cryptography

MDPI

Article

CONFISCA: An SIMD-Based Concurrent FI and SCA Countermeasure with Switchable Performance and Security Modes

Ehsan Aerabi [1,2], David Hély [2,*], Cyril Bresch [2], Athanasios Papadimitriou [2,3] and Mahdi Fazeli [1,*]

[1] School of Computer Engineering, Iran University of Science and Technology, Tehran 16846-13114, Iran; e_aerabi@comp.iust.ac.ir
[2] Grenoble INP, Laboratoire de Conception et d'Intégration des Systèmes (LCIS), Université Grenoble Alpes, 26902 Valence, France; cyril.bresch@lcis.grenoble-inp.fr (C.B.); thanospap@unipi.gr (A.P.)
[3] Department of Informatics, University of Piraeus, 18534 Piraeus, Greece
* Correspondence: david.hely@lcis.grenoble-inp.fr (D.H.); m_fazeli@iust.ac.ir (M.F.)

Abstract: CONFISCA is the first generic SIMD-based software countermeasure that can concurrently resist against Side-Channel Attack (SCA) and Fault Injection (FI). Its promising strength is presented in a PRESENT cipher case study and compared to software-based Dual-rail with Pre-charge Logic concurrent countermeasure. It has lower overhead, wider usability, and higher protection. Its protection has been compared using Correlation Power Analysis, Welch's T-Test, Signal-to-Noise Ratio and Normalized Inter-Class Variance testing methods. CONFISCA can on-the-fly switch between its two modes of operation: The High-Performance and High-Security by having only one instance of the cipher. This gives us the flexibility to trade performance/energy with security, based on the actual critical needs.

Keywords: hardware security; side channel attacks; fault injection; countermeasure; SIMD; NEON

Citation: Aerabi, E.; Hély, D.; Bresch, C.; Papadimitriou, A.; Fazeli, M. CONFISCA: An SIMD-Based Concurrent FI and SCA Countermeasure with Switchable Performance and Security Modes. *Cryptography* **2021**, *5*, 13. https://doi.org/10.3390/cryptography5020013

Academic Editor: Jim Plusquellic

Received: 26 February 2021
Accepted: 26 April 2021
Published: 6 May 2021

1. Introduction

Side-Channel Analysis (SCA) and Fault Injection (FI) are two different hardware attacks which can reveal secret information of sensitive digital devices (e.g., secret keys). SCA exploits the device side channel leakage (e.g., electromagnetic emanation) along with statistical approaches such as Differential or Correlation Power Analysis (DPA & CPA) to reveal the secret key [1] FI can expose the secret information by injecting faults during the computation and observing the erroneous outputs [2]. One of the main targeted secrets are cryptography keys. An exposed crypto key reveals encrypted messages and compromises the secrecy. Therefore, historically, SCA and FI attacks and countermeasures have been developed around cryptography case studies.

Here, we target software crypto implementation on embedded devices which are built around a microcontroller unit (MCU). Mobility of embedded systems (and Internet-of-Things devices) easily and frequently exposes them to these hardware attacks. Attackers have more opportunity to gain access to a portable device and mount SCA or FI and find stored secret keys. For instance, an attacker may gain access to a given IoT device, and then try to communicate with it, which in turn runs the embedded cryptography software. In the meantime, he captures power consumption (or electromagnetic emanation) of the device or tries to inject faults at runtime to reveal the secret key. These ones can further be used in order to expose proprietary firmware or confidential data.

Applying separate countermeasures on software crypto against SCA and FI may induce costly system complexity and unacceptable performance overhead, for constrained embedded systems. Moreover, FI countermeasures and SCA countermeasures do not work well together, FI countermeasures generally need data or execution redundancy of the secret computation, and this redundancy can potentially add more side channel

leakage and aggravates SCA protection [3,4]. Only a few *concurrent* (or combined) software countermeasures that can resist both SCA and FI have been proposed so far for that [5].

In this paper, we propose CONFISCA, a generic secure software implementation methodology against SCA and FI leveraging Single Instruction Multiple Data (SIMD) parallel computation. We have evaluated its strength against Electro-Magnetic (EM) CPA. CONFISCA is a continuation of our previous work [6] on concurrent FI and SCA protection for MCUs without SIMD features. CONFISCA presents more applicability, performance and protection by employing SIMD on bigger MCUs.

Energy constrained embedded devices may not always have enough energy to fulfill real-time tasks and enforce security. Therefore, they require a critical capability to trade energy/performance with security based on the actual energy budget and performance needs. This generally implies to have two implementations on the device: performance implementation and secure implementation, which poses storage and runtime overheads. The CONFISCA approach has only one implementation, which presents both performance and security preferences without switching overhead. Its performance overhead will be compared to the previous work.

The contribution of the paper is as follows:

- The paper proposes a software-based cryptography implementation method to counteract against both SCA and FI in embedded systems that benefit from SIMD features. To the best of our knowledge, this is the first use of SIMD for concurrent protection against SCA and FI.
- The proposed method has higher applicability and lower memory and performance overhead, in comparison to the related concurrent software countermeasures against SCI and FI.
- By having only one piece of software, the proposed method can disable the FI and SCA resistance in exchange of performance gain with a negligible overhead, which gives the flexibility to trade security with the energy consumption or performance.
- In a cipher case study, the resistance of the proposed method against SCA and FI has been evaluated and compared to the other related work using different evaluation techniques.

In Section 2, we cover related work, mainly DPL and encoding countermeasures. Section 3 explains the CONFISCA method, followed by a case study and its evaluation in Sections 4 and 5, and, finally, this paper finishes in Section 6 with conclusions.

2. Related Work

Two main threads of the concurrent software countermeasures against SCA and FI are *DPL-based* and *Encoding-based* methods. Both categories suffer from code or memory size explosion and performance degradation. The *DPL-based* group employs a software equivalent of Dual-rail with Pre-charge Logic (DPL) [7]. A dual bit with the opposite Boolean value is always stored and processed to neutralize the leakage of the original bit on the side channel. The software implementation was first proposed in [8] and followed by [9]. Figure 1 illustrates the software-DPL presented in [8]. It performs an arbitrary operation on 1-bit operands (2-bit in total with its opposite bit) O_1 and O_2 which are loaded in the steps 2 and 5, just after pre-charging CPU registers in the step 1 (for O_1) and 4 (for O_2). O_1 and O_2 both have a complementary bit, each of which have 2 bits stored in R_1 and R_2 and are combined in the step 6; then, the 4-bit value is used to look-up the result from a table in the steps 8 and 9. The last step is to clean the registers and data bus in a secure way.

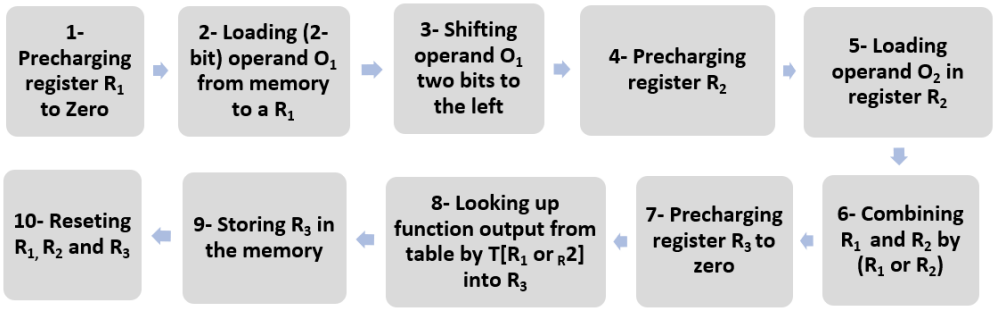

Figure 1. Software-DPL presented in [8].

Since the DPL-based approach can compute only one bit per iteration, their proposed approach is expensive both on performance and code size. Code size overheads stem from the fact that DPL fetches the outputs of the cipher's sub-functions from a look-up table, which grows exponentially with the function input size. These overheads are variable depending on the chosen cipher. For instance, DPL requires only 256 bytes of memory to secure 4-bit S-Box of the PRESENT cipher (instead of 16 bytes for the unprotected one). However, for the 8-bit AES Sbox, it requires 128 kB instead of 256 bytes.

The *Encoding-based* concurrent countermeasures use specific encoding with constant Hamming weights to, in theory, eliminate the secret leakage. The authors in [5] propose an encoding-based concurrent countermeasure based on the work presented in [10]. For instance, in [10], a 4-bit S-box operation $s_3\ s_2\ s_1\ s_0 = S(r_3\ r_2\ r_1\ r_1)$ becomes an 8-bit operation $s_3\overline{s_3}s_2\overline{s_2}s_1\overline{s_1}s_0\overline{s_0} = S(r_3\overline{r_3}r_2\overline{r_2}r_1\overline{r_1}r_0\overline{r_0})$, in which each bit accompanied by its complement bit which generates constant Hamming weight operations. The lookup table size will be 2^8 instead of 2^4. Besides exponentially larger lookup tables (as for DPL), encoding and decoding is a long process; hence, the encoding approaches also suffer from performance and memory overheads.

CONFISCA shares the same constant-hamming-weight idea but with lower overhead and theoretically all one-to-one functions can be implemented using the proposed approach.

Finally, some masking concurrent works were published recently [11]. They have targeted hardware implementation since all crypto subfunctions (e.g., Sbox, Add Round Key, etc.) are broken into many bit-wise operations, making it impractical and extremely slow for software. They are still vulnerable to higher-order attacks although adding more shares can hinder attackers by forcing them to target more intermediate values at the cost of more circuits. Nevertheless, we do not compare them here since there is no concurrent software countermeasure to the best of our knowledge.

There is a track of memory-based protection SCA countermeasures for FPGA [12,13] which use T-Box AES implementation [14]. The main idea is to implement a crypto's subfunctions using lookup tables in a way that the complement values of the output are being lookup in parallel with the original output. This yields in *low-entropy* output. Their principle is like the CONFISCA protection, but they are designed to be implemented on FPGAs since they need customized parallel lookup tables based on random signals [12]. Nevertheless, they require four times bigger lookup tables (in contrast, CONFISCA needs twice bigger tables) and, in addition, they cannot defend FI (contrary to CONFISCA).

3. CONFISCA—The Proposed Countermeasure

SIMD is aimed to boost the performance leveraging parallel computation. SIMD feature is a hardware-assisted execution that performs an operation on a vector of inputs simultaneously in parallel. SIMD is a performance improvement feature and exists nowadays in different types of processors (namely ARM families) and including embedded MCUs. SIMD comes with some specific atomic machine instructions that runs specific tasks (addition, multiplication, memory lookup) on a vector of inputs at the same time. We make use of a parallel memory lookup instruction which accepts a vector of indexes (addresses) pointing to the values stored in a table, and simultaneously retrieves the values

as a vector. The parallel lookup gives us the possibility to develop a concurrent protection scheme against FI and SCA as described below.

SCA protection is achievable by having balanced Hamming weights that implies constant number of zeros and ones being written on CPU register or memory. This *theoretically* generates data-independent power consumption or EM emanation during register/memory writes. FI protection is achievable by having duplication in data and computation. By comparing the duplicates, any attempts to manipulate the computation by faults are detectable (except exactly the same faults on two duplicates). SIMD operation gives the ability to have both complemented value (for SCA protection) and duplication (for FI protection) in parallel. For small processors without SIMD features, we developed the same concept in a concurrent countermeasure [6]. Now, by introducing CONFISCA, we extend the same idea for processors with SIMD in a more secure and efficient solution.

Assume an SIMD two-value look up instruction. It accepts a vector like (X_1, X_2) and returns ($T[X_1]$, $T[X_2]$), where $T[X]$ represents the corresponding value in the table T having the index X. We denote the operation like:

$$LU_{SIMD}(X_1, X_2) = (T[X_1], T[X_2]) \tag{1}$$

Let us assume that cipher $\Omega(P)$ is a sequence of k distinct functions F_i (*i from 1 to k*) which all in turn process the plaintext P to have the cipher-text C (e.g., *Sub-Bytes, Mix-Column, Shift-Rows & Add-Round-Key in AES*). We can write $\Omega(P)$ as:

$$\Omega(P) = F_k(F_{k-1}(\ \ldots\ F_1(P)..)) = C \tag{2}$$

To implement F_i, we fill up the look-up table with all the output values of F_i and their complements. If F_i is an n-bit function, its output values would range from 0 to 2^n. In order to fit the original output values and their corresponding complements in a look-up table, we need $2 \times 2^n = 2^{n+1}$ entries in the table. The look-up table has two halves (Figure 2). The first half (blue area) has the original outputs of the F_i and the second half (red area) has the corresponding complementary outputs. The indexes of the table are n + 1 bits long. Then, on each execution (look-up), we submit input X along with its complements $\overline{X}$ as a vector, in a way that the output value $T[\overline{X}]$ has the logical complement value of *T[X]*. In brief:

$$F_i^{SIMD}(X,\ \overline{X}) = (T[X], \overline{T[X]}) = \left(F_i(X), \overline{F_i(X)}\right) \tag{3}$$

The resulting output vector has a constant Hamming weight, which theoretically is SCA resistant. Holding property (3) throughout the cipher, the output of F_i which is $\left(F_i(X), \overline{F_i(X)}\right)$ could be used *directly* as the input vector for the next function F_{i+1}, without any modification and, subsequently, we prevent theoretically any leakage for the intermediate values. The main difficulty is how to arrange the values so that the property (3) holds for all values of X and $\overline{X}$.

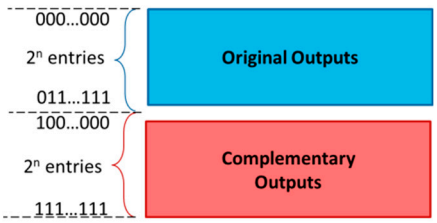

Figure 2. Memory structure for the protected look-up.

Let us represent the n-bit input X of F_i by $x_n x_{n-1} \ldots x_1$. Then, in the address of $0 \mid X$ or $0x_n x_{n-1} \ldots x_1$ which is located in the first half part of the table, we store $F_i(X)$ along with a zero in its Most Significant Bit (MSB), noted by $0 \mid F_i(X)$. Then, in the second half

part of the table in address $1 \mid \overline{X}$ or $1\overline{x}_n\overline{x}_{n-1} \ldots \overline{x}_1$, we store $1\left|\overline{F_i(X)}\right.$. Finally, if we look-up $0|X$, we would have $0 \mid F_i(X)$, while when we look-up $1 \mid \overline{X}$, we would have $1\left|\overline{F_i(X)}\right.$:

$$F_i^{SIMD}(0|X,\ 1|\overline{X}) = LU^{SIMD}(0|X,\ 1|\overline{X}) = (T[0|X], T[1|\overline{X}]) = \left(0|F_i(X), 1|\overline{F_i(X)}\right) \quad (4)$$

The term $\left(0|F_i(X), 1|\overline{F_i(X)}\right)$ as the SIMD output has constant Hamming weight (n + 1) and can be used directly as the input for the next function of the cipher Ω which is F_{i+1}. Hence, we simply cascade all k functions of cipher Ω without any modification on the intermediate values. The SCA secure cipher Ω^{Secure} will be:

$$\Omega^{\text{Secure}}(\text{P}) = F_k^{SIMD}\left(F_{k-1}^{SIMD}\left(\ldots\ F_1^{SIMD}1\,(0|\text{P},\ 1|\overline{P})..\right)\right) = (0|\text{C},\ 1|\overline{C}) \quad (5)$$

which indicates that $0 \mid \text{P}$ and $1 \mid \overline{P}$ go directly through k look-up tables, on each of which the Hamming weight of the intermediate values are constant and SCA secure.

4. A PRESENT Case Study

The proposed generic approach is applied on the PRESENT cipher [15]. Choosing PRESENT has two reasons: firstly, to have comparable cases with the other related methods which have used either of PRESENT or Prince ciphers. Secondly, PRESENT uses smaller 4-bit data path and a 4-bit S-box lookup table (in comparison to 8-bit S-box in AES) and, therefore, the countermeasure generates relatively smaller lookup tables for each subfunction $\left(2 \times 2^{4-bit} = 32 \text{ table entries for each subfunction}\right)$. It should be noted that CONFISCA needs tables two times bigger than the original cipher's data-path (as discussed earlier), while DPL needs exponentially bigger tables ($2^{2\times4} = 256$ table entries for each subfunction for PRESENT). Therefore, it is possible to apply CONFISCA for common modern ciphers like AES by having $2 \times 2^{8-bit} = 256$ table entries for each subfunction, but the same cipher with DPL needs $2^{2\times8} = 32K$ table entries for each subfunction which is costly for embedded systems.

PRESENT has three functions: AddRoundKey, S-BoxLayer, and pLayer, which are repeated 31 times to produce the cipher. We chose to secure the two first functions as a proof of concept and also because they need smaller tables. It is feasible to break pLayer into some sub-functions [16] and protect them using CONFISCA. This will be a future work of this research. A quite similar implementation (with different table values) can be used for other ciphers with AddRoundKey and SBoxLayer like AES. Since AES S-BOX is an 8-bit function, we need a SIMD table two times bigger than PRESENT.

4.1. PRESENT S-Box Layer and AddRoundKey

S-Box in PRESENT is a 4-bit function. The original s-box table has 16 entries: {12, 5, 6, 11, 9, 0, 10, 13, 3, 14, 15, 8, 4, 7, 1, 2}. For the first entry: $F_{SBoX}(0) = T[0] = 12 = 1100$. Following the CONFISCA method, we have:

$$F_{SBOX}^{SIMD}(0|0000,\ 1|1111) = (T[0|0000], T[1|1111]) = (T[0], T[31]) = (0|1100, 1|0011) = (12, 19) \quad (6)$$

This implies that, in the addresses of 0 and 31, we must store 12 and 19, respectively. Table 1 presents all the 32 values of the S-Box in PRESENT. AddRoundKey is simply a 4-bit XOR operation between the input and the key. The complete AddRoundKey look-up table is shown in Table 2 for the key value of 1010.

Table 1. PRESENT S-box Layer protected memory content.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
01100	00101	00110	01011	01001	00000	01010	01101
8	**9**	**10**	**11**	**12**	**13**	**14**	**15**
00011	01110	01111	01000	00100	00111	00001	00010
16	**17**	**18**	**19**	**20**	**21**	**22**	**23**
11101	11110	11000	11011	10111	10000	10001	11100
24	**25**	**26**	**27**	**28**	**29**	**30**	**31**
10010	10101	11111	10110	10100	11001	11010	10011

Table 2. PRESENT AddRoundKey protected memory content.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
01010	01011	01000	01001	01110	01111	01100	01101
8	**9**	**10**	**11**	**12**	**13**	**14**	**15**
00010	00011	00000	00001	00110	00111	00100	00101
16	**17**	**18**	**19**	**20**	**21**	**22**	**23**
11010	11011	11000	11001	11110	11111	11100	11101
24	**25**	**26**	**27**	**28**	**29**	**30**	**31**
10010	10011	10000	10001	10110	10111	10100	10101

There are two security concerns associated with the secure key: (1) as the usual crypto key storage, the same storage memory protection should be applied (only) to the AddRoundKey table. The only difference is between their sizes. For this case, the PRESENT key is 80 bits and the AddRoundKey table is $32 \times 5 = 160$ bits for each round. (2) online key renovation can leak the table values, but since it is done once per a new key, the amount of leakage is not enough to reveal the key.

CONFISCA can easily switch from the protected ("High-Security") to the unprotected ("High-Performance") mode by removing the '1' on the MSB of the complementary instances. Then, all the lookups are fetched from the original values (and not the complementary one). This way, we have a double performance gain.

We have implemented the CONFISCA countermeasure on a Xilinx Zybo Zynq-7000 ARM/FPGA SoC board. This SoC board is built around a Xilinx 7-series field programmable gate array (FPGA) and an ARM Cortex-A9 working on 650 MHz. The embedded ARM Cortex-A9 processor embeds the NEON SIMD architecture extension. Using NEON, we implemented the two modes:

4.2. The Performance (Unprotected) Mode

NEON extension for table look-up includes eight parallel look-up operations from a 32-byte table. First, to have an *unprotected* cipher, we utilized the extension and developed a vectorized version of AddRoundKey and SBox of the PRESENT cipher which executes eight parallel instances of the cipher algorithm. Figure 3A illustrates the eight look-up registers which contain eight different data from eight different instances of the cipher.

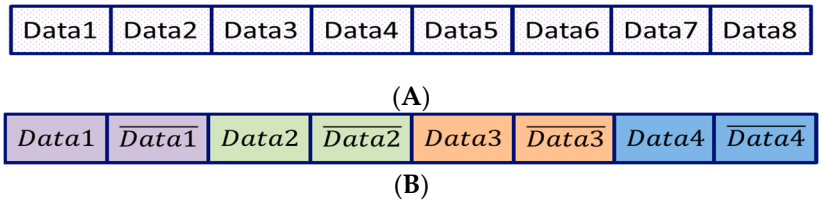

Figure 3. Data configuration in SIMD table look-up. (**A**) high-Performance (Unprotected) Mode. —eight bytes computation in parallel; (**B**) high-security (protected) mode—four parallel computations with their complements.

4.3. The High-Security Mode

We can apply the CONFISCA method on four pairs of the NEON SIMD look-up. Figure 3B shows the structure. Each pair is shown in a different color. Each instance of the data is accompanied by its complementary value shown with over-bar (e.g., *Data* & $\overline{Data1}$) and explains that the CONFISCA method is enabled. Therefore, in the high-security mode, we have four instances of the cipher.

5. Countermeasure Evaluation

In this part, we cover the SCA and FI analysis of the CONFISCA countermeasure. The analysis focuses on AddRoundKey and sBox layers of the PRESENT cipher.

5.1. SCA Analysis

The authors aimed to conduct several SCA methodologies against CONFISCA and the related work, DPL. We chose SCA analysis methods from two main SCA categories: *Evaluation-based* testing and *Conformance-based* testing [17,18]. Evaluation-based testing includes all SCA methods exploiting the device's side-channel leakage in order to find the secret key, while conformance-based methods try to find *any* data-dependent leakage by observing the correlation between input/output of the device and its side-channel information [18].

The success of evaluation-based testing depends on whether its power/EM model can effectively emulate the real power consumption or electronic emanation of the device. Hence, it is prone to a false negatives report. In other words, an evaluation-based testing may state that a test is not able to find the key, while there exists a leakage that needs another power model to be exploited.

On the other hand, conformance-based testing may generate false-positive results. It may state that there exists a considerable leakage from device, while no practical power/EM model is available to exploit the leakage.

Nevertheless, we chose electromagnetic correlation analysis (EM-CA) [19] as one of the most common evaluation-based testing methods and Welch's T-Test, SNR, and NICV from conformance-based testing methods [18].

We used HackMyMCU [20], a precise power and EM acquisition board designed in our laboratory to gather EM traces from the previously mentioned Xilinx Zybo SoC acquired on 5 GS/Sec. It has a precise 32-bit power/EM acquisition analog to digital convertor and amplifiers to increase Signal-to-Noise Ratio.

5.2. Correlation Power Analysis

The CPA targets the correct 4-bit key on each look-up table operation of the PRESENT cipher described in [19].

First, we implemented a *non*-SIMD version of the unprotected cipher implemented using simple XOR (in AddRoundKey) and look-up table (in sBox). The PRESENT cipher is broken between 2 K to 3 K traces, when the correlation coefficient of the correct key (red curve) stands out among the other key hypotheses as shown in Figure 4. This number forms a basis for our comparison between all series of experiments.

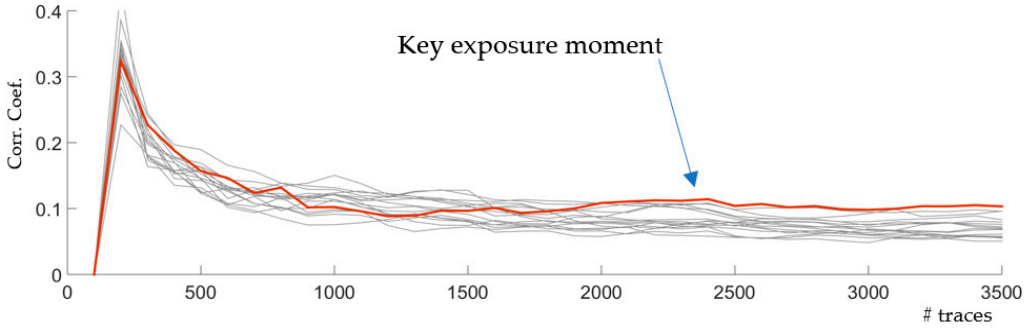

Figure 4. EM analysis against the non-SIMD PRESENT cipher.

Figure 5 illustrates the CPA results of DPL countermeasure from [9]. The authors mentioned that, in comparison with the unprotected, DPL resisted for 34 times more traces in their evaluations. Since the unprotected in Figure 4 resists until 2 K–3 K traces, we expected to have 34 times more protection for DPL (between 60–102 K). This complies with the results in Figure 5.

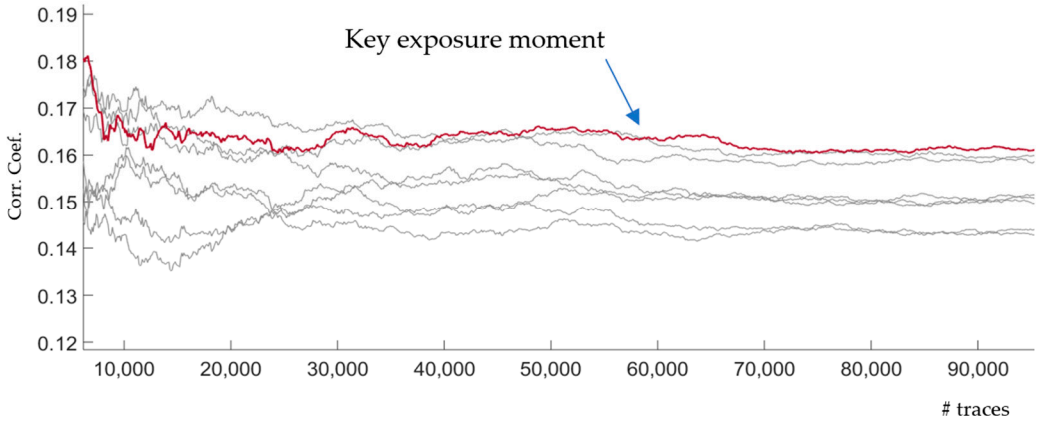

Figure 5. EM analysis against DPL countermeasure [9].

The CPA results for the SIMD implementation are shown from Figure 6A to Figure 6D. On all figures, the red curve indicates the correct key guess. Figure 6A illustrates SCA on SIMD implementation without the CONFISCA countermeasure. Moving from the non-SIMD implementation to SIMD, we can observe that SCA needs around 3000 to 4000 traces to break the cipher. Figure 6B shows the results of a SCA against the countermeasure. After acquiring 40 K traces, we were not able to find the key, and the red line is hidden. Therefore, we decided to mount stronger attacks using the averaging feature on our oscilloscope.

We set the oscilloscope to average the last 2048 traces in order to reduce the capturing noise. Figure 6C shows the SCA against the unprotected SIMD implementation. Obviously, the correlation coefficients are higher (around 0.7) than normal acquisition (around 0.1). The key stands out after the first 10 or 20 acquisitions (2048 averaged traces each. Using the same averaging approach, Figure 6D illustrates the SCA against the protected SIMD mode. It should be noted that, in Figure 6D, we let the SCA continue until 4 K $\times$ 2048 $\approx$ 8 M traces on 5 GS/s to experimentally test the strength of CONFISCA. However, the correct key is well hidden among the other keys and does not seem to stand out after this order of magnitude of captured EM traces.

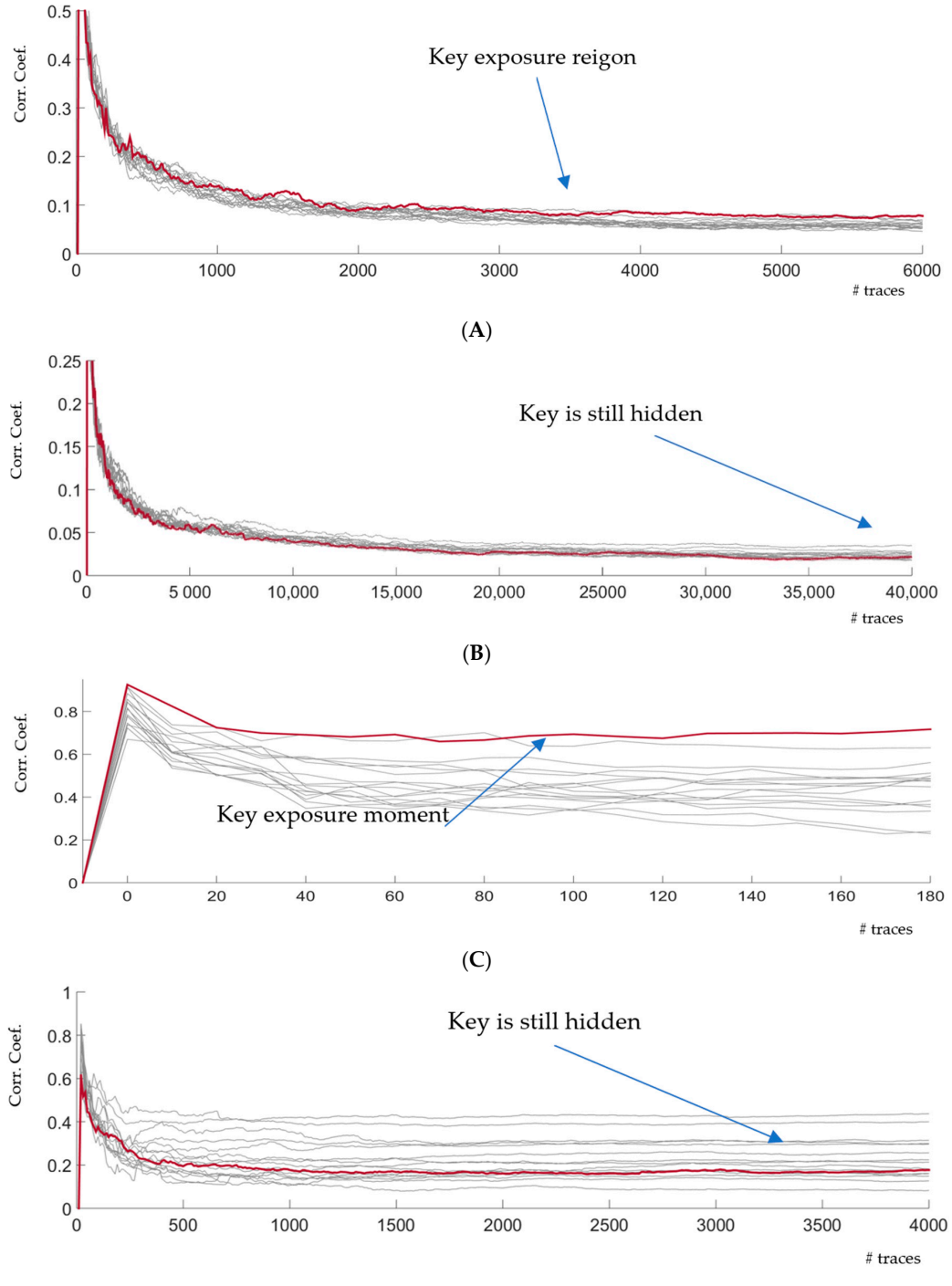

Figure 6. SCA attack to the unprotected and protected designs. (**A**) EM analysis of the unprotected SIMD PRESENT using HackMuMCU; (**B**) EM analysis of the CONFISCA-protected SIMD using HackMuMCU; (**C**) EM analysis of the unprotected SIMD using HackMuMCU using averaging; (**D**) EM analysis of the CONFISCA-protected SIMD using HackMuMCU using averaging. (we continued to 4000 averaged traces to see the ultimate strength of the protection.)

A subtle difference that presents a protection is that the DPL and Encoding methods fetch constant-Hamming-weight values from the main memory, while CONISCA fetches them from SIMD memory inside the processor which has less leakage as shown in our case study.

5.3. T-Test

Welch's T-test is a statistical test which determines if two populations have different means. By extending this general method for SCA, if mean value of a set of power/EM acquisitions with constant-input encryptions differs from that of random encryptions, then we can infer that there is a data-dependent leakage. NIST provides a guideline [21] to have a constant -input and a random-input sets, each containing at least 5000 power/EM acquisitions. It recommends a threshold of 4.5 to pass the test. In other words, if the mean value difference is less than 4.5, the device is considered secure for a confidence of >99.999%.

Figure 7 illustrates the T-test results for our implementations using 30,000 traces from constant-input and random-input traces computed by LASCAR, Ledger's Advanced SCA tool [22]. To have a constant-input traces, we separated EM traces whose first byte of their plaintext is zero. Accordingly, the rest of (non-zero values) were considered random-input traces. The LASCAR t-test engine gives an output curve that represents how distinguishable the constant and random traces are. As was mentioned, the max score is recommended to be less than 4.5 to have enough confidence on SCA protection.

Non-SIMD PRESENT, as the unprotected implementation, has the max T-test between 7 and 8 (Figure 7A) while DPL-based PRESENT has a higher T-test value (between 12 and 13), which indicates even more data-dependant leakage. This can be linked with the fact that non-SIMD performs 4-bit operations on data, and leakage from each bit is weakened by the EM of three other simultaneous bits acting as semi-random noise. On the contrary, DPL-based PRESENT processes bits in serial and hence each bit experiences less noise from other bits of the data, since they are processed at different times. In this sense, DPL and non-SIMD both fail on T-tests. On the contrary, CONFISCA presents a T-test value around 4.5, which is the NIST's threshold to pass the test.

5.4. SNR

SNR shows the level of leakage in comparison to the device background noise or the noise generated by a countermeasure. *SNR* is determined by calculation of:

$$SNR = \frac{VAR(Y|P)}{E(VAR(Y|P))} \tag{7}$$

where Y is the power/EM acqisitions of the device while processing plain-text P. Y is usually assumed as an addition between normally distributed noise N and leakage a leakage L:

$$Y = L + N \tag{8}$$

L is proportional to a leakage model l by a constant scale of e:

$$L = e.l \tag{9}$$

An example for l is Hamming weight or Hamming distance of a function of Plain-text and the secret key. For instance, $l(P,k) = HW(P \text{ x or } k)$.

To calculate *SNR*, we used 30K EM acquisitions and clustered them into two classses based on the first bit of their input plaintext. The aim was to determine level of signal produced by processing only 1-bit of plaintext to the noise. We chose this since for DPL, all bits are processed separetly, and their leakage is seperated in time. Therefore, an attacker will try to guess each bit separately.

Figure 8 illustrates the results for DPL-based and CONFISCA-protected PRESENTs. DPL-based has about two times higher SNR (0.012 in Figure 8A) than CONFISCA with SNR of 0.005 (Figure 8B). Execution of two PRESENT sub-functions, namely AddRoundKey and Sub-Bytes, are distinguishable in Figure 8B as two waves around the 15,000th and 27,000th EM samples.

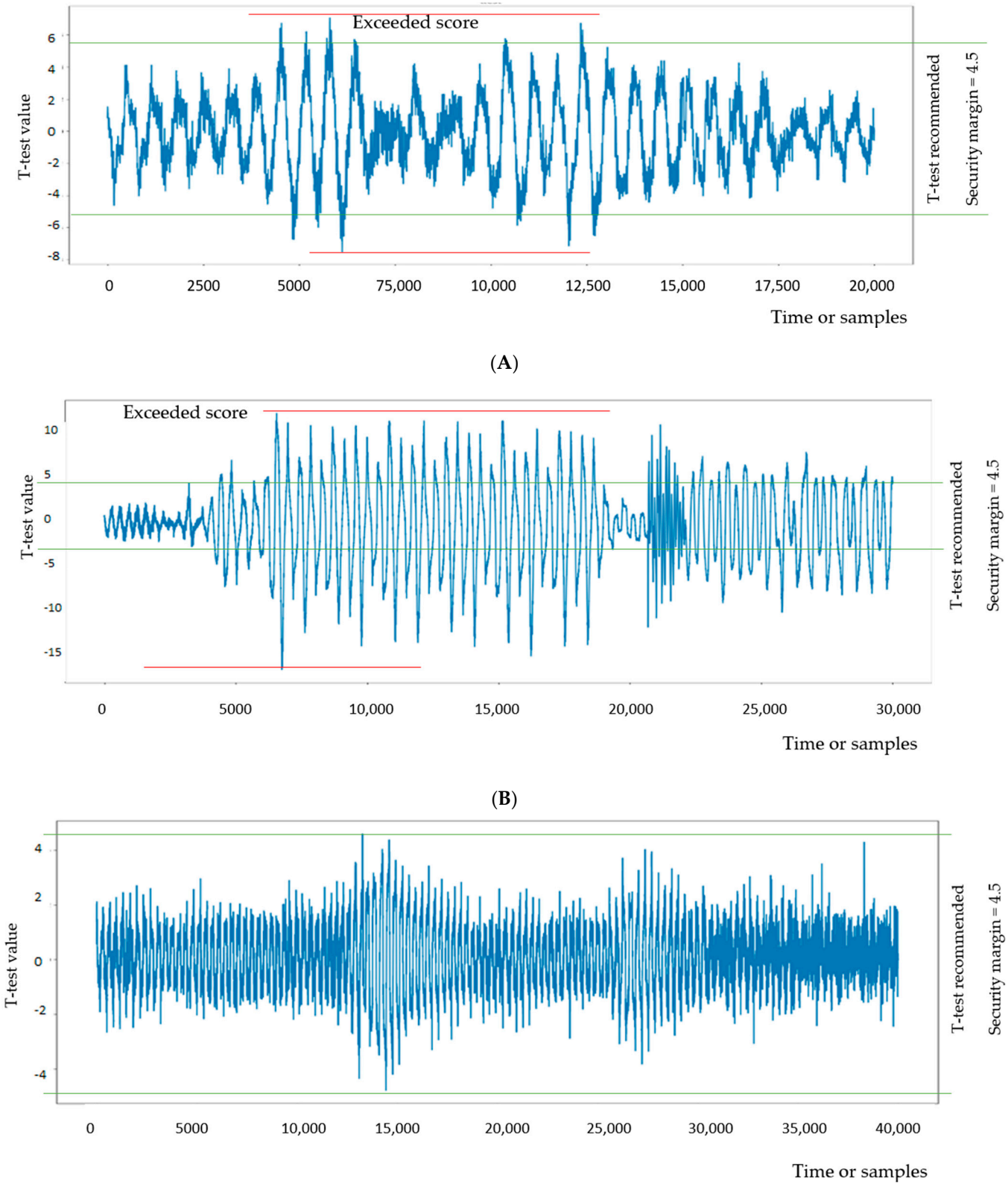

Figure 7. T-test on three different implementations of PRESENT cipher. (**A**) T-test on 30 K EM acquisitions of non-SIMD PRESENT; (**B**) T-test on 30 K EM acquisitions of DPL-based PRESENT; (**C**) T-test on 30 K EM acquisitions of CONFISCA-protected PRESENT.

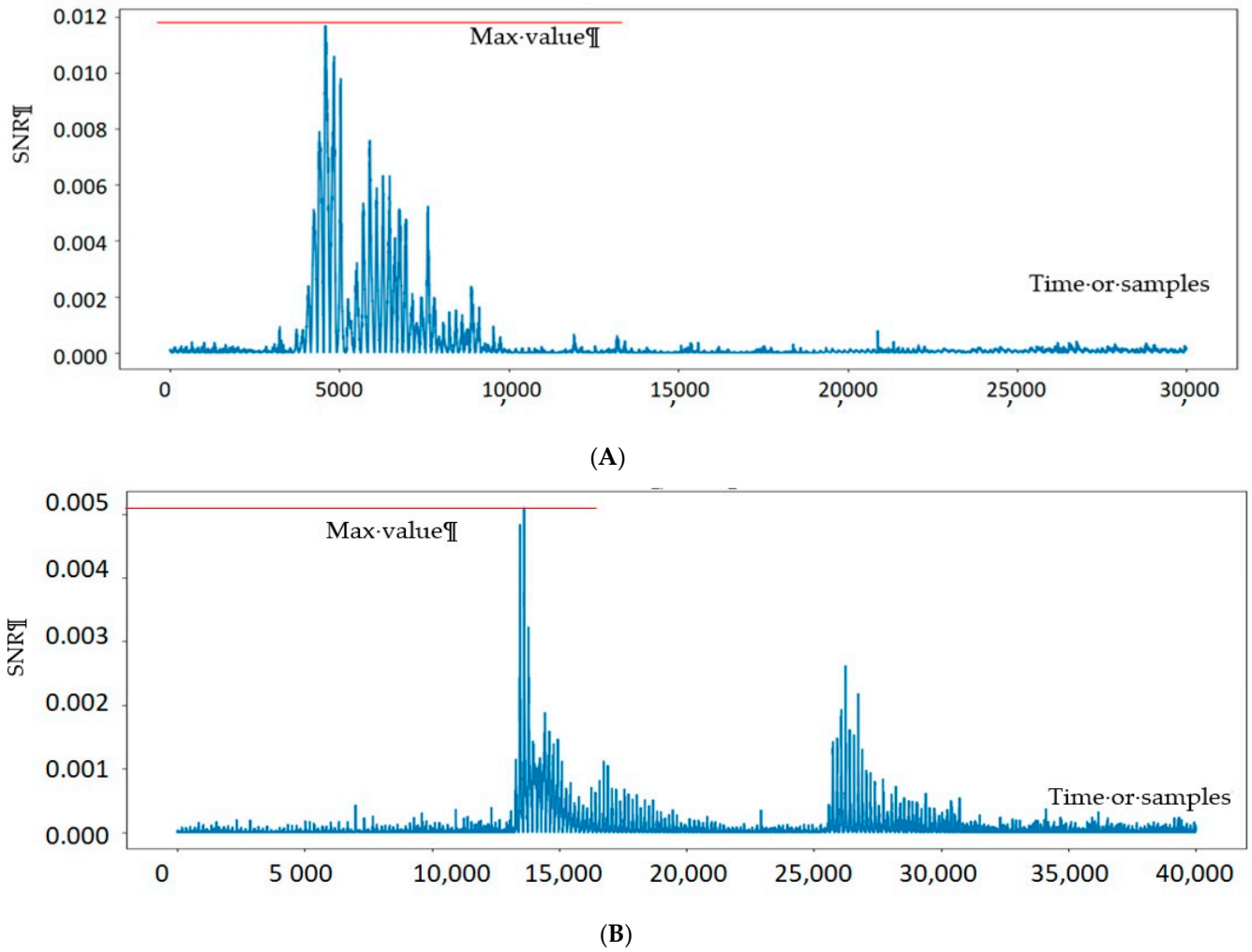

Figure 8. SNR test on two different countermeasures of PRESENT cipher. (**A**) SNR test of one bit on 30 K EM acquisitions of DPL-based PRESENT; (**B**) SNR test of one bit on 30 K EM acquisitions of CONFISCA PRESENT.

Although SNR shows a leakage residue, usual power models (Hamming weight and distance) in CPA cannot take advantage of this leakage to find the key, as was shown in the CPA sub-section.

5.5. NICV

Normalized Inter Class Variance or *NICV* [23] is a metric to measure leakage without knowing anything (including the secret key) except input or output of the device. It reveals to what extent the data-dependent variation on power or EM is distinguishable over the device's noise. It has a definition as:

$$NICV = \frac{VAR(E(Y|X))}{VAR(Y)} \tag{10}$$

in which X is the plaintext or ciphertext, and Y is the power or EM acquisition. This value is the maximum of all possible correlations computable by having Y and X. This equation is the ratio of power or EM variance that is dependent on the input X (the numerator of the fraction) over the total power consumption (or EM) variance of the hardware. To calculate this fraction, variance of the whole power or EM acquisitions gives $VAR(Y)$, while $VAR(E(Y|X))$ is calculated by separating acquisitions based on their inputs and extracting the variance of each separately.

We used four bits of plaintext to separate and have 16 groups of EM traces. The results in Figure 9 show how 16 groups of traces are different in their variance, which in turn shows how much leakage exists in each countermeasure. *NICV* presents similar ranking in comparison to *SNR*, even by examining 4 bits, while DPL has an *NICV* value between 0.03 and 0.35, and CONFISCA presents a better value between 0.14 and 0.16. Again, we should note that conformance-based approaches like *SNR* and *NICV* show the level of dependency between acquisitions and input/output, but this dependency does not necessarily mean that practical attacks can be performed to find the key.

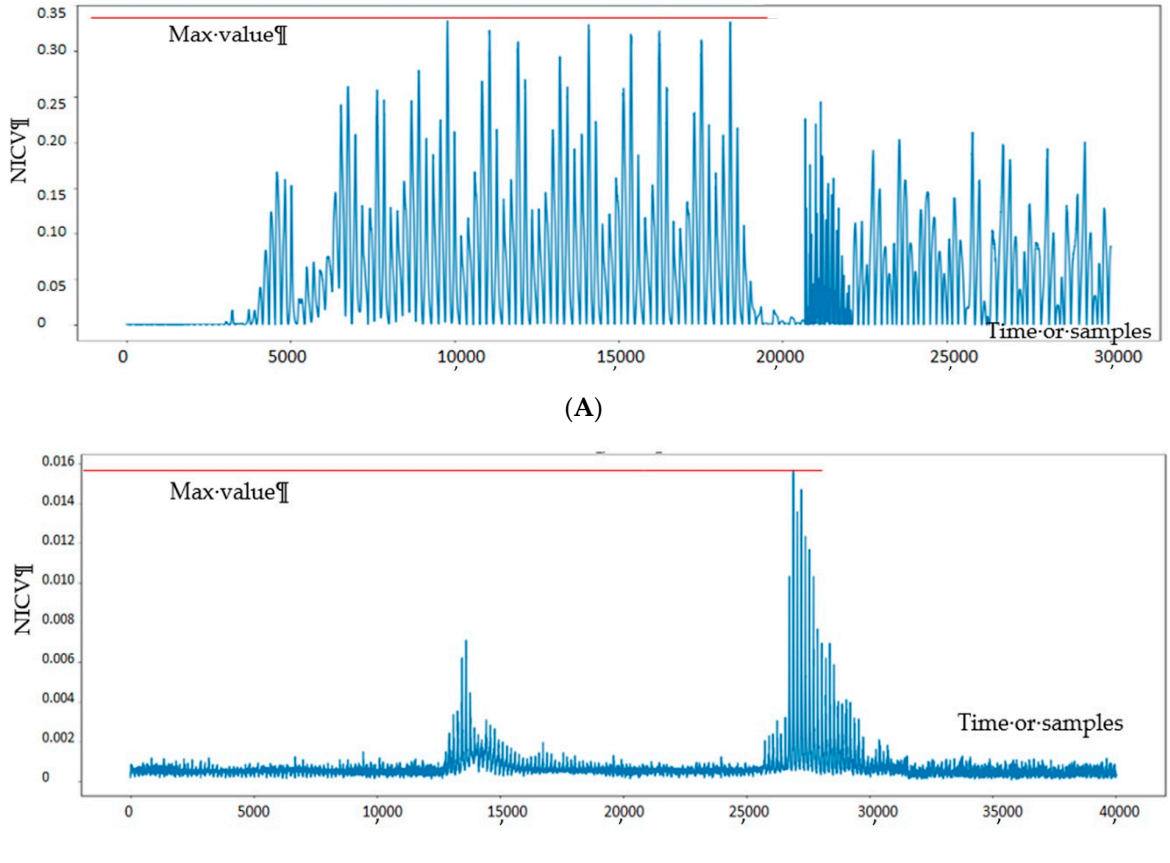

Figure 9. NICV test on two different countermeasures of PRESENT cipher. (**A**) NICV test of four bits on 30 K EM acquisitions of DPL-protected PRESENT; (**B**) NICV test of four bits on 30 K EM acquisitions of CONFISCA-protected PRESENT.

After conducting CPA, T-test, SNR and NICV, we can conclude that CONFISCA presents higher security than DPL since, for CPA, the CONFISCA key is not revealed after 2M averaging EM traces, while the DPL key is revealed after 60 K EM traces. CONFISCA passes a T-test (while DPL fails), and it has lower leakage on SNR and NICV reports in comparison to DPL countermeasure.

5.6. Higher-Order SCA

Since CONFISCA is a hiding countermeasure, higher-order SCA does not pose a threat unlike for masking countermeasures [24]. Higher-order SCA is used against masking countermeasures in which the secret information is split into $n + 1$ shares being processed independently and merged in the end of computation. The principle hinders SCA in a way that an attacker needs to guess at least $n + 1$ intermediate values of the computation to find all the secret shares and construct the secret key. This scheme is not useful for hiding countermeasures, like CONFISCA, because the secret is not split. In other words, guessing

one intermediate value is enough for attackers and guessing more intermediate values are equivalent and does not help attackers anymore.

5.7. FI Detection Analysis

Two main categories of FI are bit-flip on dataflow or storage which change the value of the data being transferred or stored, and faults on control flow, which alters the execution path of the software (and skips instructions). CONFISCA protects against both bit-flip FI and control flow faults (e.g., instruction skip faults). The duplicate of data defends against bit-flips since every modification on one copy is detectable on the other copy (except the exact same errors on both). For control-flow FI protection, one pair of the SIMD operations is devoted to computing a constant encryption (fixed plain-text). By comparing the corresponding output with the expected value, we can detect instruction-skip FI. Figure 10 illustrates the constant computation for the protected mode.

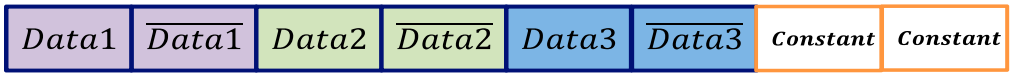

Figure 10. A pair of constant computation to defend instruction-skip FI (still three pairs of SCA-protected run in parallel).

Control flow faults are not detectable by the other mentioned countermeasures.

We didn't realize practical FI but conducted a theoretical analysis which should further be consolidated with experiments. To this aim, make the assumption that faults are injected in the data-path and will end up registered in the state register. Moreover, we assume that any fault combination for each 32-bit word has the same probability. Under the assumptions above, the only way to inject undetected multiple faults is to inject the exact same bit-flip faults in the two duplicates of the computation (e.g., dual faults on the 3rd bits of Data2 and it complement). The total amount of possible faults is $2^{64} - 1$. On the other hand, the undetectable fault scenarios are $\sum_{i=1}^{4} \binom{4}{i} = 2^4 - 1$ faults for each four computations Data1, Data2, Data3 and Data4. Therefore, the condition to inject an undetected fault is when four, three, two, or one pair of the computations have at least one (pairs of) fault. Then, the probability for the fault to be undetected is:

$$\frac{(2^8-1)^4 + \binom{4}{1}(2^4-1)^3 + \binom{4}{2}(2^4-1)^2 + \binom{4}{3}(2^4-1)}{2^{64}} = \frac{2^{16}-1}{2^{64}} \approx 2^{-48} \approx 1.5259 \times 10^{-5} = 0,00001525 \quad (11)$$

5.8. Overheads

Both the high-performance and the high-protection modes use the same code instance and only their data configurations are different (Figure 3). Therefore, there is no code size overhead over the unprotected mode. Comparing SIMD and non-SIMD implementations, applying the countermeasure needs twice the memory to compute the complementary outputs (Figure 2).

As concerns the performance, Table 3 shows the source codes and performances for both SIMD and non-SIMD PRESENT implementation. In the SIMD part, lines 1 and 4 are packing and unpacking the inputs into the NEON vectors. Lines 3 and 4 compute eight parallel AddRoundKey and Sbox sub-functions. The whole process takes 406 cycles. For the non-SIMD code, an 8-iteration loop performs the serial computations and the whole process takes 608 cycles to produce the same amount of data. Therefore, the unprotected SMID is about 33% *faster* than the non-SIMD implementation. As for the protected mode, half of the data used for protection, the protected mode is about 33% *slower* than the non-SIMD implementation. Obviously, the unprotected mode is two times faster than the protected.

Table 3. Performance of the SIMD and non-SIMD PRESENT.

	Duration	Code
SIMD	32	Vectors = vld1_u8(input);
	171	Vectors = vld4_u8(ARK_Table,Vectors);
	171	Vectors = vld4_u8(SBOX_Table,Vectors);
	32	vst1_u8(vectors,N_output);

Non-SIMD	608	for(int i = 0; i < 8; i++){
		buffer[i] = key[i]^input[i];
		output[i] = SBOXTable[buffer[i]];
		}

	total	

Table 4 provides a comparison between the overheads of *DPL*, *encoding*, and CONFISCA approaches. CONFICA is significantly faster than other concurrent methods and consumes less memory. There is an explanation on the overheads of the related work in [18]. Comparing the resistance against CPA, the DPL implementation in [10] needs 34 times more execution traces (broken after 4800 traces) and for encoding in [17] about 100 times more traces (broken around 10 K) in comparison to the unprotected versions, while CONFISCA was not broken after 8M averaging traces.

Table 4. Comparison of the overheads.

Method Ref.	Cipher	Overheads		
		Performance	Memory	Code
DPL [8]	PRESENT	≈900%	200%	
DPL [9]	PRESENT	≈2400%	20%	188%
Encoding [10]	Prince	767%	1966%	235%
CONFISCA	PRESENT	33%	100% -	

Finally, DPL and encoding cannot detect instruction skips. As an example of a potential threat, an instruction skip in DPL can skip the important "pre-charge" phase and expose the DPL value. CONFISCA can detect this fault in any stage of computation.

CONFISCA is a look-up-based implementation. Therefore, large input functions necessitate large tables. In this case, we break the function into smaller sub-functions and then apply CONFISCA or use other protection methods. While CONFISCA's approach is generic, simple, and effective based on the given results, it has a drawback that cannot be used on all processors (without SIMD).

6. Conclusions

CONFISCA is a concurrent software countermeasure against SCA and FI. Comparing with DPL and encoding countermeasures, it has considerably lower performance and memory overhead. Through a CPA experiment, it was not broken even after 8 million averaging traces, while the compared DPL could not resist after 200 K traces. We utilized T-test SNR, and NICV methods to evaluate leakage produced by each countermeasure. CONFISCA presents higher security than previous concurrent methods since CPA cannot find its hidden key, it passes the T-test, and has lower leakage on SNR and NICV reports.

Despite DPL and encoding, it also resists against instruction skip FI. Finally, enabling and disabling CONFISCA is feasible by flipping a bit on the input data without code change, giving CONFISCA the capability to trade performance and security on-the-fly.

The proposed method is generic and could be applied to a vast variety of cipher structures.

Author Contributions: Conceptualization, E.A., C.B., D.H. and A.P.; methodology, E.A.; C.B.; D.H. and A.P.; software, E.A.; validation, E.A. and A.P.; formal analysis, E.A. and D.H.; investigation, E.A. and D.H.; resources, D.H.; data curation, E.A.; writing—original draft preparation, E.A.; writing—review and editing, D.H.; visualization, E.A.; supervision, D.H. and M.F.; project administration, D.H. and M.F.; funding acquisition, D.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work is partially carried out under the SERENE-IoT project, a project labeled within the framework of PENTA, the EUREKA cluster for Application and Technology Research in Europe on NanoElectronics

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All data are contained within the article.

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

References

1. Peeters, E. *Advanced DPA Theory and Practice: Towards the Security Limits of Secure Embedded Circuits*; Springer: Berlin/Heidelberg, Germany, 2013.
2. Joye, M.; Tunstall, M. (Eds.) *Fault Analysis in Cryptography*; Springer: Berlin/Heidelberg, Germany, 2012; Volume 147.
3. Pahlevanzadeh, H.; Dofe, J.; Yu, Q. Assessing CPA resistance of AES with different fault tolerance mechanisms. In Proceedings of the 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC), Holiday Inn Macao Cotai Central, Macao, China, 25–28 January 2016; pp. 661–666.
4. Papadimitriou, A.; Nomikos, K.; Psarakis, M.; Aerabi, E.; Hély, D. You can detect but you cannot hide: Fault Assisted Side Channel Analysis on Protected Software-based Block Ciphers. In Proceedings of the 2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Frascati, Italy, 19–21 October 2020; pp. 1–6.
5. Breier, J.; Jap, D.; Bhasin, S. The other side of the coin: Analyzing software encoding schemes against fault injection attacks. In Proceedings of the 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, USA, 3–5 May 2016; pp. 209–216.
6. Aerabi, E.; Papadimitriou, A.; Hély, D. On a side channel and fault attack concurrent countermeasure methodology for MCU-based byte-sliced cipher implementations. In Proceedings of the 2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS), Rhodes Island, Greece, 1–3 June 2019; pp. 103–108.
7. Danger, J.L.; Guilley, S.; Bhasin, S.; Nassar, M. Overview of dual rail with precharge logic styles to thwart implementation-level attacks on hardware cryptoprocessors. In Proceedings of the 2009 3rd International Conference on Signals, Circuits and Systems (SCS), Medenine, Tunisia, 6–9 November 2009; pp. 1–8.
8. Hoogvorst, P.; Duc, G.; Danger, J.L. Software Implementation of Dualrail Representation. *COSADE* **2011**, *51*, 24–25.
9. Rauzy, P.; Guilley, S.; Najm, Z. Formally proved security of assembly code against power analysis. *J. Cryptogr. Eng.* **2016**, *6*, 201–216. [CrossRef]
10. Chen, C.; Eisenbarth, T.; Shahverdi, A.; Ye, X. Balanced encoding to mitigate power analysis: A case study. In *International Conference on Smart Card Research and Advanced Applications*; Springer: Cham, Switzerland, 2014; pp. 49–63.
11. De Meyer, L.; Arribas, V.; Nikova, S.; Nikov, V.; Rijmen, V. M&M: Masks and Macs against physical attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2019**, 25–50. [CrossRef]
12. Chaves, R.; Chmielewski, Ł.; Regazzoni, F.; Batina, L. SCA-Resistance for AES: How Cheap Can We Go? In Proceedings of the International Conference on Cryptology in Africa, Marrakesh, Morocco, 7–9 May 2018; Springer: Cham, Switzerland, 2018; pp. 107–123.
13. Resende, J.C.; Maçãs, R.J.; Chaves, R. Mask Scrambling Against SCA on Reconfigurable TBOX-Based AES. In Proceedings of the 2020 30th International Conference on Field-Programmable Logic and Applications (FPL), Virtual, 31 August–4 September 2020; pp. 243–248.
14. Chaves, R.; Kuzmanov, G.; Vassiliadis, S.; Sousa, L. Reconfigurable memory-based AES co-processor. In Proceedings of the 20th IEEE International Parallel & Distributed Processing Symposium, Rhodes Island, Greece, 25–29 April 2006; p. 8.
15. Bogdanov, A.; Knudsen, L.R.; Leander, G.; Paar, C.; Poschmann, A.; Robshaw, M.J.; Seurin, Y.; Vikkelsoe, C. PRESENT: An ultra-lightweight block cipher. In Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems, Vienna, Austria, 10–13 September 2007; Springer: Berlin/Heidelberg, Germany, 2007; pp. 450–466.
16. Franchetti, F.; Püschel, M. Generating SIMD vectorized permutations. In Proceedings of the International Conference on Compiler Construction, Budapest, Hungary, 29 March–6 April 2008; Springer: Berlin/Heidelberg, Germany, 2008; pp. 116–131.
17. Mangard, S.; Oswald, E.; Popp, T. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*; Springer: Berlin/Heidelberg, Germany, 2008; Volume 31.

18. Roy, D.B.; Bhasin, S.; Guilley, S.; Heuser, A.; Patranabis, S.; Mukhopadhyay, D. Leak Me If You Can: Does TVLA Reveal Success Rate. Available online: https://eprint.iacr.org/2016/1152 (accessed on 28 April 2021).
19. Heuser, A.; Picek, S.; Guilley, S.; Mentens, N. Side-channel analysis of lightweight ciphers: Does lightweight equal easy? In Proceedings of the International Workshop on Radio Frequency Identification: Security and Privacy Issues, Hong Kong, China, 30 November–2 December 2016; Springer: Cham, Switzerland, 2016; pp. 91–104.
20. Kazemi, Z.; Papadimitriou, A.; Souvatzoglou, I.; Aerabi, E.; Ahmed, M.M.; Hély, D.; Beroulle, V. On a Low-Cost Fault Injection Framework for Security Assessment of Cyber-Physical Systems: Clock Glitch Attacks. In Proceedings of the 2019 IEEE 4th International Verification and Security Workshop (IVSW), Rhodes Island, Greece, 1–3 July 2019; pp. 7–12. [CrossRef]
21. Gilbert Goodwill, B.J.; Jaffe, J.; Rohatgi, P. A testing methodology for side-channel resistance validation. In Proceedings of the NIST Non-Invasive Attack Testing Workshop, Nara, Japan, 25–27 September 2011; Volume 7, pp. 115–136.
22. LASCAR. Ledger's Advanced Side Channel Analysis Repository. Available online: https://github.com/Ledger-Donjon/lascar (accessed on 20 March 2021).
23. Bhasin, S.; Danger, J.L.; Guilley, S.; Najm, Z. Side-channel leakage and trace compression using normalized inter-class variance. In Proceedings of the Third Workshop on Hardware and Architectural Support for Security and Privacy, Minneapolis, MN, USA, 15 June 2014; pp. 1–9.
24. Wanderley, E.; Vaslin, R.; Crenne, J.; Cotret, P.; Gogniat, G.; Diguet, J.P.; Danger, J.L.; Maurine, P.; Fischer, V.; Badrignans, B.; et al. Security fpga analysis. In *Security Trends for FPGAS*; Springer: Dordrecht, The Netherland, 2011; pp. 7–46.