Tal G Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Advances in Cryptology-EUROCRYPT 2009*, pages 443–461. Springer, 2009.

[Sta10] François-Xavier Standaert. Introduction to side-channel attacks. *Secure integrated circuits and systems*, pages 27–42, 2010.

[VCMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In *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 18*, pages 740–757. Springer, 2012.