IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2017, No. 3, pp. 108–129. DOI:10.13154/tosc.v2017.i3.108-129

Practical Evaluation of FSE 2016 Customized Encoding Countermeasure

Shivam Bhasin[1], Dirmanto Jap[1] and Thomas Peyrin[1,2,3]

[1] Temasek Laboratories, Nanyang Technological University, Singapore, Singapore
sbhasin@ntu.edu.sg,djap@ntu.edu.sg,thomas.peyrin@ntu.edu.sg
[2] School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore, Singapore
[3] School of Computer Science and Engineering, Nanyang Technological University, Singapore, Singapore

Abstract. To protect against side-channel attacks, many countermeasures have been proposed. A novel customized encoding countermeasure was published in FSE 2016. Customized encoding exploits knowledge of the profiled leakage of the device to construct an optimal encoding and minimize the overall side-channel leakage. This technique was originally applied on a basic table look-up. In this paper, we implement a full block cipher with customized encoding countermeasure and investigate its security under simulated and practical setting for a general purpose microcontroller. Under simulated setting, we can verify that customized encoding shows strong security properties under proper assumption of leakage estimation and noise variance. However, in practical setting, our general observation is that the side-channel leakage will mostly be present even if the encoding scheme is applied, highlighting some limitation of the approach. The results are supported by experiments on 8-bit AVR and 32-bit ARM microcontroller.

Keywords: side-channel attacks, software countermeasures, customized encoding, block cipher implementation, microcontroller

1 Introduction

Side-channel attacks (SCA) have been shown to bypass even a theoretically secure cryptosystem, simply by attacking the vulnerability of a cryptographic implementation on a real device [Koc96]. They have been widely investigated and are now considered a serious threat, especially against any security device which solely relies on the theoretical security. Thus, an ever-increasing need to protect confidential data has motivated research on side-channel countermeasures that could reduce (or eliminate) the leakage exploitation of the data.

These countermeasures can be mainly classified into two categories, namely, masking and hiding countermeasures [MOP07]. In hiding countermeasures [TV04], the designer tries to hide the dependencies between the processed data and the side-channel leakage, by altering (or balancing) the side-channel leakage characteristics. Masking countermeasures [GP99] use a random number for side-channel protection: the designer tries to mask the intermediate value by introducing a random element into the sensitive computation and thus, even if side-channel leakage can be observed, it corresponds to the randomized computation which makes the sensitive data hard to deduce. This paper focuses on hiding countermeasures specially designed for software (or microcontroller) environment.

Received: 2017-06-01, Accepted: 2017-08-01, Published: 2017-09-19

1.1 Related Works

Regarding software hiding countermeasures, there have been several novel proposals. The first hiding countermeasure in software was proposed by Hoogvorst *et al.* [HDD11]. In their work, they used the principle of dual-rail precharge logic (DPL [TV04]), i.e. a hardware hiding countermeasure, and modified it to suit software implementations, in order to balance the leakage of the data. Then, Rauzy *et al.* [RGN13] reported the first practical encoding implementation of `PRESENT` cipher using bit-slice method on assembly instructions. In this case, each bit is encoded with its negation (typically 1 will be processed as 01 and 0 will be processed as 10). Later, Chen *et al.* [CESY14] proposed assembly-level protection by balancing the number of 1s and 0s in each instruction. For example, if the processed data is $a_0a_1...a_n$, it will be encoded to $a_0\bar{a_0}a_1\bar{a_1}...a_n\bar{a_n}$. The difference with previous approach is that here, multi bits could be used, whereas in previous approach, it is bit-slice method, processing only 1 bit, and replicated into several bits. Finally, at FSE 2016, it was shown that previous works on encoding countermeasures have neglected to take into consideration that each bit processed might not have the same leakage, and thus, there will be some residual exploitable leakage [MSB16]. They then proposed to perform a stochastic characterization on the leakage of the device, and based on the leakage characteristics, construct an optimal encoding scheme which could minimize the variance of the leakage in general. In fact, authors in [RGN13] and [CESY14] also observed that there are different leakage characteristics for different bits in a byte. However, their encoding did not take this imbalance into consideration and this was only dealt with in [MSB16]. Recently, these encoding schemes were also shown to have good fault resistance properties [BJB16, BH17], which makes them interesting candidates for combined countermeasures.

Our Contributions: In this paper, we investigate the claims from [MSB16], and we will refer to their countermeasure as *customized encoding*. Our investigation is done on the encoding properties in a simulated setting, followed by a comprehensive analysis with real measurements from a general purpose microcontroller. Moreover, we also expand this previous work, which only verified the results on a memory look-up operation, to a full cipher implementation. The contributions of our paper is therefore as follows:

- We performed the evaluation of the customized encoding countermeasure proposed in [MSB16]. We have performed the profiling and the implementation of the encoding countermeasure on practical settings. Our results show that in general, the leakage can still be observed and in practice, it is hard to implement the customized encoding countermeasure.

- We highlight the drawbacks of the approach for practical implementations, and in particular the difficulties in performing accurate profiling of the device which causes a compromise on the encoding. Besides, the encoding countermeasure is shown to be restricted on the setup used, which can not generalize to other settings. These difficulties and restrictions for the encoding countermeasure can be observed from the results that using different registers, targeting different operations, or different points in time, that deviates from one used for the profiling can lead to a leakage.

Our paper is structured as follows: in Section 2, we provide a brief summary of the customized encoding countermeasure. Then, in Section 3, we describe the standard methods used for side-channel evaluation. In Section 4, we provide the theoretical evaluation results, followed by practical results on the tweakable block cipher `SKINNY` in Section 5 and further investigations on different platforms in Section 6. We eventually discuss our findings in Section 7 and conclude the paper in Section 8.

2 Introduction to Customized Encoding

At FSE 2016, Maghrebi *et al.* [MSB16] proposed a customized encoding countermeasure built according to the precise leakage model based on a stochastic profiling. This method takes advantage of the adversary's knowledge of the physical leakage. In their work, the main objective is to choose the corresponding optimal encoding that balances (or minimizes) the variation in the side-channel leakage. This approach contrasts with previous encoding methods, as they were based on the assumption of the Hamming Weight (HW) leakage model, which, in practice, might not necessarily hold. Under imperfect HW leakage model, other encoding countermeasure will leak exploitable side-channel leakage and compromise the security of the implementation. Thus, the work in [MSB16] aims to address this issue and the encoding is designed by first estimating the actual leakage model observed from the device and then optimizing the encoding function selection based on this estimation.

The side-channel leakage is normally dependent on the device and the operations performed. For the case of software or microcontroller, it can often be observed that there might be a slight variance when measuring the leakage across different registers (although in [MSB16] the authors argued that the variance of the leakage in different registers is not very significant).

In general, the theoretical leakage can be modeled based on the processed intermediate values and thus the side-channel leakage can be formulated as follows:

$$T(x) = L(x) + \epsilon, \tag{1}$$

where L is the device specific leakage function that maps a deterministic intermediate value (x) processed in the register to its respective side-channel leakage, and ϵ is the (assumed) mean-free Gaussian noise ($\epsilon \sim N(0, \sigma^2)$). A commonly used leakage function for microcontrollers is the HW model or, more recently, the n-bit stochastic representation. For example, in an 8-bit microcontroller, the leakage can be represented as $L(x) = \beta_0 + \beta_1 x_1 + ... + \beta_8 x_8$, where x_i is the i^{th} bit of the intermediate value, and β_i is the i^{th} bit weight leakage for a specific register [SLP05]. For the HW model, β_1 to β_8 are considered to be unity. However, in reality, due to several physical device parameters, β will deviate from unity. For the remaining of the paper, we will use the stochastic representation when dealing with the leakage profiling of the device.

The deterministic part of the leakage can then be determined as $\tilde{L} = \mathbf{A} \cdot \beta$, where $\mathbf{A} = (x_{i,j})_{1 \leq i \leq N; 0 \leq j \leq n}$, with $\vec{x_i}$ as a row element of $\mathbf{A}$ and N denotes the number of measurements. Here, A is the matrix of the intermediate value from N measurements. Taking example from 8-bit microcontroller earlier, the i^{th} row can be represented as [1 x_1 ... x_8], where $n = 8$. We can then determine $\beta = (\beta_j)_{0 \leq j \leq 8}$ based on the set of traces $\mathbf{T}$, as follows:

$$\beta = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{T}. \tag{2}$$

After the profiling phase of the device, the weight leakages β coefficient of the device can be estimated, and then the encoding function can be computed, based on the method from [MSB16] shown in Algorithm 1. The algorithm can be summarized as follows: from the codewords space, calculate the hypothetical leakage based on estimated β coefficients, sort and determine which set minimizes the leakage difference among the codewords in the set.

After the algorithm returns the encoding function, the mapping from the original value to its corresponding codewords can be represented as a look-up table or a memory look-up in the case of a microcontroller implementation. The encoding function is shown to be optimum, as it minimizes the variance of the side-channel leakage assuming that the hypothetical β coefficients are estimated correctly. The hypothetical leakage, based here on the stochastic profiling, assumes that each bit is processed independently of the others and might not leak equally. Since different registers might leak differently, it might

Algorithm 1: Selection of the optimal encoding function [MSB16].

Input : m: the codeword bit-length, n: the sensitive variable bit-length, β_i: the leakage bit weights of the register, where i in $[\![1, m]\!]$

Output : 2^n codewords of m-bit length

for X *in* $[\![0, 2^m - 1]\!]$ **do**
  Compute the estimated power consumption for each codeword X and store the result in table D: $D[X] = \Sigma_{i=1}^{m} \beta_i X[i]$;
  Store the corresponding value of the codeword in the index table $I : I[X] = X$;
Sort the estimated power consumption stored in table D and the index table I accordingly
for j *in* $[\![0, 2^m - 2^n]\!]$ **do**
  Find the $argmin$ of $[\![D[j] - D[j + 2^n]]\!]$;
return 2^n codewords corresponding to $[\![I[argmin], I[argmin + 2^n]]\!]$

be necessary to generate different encoding functions for different registers, or to restrict the usage only to the profiled register.

In Table 1, an illustration is provided to show how encoded memory access is done. In this case, an operation between two words (could be the XOR operation or a SBox look-up), each encoded in 7 bits, is performed using the customized encoding. Note that it might be hard to implement directly an encoding with codewords of longer length, such as mapping from a nibble to a byte, since the tables will grow quadratically with the length of the code. Hence, the authors of [MSB16] proposed to encode a n-bit variable as two separate halves for any operation with two operands. Indeed, the encoding function does not preserve the algebraic property of the original data, and hence the look-up table alternative has to be used.

Table 1: Encoded memory access [MSB16]

#	Instruction
1	// R3 = *table_msb*, R4 = *table_lsb*, R5 = *shift*1_*table*, R6 = *shift*7_*table*
2	// R0 = operand MSB = 0xxxxxxx, R1 = operand LSB = 0yyyyyyy
3	LDRB R2, [R5,R0] // R2 = 00000000xxxxxxx0
4	EOR R0,R0,R0 // Clear R0
5	LDRH R0, [R6,R2] // R0 = 00xxxxxx0000000
6	EOR R0,R0,R1 // R0 = 00xxxxxxyyyyyyy
7	EOR R1,R1,R1 // Clear R1
8	EOR R2,R2,R2 // Clear R2
9	LDRB R1, [R3,R0] // R1 =*table_msb*[operand] (7 bits)
10	LDRB R2, [R4,R0] // R2 =*table_lsb*[operand] (7 bits)

From [MSB16], it is claimed that the customized encoding countermeasure can reduce the leakage close to minimal, based on a security evaluation that was conducted by the authors. This evaluation also indicated that the countermeasure is harder to attack than first-order masking. Moreover, their practical assessment also confirms the possible capability of their countermeasure to protect the cryptographic operations.

3 Side-Channel Evaluation Methods

In this section, we quickly recall the side-channel evaluation tools that we will use in the rest of the paper. As mentioned earlier, if all assumptions hold, the generated customized

encoding function can minimize the side-channel leakage, and thus increase the difficulty for the attacker to recover the secret key. To perform the evaluation, we considered the two most commonly used methods: the Correlation Power Analysis (CPA) and the Test Vector Leakage Assessment (TVLA). We used TVLA to test presence of any side-channel leakage, and CPA to verify if indeed the leakge is exploitable or not.

3.1 Correlation Power Analysis (CPA)

In [KJJ99], Kocher *et al.* proposed a Power Analysis Attacks based on the Difference of Means (DoM). The attack is done using a divide and conquer approach, where the attacker chooses one block of the data and performs the attack on this smaller subspace. In block ciphers, one of the main target is usually the output of a non-linear operation after the round key addition in the initial stage (for decryption) or the input of non-linear operation before the post key-whitening in the last stage (for encryption). The attacker then measures the side-channel leakage corresponding to the operation. Since the key subspace is smaller (only one round or less is computed from the target), the attacker can then calculate the hypothetical value based on different key hypothesis. He then computes one bit of the hypothetical value and computes the mean of different classes (0 or 1) from the leakage. The correct key hypothesis will then lead to the highest difference, since for a wrong hypothesis, the means of different classes will be more or less random, and hence the difference will tend to move towards 0.

This approach was then expanded to different leakage functions with multiple bits, by Brier *et al.* [BCO04], called Correlation Power Analysis (CPA). The authors proposed an improved attack based on a generic leakage model like the hamming weight (HW) model and using Pearson correlation as a distinguisher. It can be formulated as follows:

$$r = \frac{\sum_{i=1}^{N}(h_{i,k} - \bar{h_k})(t_{i,j} - \bar{t_j})}{\sqrt{\sum_{i=1}^{n}(h_{i,k} - \bar{h_k})^2 \sum_{i=1}^{N}(t_{i,j} - \bar{t_j})^2}}, \tag{3}$$

where $\vec{t_j} = \{t_{1,j}, ..., t_{N,j}\}$ and $\vec{h_k} = \{h_{1,k}, ..., h_{N,k}\}$ are the side channel measurements at point in time j and the hypothetical value for key hypothesis k. The values $\bar{t_j}$ and $\bar{h_k}$ denote the mean value of $\vec{t_j}$ and $\vec{h_k}$ respectively. The higher $|r|$ is, the stronger the correlation coefficient and the linear relationship between the hypothesis and the measurements. Later, it was also shown that the stochastic model [SLP05] can be used as an alternative leakage model to HW for CPA, in order to obtain more precise estimations of the leakage.

3.2 Leakage Detection

Feature selection is often done as a pre-processing step before SCA evaluation. It involves in identifying interesting features (or time-samples) and continue the evaluation only with selected subset. It helps to accelerate the attack as well as avoid ghost peaks from irrelevant sections of the trace. We use Normalized Inter-Class Variance (NICV [BDGN13, SGV08, PRB09]) for feature selection. It is computed as:

$$NICV = \frac{\mathbb{V}\{\mathbb{E}\{T|X\}\}}{\mathbb{V}\{T\}} \ , \tag{4}$$

where T denotes a side-channel trace and X is the public parameter (plaintext/ciphertext), used to partition the traces. $\mathbb{E}\{.\}$ and $\mathbb{V}\{.\}$ are statistical expectation and variance. NICV is bounded in the range $[0, 1]$. The key advantage of NICV is that it does not depend on the underlying leakage model but only on public input or outputs.

3.3 Test Vector Leakage Assessment (TVLA)

Comparing different implementations for SCA resistance is a tricky task. This is even harder when dealing with protections, as protection schemes are designed to alter the leakage behaviour. Effectiveness of a CPA (or SCA in general) depends on three principle components: measurement quality, leakage model and statistical test. In our study, the measurement quality can be considered constant as the same setup is used. The leakage model has a very high dependence on the underlying implementation and on the target platform or device. Countermeasures which modify leakage behavior actually impacts the leakage model. Any SCA countermeasure tries to invalidate standard leakage models. In the case of encoding, the countermeasure tries to nullify the leakage by balancing data bits. The same restriction applies on the statistical test also known as distinguisher. A simple distinguisher like correlation (CPA [BCO04]) is best when targeting a linear leakage. As encoding does not modify the leakage moments, CPA can be considered a good choice. For fair evaluation, the dependence on the leakage model should be carefully dealt with or rather removed. This brings in leakage detection techniques.

A leakage detection test investigates the presence of data-dependent leakage in side-channel measurements. We use Test Vector Leakage Assessment (TVLA [GJJR11]) as a leakage detection technique. TVLA checks for any first-order leakage in the captured traces without any assumption on the implementation or leakage model. We apply the non-specific fixed vs random test as proposed in [GJJR11]. The non-specific TVLA checks if the leakage from a cryptographic implementation depends on the plaintext (i.e. data dependency), without any assumption on the internal states of the cipher. Measurements are captured in two groups for the same secret key: fixed plaintext and variable plaintext. A null hypothesis is assumed to test if the two sets are identical using a **Welsh T-test**. If the absolute value of the T-test is $>$**4.5** (i.e. null hypothesis is rejected), then it indicates the presence of data-dependant leakage in side-channel traces. The measurement procedure for a device under test is as follows:

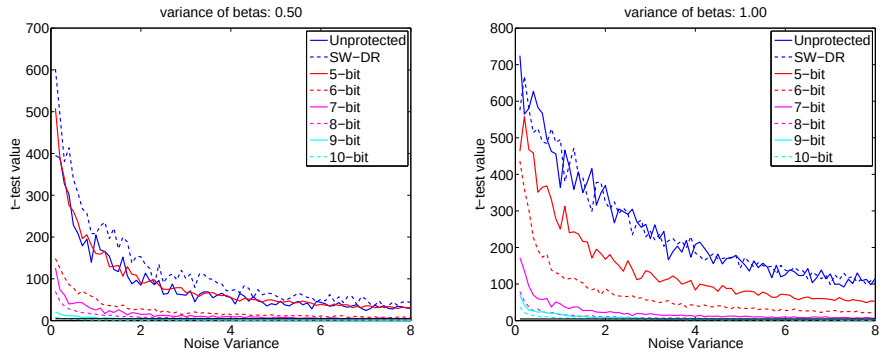
- **Step 1:** Capture N side-channel measurements for a fixed key and variable plaintext.
- **Step 2:** Compute the mean μ_r and standard deviation σ_r for these traces.
- **Step 3:** Capture N side-channel measurements for a fixed key and fixed plaintext.
- **Step 4:** Compute the mean μ_f and variance σ_f for these traces.
- **Step 5:** Compute the T-test: $T = \frac{\mu_r - \mu_l}{\sqrt{\frac{\sigma_r^2 + \sigma_f^2}{N}}}$

4 Simulation Analysis on the Impact of Noise on Balance Encoding

We perform simulation analysis for customized encoding with different variation of β coefficients and noise variances. The experiments are performed on a standard 8-bit AVR microcontroller manufactured in 350 nm technology node. An 8-bit stochastic model is used for profiling [SLP05].

The side-channel traces are captured while running a standard `AES` assembly implementation. In general, based on the profiling phase, there are three main sources of leakage in the look-up table implementation: the loading of the input (8-bit data) from the memory into the register, the loading of output value from the SBox look-up table and finally, storing back the output value into the memory (8-bit output). The approximated noise variances recorded were around 5.5 - 6.8 from multiple different time samples. Similarly, the variances of β coefficients across different time samples could vary between 0.2 - 0.8.

For simulation, 2 different β coefficients are chosen such that var(β) are 0.5, and 1. The variance of the noise is tested in range from 0.1 to 8 with an increment of 0.1. With the previously mentioned parameters, we tested the effectiveness of customized encoding, by applying the encoding function on 4 bit input. Codewords with length between 5 to 10 for customized encoding were generated according to Algorithm 1. For sake of comparison, the software dual-rail equivalent [CESY14] was also implemented. Note that all the implementations are compared under TVLA methodology.



(a) var(β)= 0.5 (b) var(β)= 1

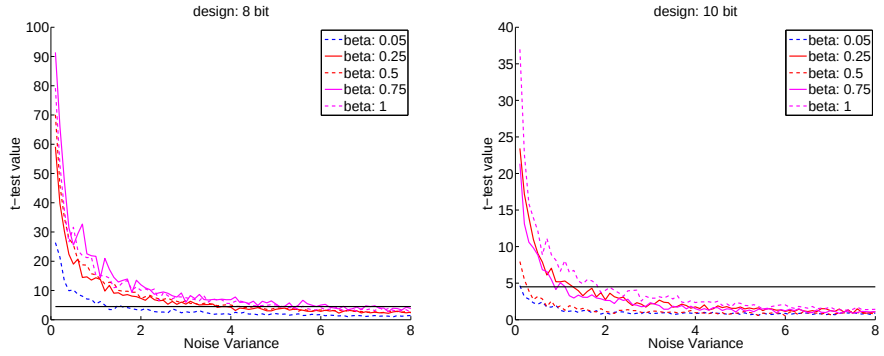
Figure 1: TVLA results for unprotected and countermeasure (5 to 10 bits encoding and software dual-rail (SW-DR)) with different β variances.

Fig. 1 highlights the (absolute) TVLA results. An implementation is considered to be leaking data-dependant information if the t-value crosses the $\pm$ 4.5 threshold (as proposed in [GJJR11] is derived from Student t-test in order to obtain the confidence level 0.99999). From the figure, we can observe that in general, the customized encoding from [MSB16] perform better than the software dual-rail countermeasure. Moreover, longer encodings are more effective in reducing the leakage. Software dual-rail worsens when the β coefficient variance increases (i.e. when the actual leakage model deviates highly from HW assumption). In order to have a better overview of the performances, we also include the results for 8-bit and 10-bit encoding separately in Fig. 2. We can observe the trend that the leakage is highly dependent on the length of the encoding scheme and the noise trade-off. In order to reduce the effect of the noise, longer codewords are needed, which might be a limitation for 8-bit architecture. These simulation results are in line with the results presented in [MSB16].

Based on the figures, the encoding function with 8- to 10-bit length codewords should theoretically be enough to reduce any exploitable side-channel leakage, under the observed noise variance from the `AES` traces. This shows that even for 8-bit architecture, and with reasonable noise level (around 5.5 - 6.8), the customized encoding is capable of protecting against the leakage. Thus, if all the assumptions are met, it can indeed prevent exploitable first order leakage, as highlighted by the TVLA results.

5 Experimental Study on `SKINNY`

After verifying the theoretical soundness of the customized encoding countermeasure in simulated setting, we now analyze it on real target. Since we are interested in testing longer encoding, `AES` might not be an ideal candidate. Moreover, for the customized encoding method, not many ciphers can easily be implemented due to some restrictions, such as the fact that only one register is profiled. Therefore, we decided to use `SKINNY` [BJK+16] as



(a) 8-bit encoding (b) 10-bit encoding

Figure 2: TVLA results for 8 to 10-bit encoding schemes with different noise levels

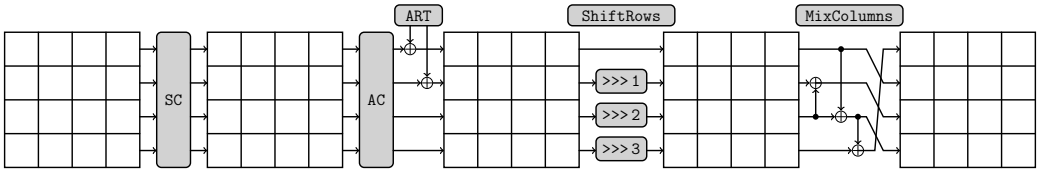


Figure 3: Illustration of one round of SKINNY [Jea16]

testing algorithm owing to its lightweight nature and bit-slice friendly design.

5.1 The SKINNY cipher

SKINNY is lightweight tweakable block cipher proposed at Crypto 2016 [BJK+16], as a competitor to NSA's SIMON in terms of performance. For software and microcontroller situations, it has a good efficiency and outperforms other known ciphers in term of performance. One round of SKINNY is highlighted in Fig. 3 and consists in the following operations: SubCells, AddConstants, AddRoundTweakey, ShiftRows, and MixColumns. For more information about the cipher, one can refer to [BJK+16].

For the SubCells operation, which is the operation that interests us here, if x_0, x_1, x_2 and x_3 represent the four inputs bits of the SBox (x_0 being the least significant bit), one simply applies the following transformation: $(x_3, x_2, x_1, x_0) \rightarrow (x_3, x_2, x_1, (x_0 \oplus \overline{(x_3 \vee x_2)}))$, followed by a left shift bit rotation. This process is repeated four times, except for the last iteration where the bit rotation is omitted.

5.2 The SKINNY implementation

In our experiments, we applied the customized encoding for implementing SKINNY on a 8-bit AVR microcontroller. We started with a bit-slice implementation of SKINNY written in AVR assembly language. The code is downloaded on a Arduino UNO microcontoller board running with a clock frequency of 16 Mhz. The board is modified to allow EM and power measurements. For measuring the traces, Lecroy WaveRunner 610zi oscilloscope is used at a sampling rate of 10 GSamples/s. The probe is positioned over the chip with the package mechanically opened, and the probe position is heuristically determined first by trial and error, and later verified by some test attack code for measurement correctness.

Since the encoding does not preserve the algebraic structure of the data, all the operations are implemented using look-up tables. The input length increases with the size

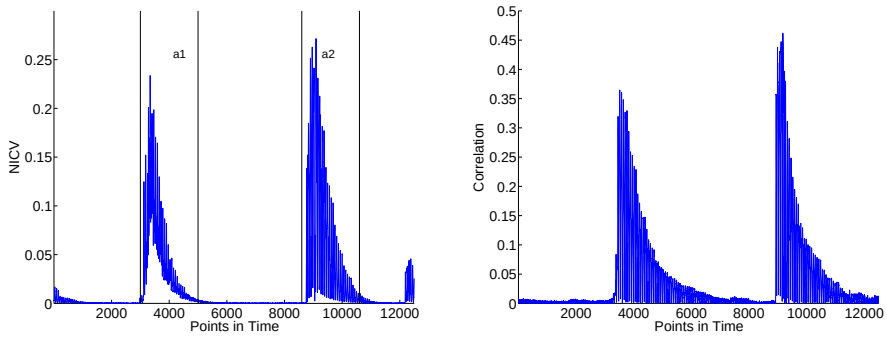
of the encoding (especially when concatenating 2 encoded inputs, $x_1||x_2$, for the look-up table input). In order to fit it within the frame of a 8-bit microcontroller, we used a bit-slice implementation where each bit is encoded on 4 bits (hence the concatenation will only take 8 bits). Note that we also chose the cipher `SKINNY` because its implementation only requires two logic operations, XOR and NOR, which makes it easier to apply the customized encoding.

5.3 Obtaining the β values

Using the stochastic approach, we first profiled the value for the β coefficients at different points in time. Since, we had already profiled the target 8-bit AVR microcontroller for an `AES` SBox look-up in the previous study under simulated setting (see Sec. 4), we use the same β coefficients. Indeed, any 8-bit operation can be potentially used for profiling. Since we target a 8-bit microcontroller and the `AES` SBox look-up table (LUT) is also over 8-bits, it forms a good match for the profiling phase. For the experiments, we focused on the Load (LD) and Store (ST) operations in the assembly. Thus, we modified our code as follows:

- trigger starts
- 20 NOPs
- LD from look-up table (X register)
- 20 NOPs
- ST back to memory (Z register)
- 20 NOPs
- end trigger

The measurements are taken using a high-sensitivity electromagnetic emanation (EM) probe, to capture localised sensitivity activity over a pre-determined target area. Several No Operation (NOP) instructions were included because in initial observation during the analysis, even though LD and ST operations are supposed to take only 2 clock cycles each, the leakage can still be observed several clock cycles later (shown as decaying trend in Fig. 4a, which might be attributed to the probe used for the experiments). Thus, the NOPs are added to isolate different operations from each other.



(a) Region of Interest (b) CPA results

Figure 4: Feature selection for β.

We have performed a feature selection (NICV) in order to identify the region of interest, which is highlighted in Fig. 4a. There are two main sources of leakages corresponding

to SBox output, the loading of the input (8-bit data) from the memory into the register, and storing back into the memory (8-bit output). To reconfirm this, we performed CPA and we can see from Fig. 4b that there are some time regions that are highly correlated with the SBox LUT (which can be the Load and Store from memory to register and vice versa). Both figures show consistency in the choice of features for constructing the encoding scheme. We performed the profiling for the LD operation, and measured two profiling sets, in order to check the consistency of the results.

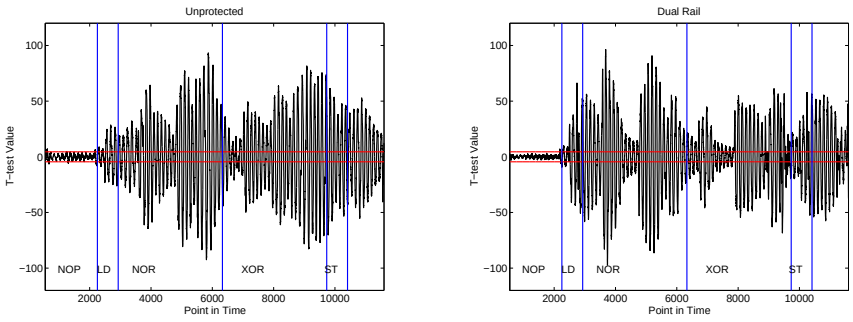
We then constructed the codewords based on these β values using Algorithm 1. One of the issues with the encoding scheme is the time selection for the β coefficients. As β values vary with time, it is not clear which β values to consider. In [MSB16], the authors proposed to take the average of β values over all time samples where the attack successes. Here, we first choose two points with the highest correlation (denoted as a1 and a2 respectively) from Fig. 4. We then took the average of the β coefficients for 2 clock cycles around these points, generating two independent sets of β coefficients.

We first start by testing the reference implementation. We measured the EM traces when the device was processing the $(x_0 \oplus (\overline{(x_3 \vee x_2)}))$ operation from the `SKINNY` SBox. To compute TVLA, 25k traces for both fixed and variable plaintexts were collected. We note that since everything is computed using look-up tables, the implementation can be quite large. For instance, the `SKINNY` implementation, with 1-bit datapath, reports 6.3 kB code size and 1.5 kB memory size, and around 400k clock cycle count on AVR architecture.

Another restriction is that since we only did profiling on a single register (r16 in this case), all the computations will be restricted to and have to be performed on that register. In Table 1, it uses several registers, but since our profiling is restricted to r16, to deal with operation with two inputs, we have to keep storing and loading the operand from the memory. We can use other registers as well, but only for processing non critical data, such as storing and loading constant values. We also repeated the experiments with r17 to confirm the consistency of the results.

In Fig. 5a, we show the test result of the unprotected implementation (which is also based on a look-up table, where the table is based on the identity mapping instead of encoding function). As can be seen, there are observable leakages at different points in time (T-test value largely out of the TVLA threshold bounds). The only non-leaking part is the NOP operation (inserted after the trigger, so the trigger leakage would not affect the actual operations). For better understanding, the various operations are highlighted in the figure. Similarly, the results from the constant weight countermeasure, or the software dual rail, is also presented in Fig. 5b. The software dual-rail does not result in a major leakage reduction and thus fails in the current scenario. As from previous characterization, we know that the device does not present a perfect HW leakage, the software dual-rail countermeasure is thus ineffective.

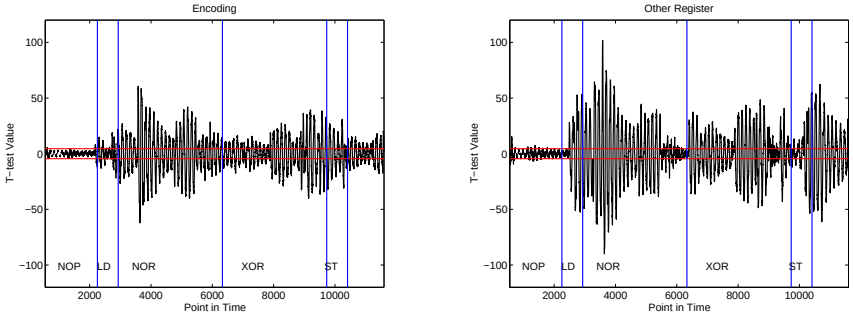
Next, we test the protected version obtained by customized encoding constructed with previously determined β coefficients values. In Fig. 6a, we provide the test results on the encoded implementation on profiled register (r16), which is based on β value approximated at average around sample a1. A reduction in leakage due to the customized encoding can be observed, however, the leakage still remains above the $\pm$ 4.5 threshold. This confirms the fact that customized encoding turns out to be more effective than software dual-rail, by taking the β coefficient into account. The leakage observed is mainly present during the NOR operation, which leaks during nibble shift operation (when swapping the data from (encoded) low nibble to (encoded) high nibble in the beginning before the NOR operation). This is an important observation. As a complex cryptographic algorithm will be composed of several distinct instructions, **it is hard to consider all the instructions in a single encoding**. As the encoding was built on profiled LD instruction, we managed to suppress its leakage compared to other instructions. These reductions can be observed from LD of the second operand (x_2), from the XOR operation, LD of the second operand (x_0) and



(a) (b)

Figure 5: TVLA on unprotected and software dual-rail. The red lines indicate the upper and lower limits of the TVLA threshold (± 4.5).

LD from the XOR LUT. However, other instructions do not follow this encoding and still continue to leak.



(a) r16 (b) r17

Figure 6: TVLA on encoding a1. The red lines indicate the upper and lower limits of the TVLA threshold (± 4.5).

Using a different register. Another important fact to consider is the use of different registers, A microcontroller contains several general purpose registers. Due to the complexity of the cryptographic operations, it is often hard to limit the code to use a single register. We analyze the customized encoding under such scenario. The customized encoding was developed based on profiling register r16. Now, we analyze the impact if this encoding uses another register for processing the data. Since another register will have some physical differences from the profiled one, the β coefficient will be different, leading to incompatibility with the constructed encoding. In Fig. 6b, we show the TVLA result when encoding is profiled for one register (r16) but used on another register (r17). A practical setting of such scenario will be when two sensitive operands in different register are operated upon, and encoding can only abide by one of them. From the figure, we can see that there are more leakages from r17 version when compared to the profiled register r16. Hence, we can say that **the encoding countermeasure requires a very specific profiling and for different registers, different encoding functions have to be constructed**. This limits the application of encoding in complex implementations.

For encoding a1, we can observe that the encoding can help to reduce the leakage, also if compared to the unprotected version, albeit with still exploitable side-channel leakage.

Using a different point in time. In the next experiment, we highlight the problem of choosing β coefficient. Recall that two different customized encodings were constructed from leakage points a1 and a2 for LD using r16. Since LD instruction leaks over several cycles, we chose a1 and a2 from different cycles and constructed two encodings by averaging β values over one clock cycle each. Previous experiments showed that encodings built over a1 provide good leakage reduction. We repeated the same experiment on an encoding constructed with a2. The TVLA results are given in Fig. 7, which shows a much higher leakage than the encoding over a1. Thus, it is crucial to select good β values, but the guidelines for selection are unknown. One dominant reason which makes one encoding more or less efficient than the others is because when a microcontroller leaks, it leaks from several resources like registers, data bus, instruction bus, memory etc. The encoding is built on a single register, so when the leakage is dominant from register r16 and the encoding is built on it, the leakage reduction is significant. However, in following instructions (a2 in this case), other resources might be leaking more than the original register, thus preventing the encoding to be effective. **For a perfect encoding countermeasure, one would need to update the encoding function from one cycle to another**. Such encoding conversion might have sensitive leakage and would come at a significant cost overhead anyway.

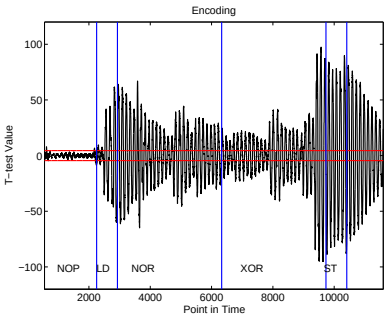


Figure 7: TVLA on encoding a2. The red lines indicate the upper and lower limits of the TVLA threshold ($\pm$ 4.5).

As the current experiments were unsuccessful to protect the cipher implementation, for the next experiments in Section 6 we will consider restricting the analysis to each individual operation. We expect to obtain a better understanding of the information leakage from the encoding countermeasure.

6 Instruction Specific Evaluation on Practical Setup

The experiments in the previous section show that when a full cipher is implemented, it is hard to balance the leakage across multiple operations. In this section, we concentrate on a single instruction to explore the effectiveness of the encoding countermeasure. In particular, we investigate the LD operation for the encoded SBox output. We use two target devices, the AVR ATmega328p mounted on a Arduino Uno (8-bit architecture, used in the previous experiments) and the ARM Cortex M3 on a Arduino Due (32-bit architecture, for testing longer encoding function). The setup used are same as previous.

6.1 On AVR 8-bit microcontroller

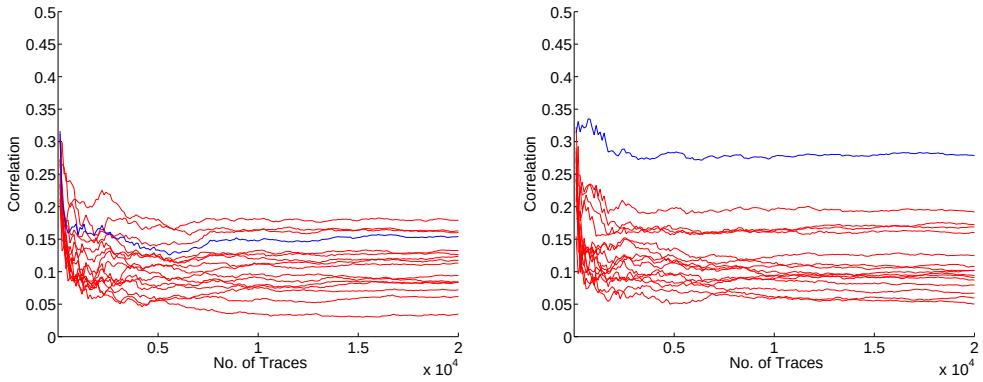
We investigate the selection method for β coefficient used to generate the codewords for the countermeasure. In [MSB16], the average is taken among different points in time during the attack. For our experiments, since we are only concerned about the profiling of LD operation, we used a similar approach.

6.1.1 Selection of β coefficients

The issues arising with the selection of appropriate β values has been already demonstrated in the previous section, with different leakages for encoding based on point around a1 and a2. Additionally, we also consider choosing only a single point for representing the leakage of the operations, which in our case is the LD operation from the SBox:

- b1: we take the mean value for all the β coefficients for LD instruction from 2 clock cycles (similar to [MSB16] and previous section), and
- b2: we take the β coefficients value from the highest correlation peak in the LD instruction.

For evaluation, we considered a simple encoding function, which maps 4 bits to 8 bits codewords. We tested it on the `PRESENT` SBox LUT (any other non-linear 4x4 SBox would do, but we had an existing `PRESENT` implemented in LUT, whereas for `SKINNY` it is implemented in terms of algebraic operation). The assembly code for the encoded implementation is similar to the one used for `AES` for profiling. However, we add an additional step: before doing a table look-up, the input is encoded and the SBox look-up is done on encoded input, while the table is updated with proper encoding to return the encoded output. The rest of the code is similar to what we did before: after the trigger, we perform 20 NOPs, LD from the memory (encoded value), 20 NOPs, and ST the (encoded) value to the memory.

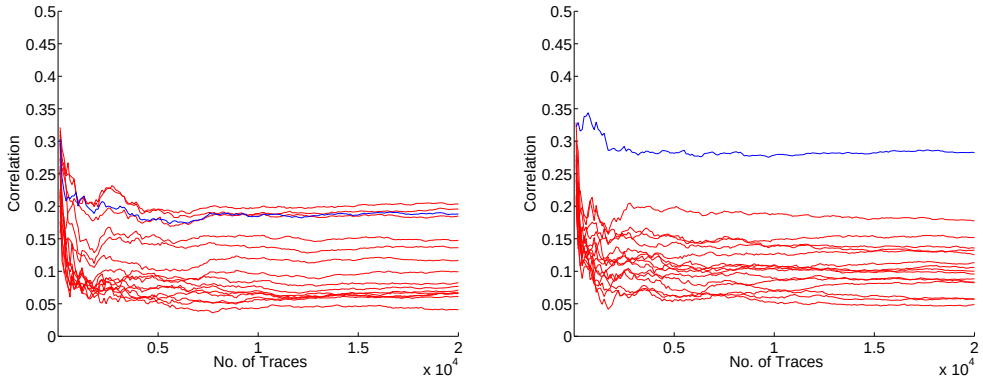


(a) encoding b1 (b) encoding b2

Figure 8: CPA on encoded traces, blue line indicating correct key hypothesis and red lines indicating the wrong key hypotheses

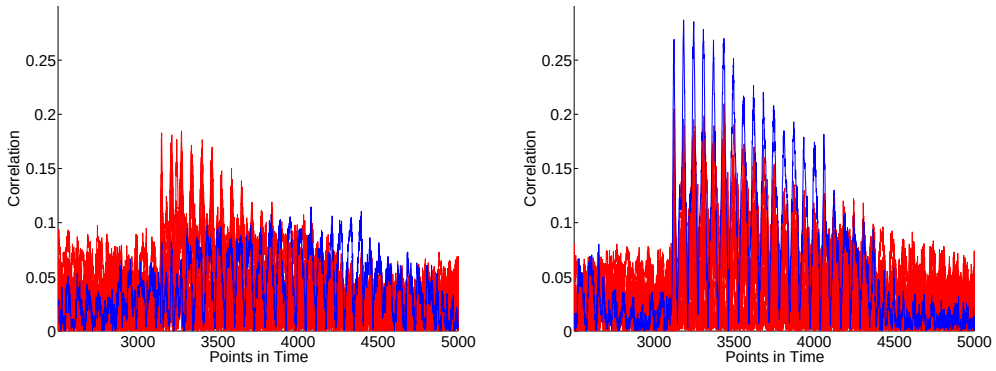
We generate 2 different encoding functions, based on 2 different β coefficient values, described earlier. These experiments are conducted based on 100k measured EM traces. For this case, we do not use TVLA for evaluation because we do not implement a sufficient number of round of the cipher and hence there is no diffusion of the data. Depending on which input is used as fixed value, the TVLA plot might be biased.

From Fig. 8, we show the CPA results for both encoding functions, with blue line indicating correct key hypothesis and red lines indicating the wrong key hypotheses. We can see that for b1 encoding function, the correct key can not be distinguished from wrong key hypotheses, whereas for b2 encoding function, the correct key can be clearly distinguished, even with very few number of traces. Hence, **the approach of using the average of β coefficient seems to be the best that we can get**. Although it works well for one instruction, it was shown in previous section that a full cipher is hard to protect.



(a) b1 (b) b2

Figure 9: Enhanced CPA on encoded traces, blue line indicating correct key hypothesis and red lines indicating the wrong key hypotheses



(a) encoding b1 (b) encoding b2

Figure 10: Enhanced CPA on encoded 100k traces, blue line indicating correct key hypothesis and red lines indicating the wrong key hypotheses

We then considered the enhanced CPA attack, which instead of using a generic HW model, relies on the β coefficients learned by the attacker using a stochastic profiling. Once the β coefficients are known, the attacker can estimate the encoding function and the underlying codewords using Algorithm 1. In this case, the enhanced CPA assumes that the attacker has knowledge of the correct β coefficients and thus a precise leakage function.

In Fig. 9, the results for enhanced CPA are shown for both encodings. We can see that the correct key still can not be distinguished from the wrong key hypothesis for b1

encoding. The results for the enhanced CPA on all 100k traces are shown in Fig. 10. For the encoding b1, the correct key does not leak during the LD operation, while encoding b2 has a dominant leakage. We do observe some residual leakage for b1, but that is suspected as some residual leakage after the LD operation.

To further evaluate this result, we performed 10 different experiments for encoding b1. Each experiment was conducted by modifying the placement of the EM probe. As the probe position changes, the leakage characteristic changes as well, which impacts β coefficients and thus the encoding functions. However, the result could not be repeated, as most of the time (6 out of 10), the correct key can be recovered with sufficient number of traces (under 100k) even if similar procedure is used for determining the β coefficients and generating the encoding function. In an extreme case, this might be seen as a drawback where designer protects using encoding derived from one location, which turns out ineffective in another location.

6.1.2 Evaluation on Different Side-Channel Information

A very common attack scenario is that where the designer deploys encoding based on EM measurement profiling and the attacker uses power or vice versa. To further investigate, we now perform a profiling using the same implementation, but with different side-channel information, namely EM and power, captured in parallel. In Fig. 11c, we can see that even with same codes and same scope setting, we observe different β coefficients for different side-channel. Moreover, from Fig. 11, we observe that, even when using proper β coefficients with respective side-channel leakage, the leakages are still present as indicated by TVLA. Thus the impact of choice of β coefficients and methodology to choose it remains an open question.

In this experiment, we used an EM probe with fairly large diameter, making the measurements global in nature. Indeed, the prime objective was to demonstrate that encoding won't hold from power to EM measurements. However, a localized EM measurement can potentially worsen the scenario.

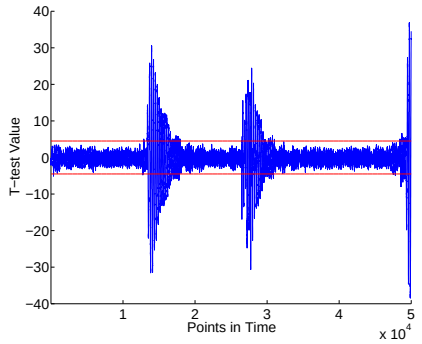
6.1.3 Verification with Leakage Detection Test

To analyse encoding b1 and b2 with TVLA methodology, we switch back to `SKINNY`. Further TVLA test are performed with 100k traces each. We target a LD operation in a chosen middle round during the encryption process (round 12). In Fig. 12, the TVLA plots for `SKINNY` LD operation are shown. In this case, similar trends can be seen, both of the encodings can reduce the leakage, however, we can see that for all the cases, the TVLA indicates that there is a first-order leakage, which might still be exploitable in different way.

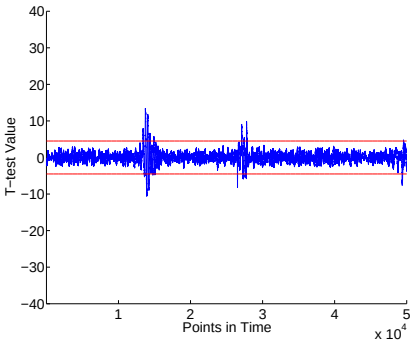
To summarize, from our experiments, it is shown that the customized encoding countermeasure might protect against CPA (or enhanced CPA), however, when checking with the TVLA, the test indicates that there is still leakage. This can be due to several factors as stated previously, like leakages from other microcontroller resource (bus leakage or address bit leakage) or the beta coefficients are not properly estimated.

6.2 On ARM 32-bit microcontroller

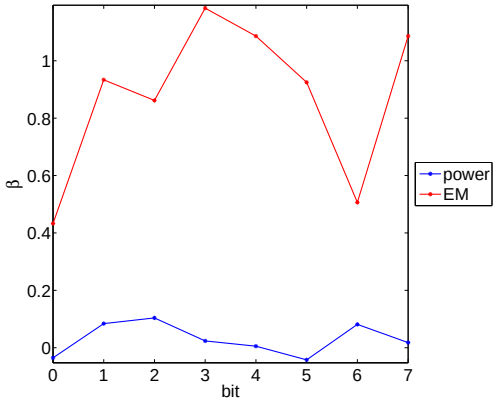
To further investigate, we analyze the encoding countermeasure on the ARM Cortex M3 on a Arduino DUE, which is a 32-bit microcontroller. As observed under simulated settings, longer encoding can further boost up the security. To evaluate the encoding countermeasure on a longer length of codewords, which was not possible on a 8-bit architecture, we switch to ARM Cortex M3. We conduct the experiments in similar manner as described earlier.



(a) TVLA EM

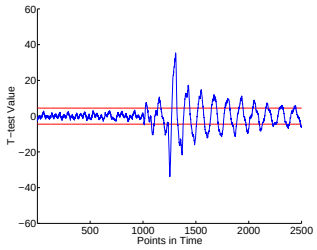


(b) TVLA Power

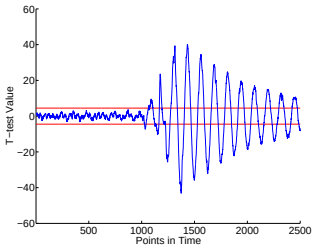


(c) β profiled EM vs power

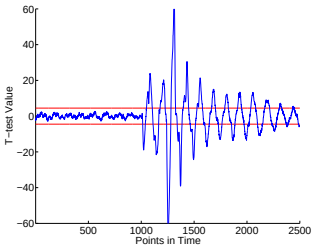
Figure 11: Leakage profiling comparison: EM vs Power. (Upper Figures) The red lines indicate the upper and lower limits of the TVLA threshold ($\pm$ 4.5). (Lower Figure) The β coefficients obtained from EM and power under the same setup.



(a) encoding b1



(b) encoding b2



(c) unprotected

Figure 12: TVLA on `SKINNY`. The red lines indicate the upper and lower limits of the TVLA threshold ($\pm$ 4.5).

We first profile the device to obtain the β coefficients by profiling during a non-linear operation.

To obtain the β for longer bit, we combine 2 SBoxes (`PRESENT` and `AES`) to get the β coefficients for 12 bits. The leakage analysis of the traces is then shown in Fig. 13. In

the figure, the CPA input denotes the leakage when loading the input from the memory, the CPA output denotes the loading of SBox output from the LUT into the register, and NICV denotes model-independent input-based observable leakages.

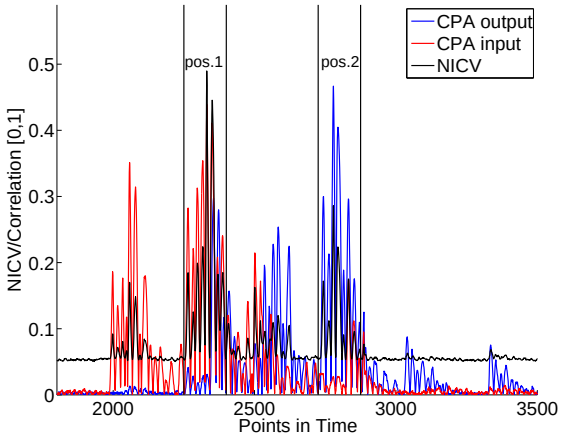
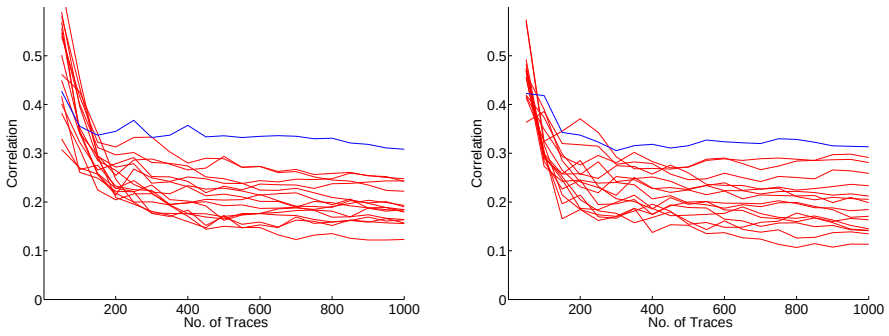


Figure 13: Profiling results on ARM, using CPA on the SBox input, CPA on the SBox input, and NICV

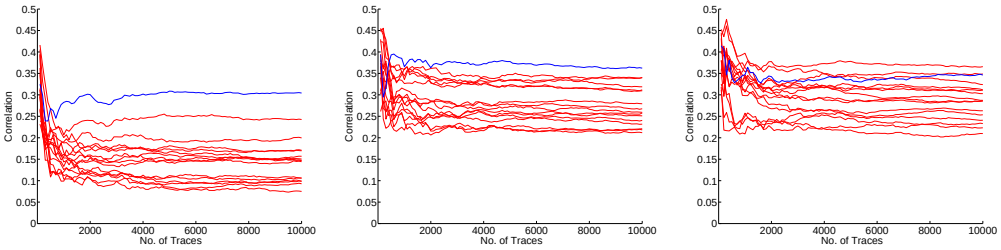
As the size of the codewords increases, the resulting encoding, decoding and the size of SBox lookup table increase as well. Due to memory limitation, only codes with length up to 10 bits can be implemented on the device. For comparison, we used an unprotected implementation, which is a standard 4-bit `PRESENT` SBox. We then chose a point of interest, denoted as the peaks in Fig. 13 at NICV peak. The encoding is obtained by averaging the β values over the clock cycle of the chosen point. We construct the codewords using the customized encoding for 2 different positions, using similar approach as before.



(a) position 1 (b) position 2

Figure 14: Basic CPA with HW model

Fig. 14 illustrate the CPA results on 32-bit ARM microcontroller, with blue line denoting the correct key and red lines denoting the wrong keys. In this experiment, we can see that basic CPA using HW model can break most of the encoding scheme, where the correct key is distinguishable, even with less than 1000 traces. This is less than the traces required for the previous attack, which is using only 8-bit codewords. As the profiling was limited to the last 12 bits due to the underlying complexity, the profiling might be



(a) unprotected (b) customized encoding (c) HO customized encoding

Figure 15: Basic CPA with HW model assuming HO leakage, blue line indicating correct key hypothesis and red lines indicating the wrong key hypotheses

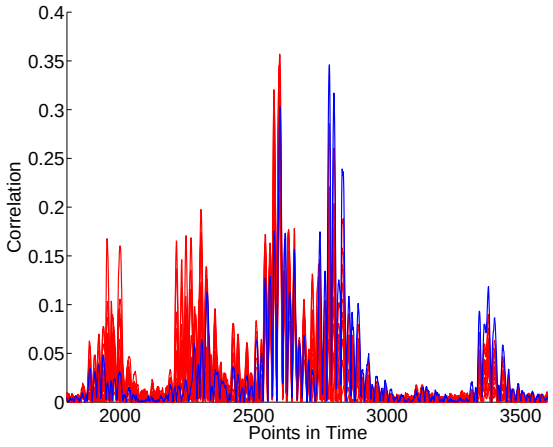


Figure 16: CPA on HO encoding with 100k traces, blue line indicating correct key hypothesis and red lines indicating the wrong key hypotheses

inaccurate. However, it is hard to perform the exact 32 bits due to high complexity.

6.2.1 Analysis on Higher Order Leakage

In [HKSS12], authors report leakage from higher order (HO) β coefficients, which can arise from coupling of individual bits. Neglecting such higher-order leakage can reduce the precision of profiling. To investigate, we perform the stochastic approach with higher order model, removing the assumption that the leakage for each bit is independent of each other. Here, we only observe the leakage at the NICV peak, and we construct the codewords for standard and higher order stochastic (upto degree 5, $x_{i_1}...x_{i_5}$). Degree 5 means considering impact of individual bits, pair wise, up to coupling of any 3, 4 or 5 bits together. Fig. 15 shows the CPA results, where we can see that in the encoding based on HO model, the correct key cannot be distinguished, as compared against basic customized encoding. However, when looking at CPA trace computed with all 80k traces as in Fig. 16, we can see that there is a ghost peak, but the attacker still only need to guess 1 more key candidate in order to guess the correct key. Hence, even HO-encoding based countermeasure cannot necessarily prevent SCA leakage.

7 Discussion

From our experimental results, the general observation is that the side-channel leakage will mostly be present even if the encoding scheme from [MSB16] is applied. A basic CPA with HW leakage model proves to be sufficient for exploiting the leakage. In the worst case, the correct key may not be top ranked, but it is always observed in the top key candidates. Hence, key enumeration methods [VCGRS12] can be exploited for key retrieval. The presence of this side-channel leakage can be explained by a few possible issues, such as:

- The main problem is the estimation of β coefficients. The initial recommendation was to take the average of the β from the period where the attack occurs [MSB16]. However, from the experiments, it can be observed that even if the profiling is done following this advice, the implementation still shows some residual leakage. Thus, one major question is how to exactly determine the β value when constructing the look-up table for the operations. It might be attributed to the non-linear leakage, as highlighted in the experiments earlier, where different bits are affecting each other, or the bit transitions within the clock cycle may also be affecting the estimation. Hence, the encoding requires a very strong assumption regarding the leakage, which might not be practically achievable.

- It is hard to obtain a generic encoding which could protect against side-channel leakage for different operations from different leakage sources. For a single instruction such as LD, we show that this might be achievable on the profiled location. However, from the experiment, we can see that the profiling done on the LD operation can be broken by attacking other operations, such as NOR. Similarly, based on further investigations, we show there could be leakage for other cases. It might be that the β value was changing during different operations (which means that for different point in time, different look-up tables had to be used). In theory, one could use a dynamic encoding, which would adjust the encoding based on different leakage point. However, one will have to design different look-up tables with every change of instructions, data registers and memory address. Moreover, every conversion from one encoding to another will have overhead. Thus,this will be very expensive and inefficient. From the last experiment, we can see that even if the leakage is profiled correctly using the proposed algorithm in a particular location, simply by moving the measurement channel (power to EM), the encoding will totally change, rendering the protection ineffective.

Although the idea of customized encoding is sound and works well under simulated environment, there are several practical assumptions which are not easily achievable. For the encoding to be implemented, it needs to be done using look-up table. However, a look-up table implementation has been shown to be prone against cache timing attack [Ber05]. Similarly, since everything has to be done using a look-up table, with the growing length of the encoding, the memory required as well as the execution time will be increased as well. In our experiment with `SKINNY` (which is supposed to be lightweight), we need around 400k clock cycles for each encryption and 1.5 kB for RAM, which make the implementation impractical and inefficient for many applications.

One of the advantages of the encoding scheme is explained in [BH17], where the authors show that the encoding provides resistance against fault attack along with (theoretical) side-channel protection. They construct efficient codes that are capable of protecting the underlying computation. They also provide practical experiments, showing noticeably better fault resistance properties. Another potential application of customized encoding is to be used as a noise generator for masking countermeasures as in mentioned in [KdSP+16].

However, at high β coefficients variance, short encoding length or dual-rail might be as leaky as unprotected (see Fig. 1), due to the hardness to balance the leakage.

In summary, the issues encountered in the customized encoding are based on precise relation between β coefficients and actual device physics, which is a high-level estimation at best. The estimation works well for exploitation. However, for protection, one of the main issue is the selection of β coefficients for the encoding. More understanding in this direction may help to design better codes in the future.

8 Conclusion

In this paper, we have investigated the customized encoding from [MSB16] proposed as software countermeasure for side-channel attacks. We shown through simulations that assuming the leakage is properly characterized, the encoding function can be resistant against first order attack. However, as shown by our practical experiments, several assumptions will not hold true rendering the protection ineffective. Some of the issues identified were related to temporal and spatial variance of β values, due to which an effective encoding is hard to construct. We have also implemented a full cipher using customized encoding and demonstrated the practical shortcomings of the countermeasures on two different microcontroller targets, demonstrating several test cases. Further investigation on relationships between β values and device physics could help improve encoding based countermeasures.

Acknowledgements

The authors would like to thank the anonymous referees for their helpful comments. The third author is supported by Temasek Labs (DSOCL16194).

References

[BCO04] Éric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In *CHES*, volume 3156 of *LNCS*, pages 16–29. Springer, 2004. Cambridge, MA, USA.

[BDGN13] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. Nicv: Normalized inter-class variance for detection of side-channel leakage. Cryptology ePrint Archive, Report 2013/717, 2013. http://eprint.iacr.org/2013/717.

[Ber05] Daniel J. Bernstein. Cache-timing attacks on AES, 2005.

[BH17] Jakub Breier and Xiaolu Hou. Feeding two cats with one bowl: On designing a fault and side-channel resistant software encoding scheme. In *RSA Conference Cryptographer Track (CT-RSA 2017)*, pages 1–18, Feb 2017.

[BJB16] Jakub Breier, Dirmanto Jap, and Shivam Bhasin. Mistakes are proof that you are trying: On verifying software encoding schemes' resistance to fault injection attacks. Cryptology ePrint Archive, Report 2016/932, 2016.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The skinny family of block ciphers and its low-latency variant mantis. Cryptology ePrint Archive, Report 2016/660, 2016.

[CESY14] Cong Chen, Thomas Eisenbarth, Aria Shahverdi, and Xin Ye. Balanced Encoding to Mitigate Power Analysis: A Case Study. In *CARDIS*, Lecture Notes in Computer Science. Springer, November 2014. Paris, France.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation, September 2011. NIST Non-Invasive Attack Testing Workshop.

[GP99] Louis Goubin and Jacques Patarin. DES and Differential Power Analysis. The "Duplication" Method. In *CHES*, LNCS, pages 158–172. Springer, 1999. Worcester, MA, USA.

[HDD11] Philippe Hoogvorst, Jean-Luc Danger, and Guillaume Duc. Software Implementation of Dual-Rail Representation. In *COSADE*, 2011. Darmstadt, Germany.

[HKSS12] Annelie Heuser, Michael Kasper, Werner Schindler, and Marc Stöttinger. A new difference method for side-channel analysis with high-dimensional leakage models. In Orr Dunkelman, editor, *Topics in Cryptology - CT-RSA 2012 - The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 - March 2, 2012. Proceedings*, volume 7178 of *Lecture Notes in Computer Science*, pages 365–382. Springer, 2012.

[Jea16] Jérémy Jean. TikZ for Cryptographers. https://www.iacr.org/authors/tikz/, 2016.

[KdSP+16] Dina Kamel, Guerric de Streel, Santos Merino Del Pozo, Kashif Nawaz, François-Xavier Standaert, Denis Flandre, and David Bol. Towards Securing Low-Power Digital Circuits with Ultra-Low-Voltage Vdd Randomizers. In Claude Carlet, M. Anwar Hasan, and Vishal Saraswat, editors, *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*, volume 10076 of *Lecture Notes in Computer Science*, pages 233–248. Springer, 2016.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards.* Springer, 2007.

[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 Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 223–243. Springer, 2016.

[PRB09] Emmanuel Prouff, Matthieu Rivain, and Régis Bevan. Statistical analysis of second order differential power analysis. *IEEE Trans. Computers*, 58(6):799–811, 2009.

[RGN13] Pablo Rauzy, Sylvain Guilley, and Zakaria Najm. Formally Proved Security of Assembly Code Against Leakage. *IACR Cryptology ePrint Archive*, 2013:554, 2013.

[SGV08] François-Xavier Standaert, Benedikt Gierlichs, and Ingrid Verbauwhede. Partition vs. Comparison Side-Channel Distinguishers: An Empirical Evaluation of Statistical Tests for Univariate Side-Channel Attacks against Two Unprotected CMOS Devices. In Pil Joong Lee and Jung Hee Cheon, editors, *Information Security and Cryptology - ICISC 2008, 11th International Conference, Seoul, Korea, December 3-5, 2008, Revised Selected Papers*, volume 5461 of *Lecture Notes in Computer Science*, pages 253–267. Springer, 2008.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 30–46. Springer Berlin Heidelberg, 2005.

[TV04] Kris Tiri and Ingrid Verbauwhede. A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In *DATE'04*, pages 246–251, 2004. Paris, France.

[VCGRS12] Nicolas Veyrat-Charvillon, Benoît Gérard, Mathieu Renauld, and François-Xavier Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. In *International Conference on Selected Areas in Cryptography*, pages 390–406. Springer, 2012.