CONFISCA : an SIMD-based CONcurrent FI and SCA countermeasure with switchable performance and security modes

Ehsan Aerabi[*], Cyril Bresch[3], David Hély[3], Athanasios Papadimitriou[3], Mahdi Fazeli[*]
ehsan.aerabi@lcis.grenoble-inp.fr
[3]Univ. Grenoble Alpes, Grenoble INP
LCIS
Valence, France.
[*] Iran University of Science and Technology
Department of Computer Science
Tehran, Iran.

***Abstract*. CONFISCA is the first SIMD-based cipher implementation methodology which can concurrently resist against Side Channel Attack (SCA) and Fault Injection (FI). Its promising strength is presented in a PRESENT cipher case study. It has a considerably less performance and memory overhead in comparison to the previous concurrent countermeasures. By having one instance of the cipher, CONFISCA can on-the-fly switch between its two modes of operation: The High-Performance and High-Security. This gives us the flexibility to trade performance/power with security, based on the actual needs.**

Keywords: Hardware security; Side channel attacks; Fault Injection; Countermeasure, SIMD, NEON;

I. INTRODUCTION

Side-Channel Attack (SCA) and Fault Injection (FI) are two separate categories of attacks which are aimed to reveal secret information being stored or computed in sensitive digital devices (*e.g:* secret keys from cryptographic devices). SCA exploits the device's side channel leakage (*e.g.* power consumption) along with statistical approaches such as Differential or Correlation Power Analysis (DPA & CPA) to find the secret key [1]. FI can expose the secret information by injecting faults during the computation and observing the erroneous outputs [2].

Countermeasures against SCA generally try to decorrelate the device's side channel from the value of the sensitive data, either by masking, hiding or misaligning the computation in time [1]. Countermeasures against FI try to detect any faults in the computation and mostly apply computation duplication, data redundancy or coding [2].

Applying separate countermeasures against SCA and FI imposes large overheads and complexity which make it unusable for embedded constraint applications. More important, FI countermeasures have adverse effect on SCA protection because they increase the side-channel leakage due to their data/computation duplication [3, 4]. Therefore, we found few published *concurrent* (or combined) countermeasures that can resist both SCA and FI.

In this paper, we propose CONFISCA, the first secure software implementation methodology against SCA and FI based on Single Instruction Multiple Data (SIMD) parallel computation in modern processors. The hiding effect is consistent and effective throughout the cipher and for all its intermediate values. Unlike the masking countermeasures, higher-order SCA does not concern this hiding countermeasure [5]. We have evaluated its considerable strength against Electro-Magnetic (EM) CPA using two measurement setups.

The CONFISCA approach has two on-the-fly switchable modes of operation by only one instance of the cipher: The *High-Performance* and *High-Security*. The device can switch the modes based on the performance (or energy) and security requirements with no modification. The proposed method is generic and could be applied to a vast variety of cipher structures. It has significantly less performance overhead, in comparison to the previous work.

II. RELATED WORK

Recently, we published a paper on a concurrent countermeasure for processors without SIMD feature [6]. That work utilized wide data-words in modern processors to perform several byte-wise operations in parallel. We provided a case study on the AES SubByte calculations on a 32-bit processor. The four byte-wise parallel computations encipher the data with the *true* key and a *fake* key. The fake key hides the true key due to its covering effect. The results showed satisfactory protection. Our previous work cannot secure the whole cipher building blocks because it does not apply to all types of operations (e.g. look-up, addition, multiplication). In contrast, the current work has wider applicability since theoretically all one-to-one functions can be implemented using the CONFISCA approach, as we will show in our case study. Finally, aside from the protection against bit-flip FI, it also protects against the faults on the control flow (e.g: instruction skip faults), which was missing in the previous work.

Two main other threads of the concurrent software countermeasures against SCA and FI are *DPL-based* and *Encoding-based* methods. The both categories severely suffer from code size explosion and performance degradation. An explanation and comparison on their overheads have been provided in [6]. The *DPL-based* group employs the 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 power consumption. The software implementation was first proposed in [8] and followed by [9]. Since the DPL-based approached can compute only 1 bit per iteration, their proposed approach is extremely expensive both on performance and code size. The *Encoding-based* concurrent countermeasures use specific encoding with constant Hamming weight to theoretically eliminate the secret leakage. The reference [10] proposes an Encoding-based concurrent countermeasure based on the work in [11]. Encoding the inputs, then performing the computation using exponentially larger look-up tables and finally decoding is a long process; Hence the encoding approaches (like DPL-based) suffer from huge performance and memory overheads.

Finally, some masking concurrent works were published recently [12,13] which combine a masking approach with an error detecting or data duplication methods to protect SCA and FI on hardware. These methods are vulnerable to higher-order SCA as all of the masking methods. Most importantly, these masking countermeasures necessitates to compute cipher constant values (*e.g: S-Box table*) on each computation (instead of looking them up in a table). Therefore, they have intrinsically large performance overhead on software. As their results are on hardware (FPGA), we do not compare them against CONFISCA.

III. CONFISCA: PROPOSED COUNTERMEASURE

SIMD is aimed to boost the performance by parallel computation. There are plenty of SIMD instructions available in modern CPUs [14]. Among them, we chose parallel memory lookup functionality 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 have protection against FI and SCA as we describe below:

Assume that we have an SIMD two parallel look-ups. It accepts a vector like (X_1 ,X_2) and returns (T[X_1], T[X_2]), in which, T[X] represents the corresponding value in the table T by index of X. We denote the operation like:

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

Let think the cipher Ω 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 Ω as:

$\Omega(P) = F_k(F_{k-1}(\dots F_1(P)..)) = C$ (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 $2\times2^n = 2^{n+1}$ entries in the table. The look-up table has two (Fig 1). 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 $\bar{X}$ as a vector, in a way that the output value T[$\bar{X}$] has the logical complement value of T[X]. In brief:

$$F_i^{SIMD}(X, \bar{X}) = (T[X], \overline{T[X]}) = (F_i(X), \overline{F_i(X)}) \quad (3)$$

Obviously the output vector has a constant Hamming weight, which theoretically is SCA resistant. Holding the property (5) throughout the cipher, the output of F_i which is $(F_i(X), \overline{F_i(X)})$ could be used *directly* as the input vector for the next function F_{i+1}, without any modification and subsequently, we prevent any leakage for the intermediate values. The problem is how to arrange the values to have the property (5) for all values of X and $\bar{X}$.

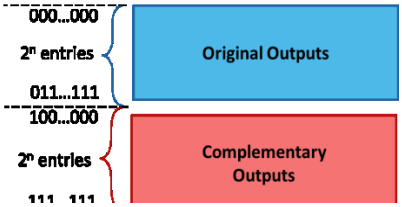

Fig 1 - Memory structure for the protected look-up

Let represent the n-bit input X of F_i by $x_n x_{n-1} \dots x_1$ then in address of $0|X$ or $0x_n x_{n-1} \dots x_1$ which is located in the first half part of the table, we store $F_i(X)$ along with a zero on its Most Significant Bit (MSB), like $0|F_i(X)$. Then in the second half part of the table in address $1|\bar{X}$ or $1\bar{x}_n\bar{x}_{n-1} \dots \bar{x}_1$, we store $1|\overline{F_i(X)}$. Finally, if we look-up $0|X$ we would have $0|F_i(X)$ and if we loo-up $1|\bar{X}$ we would have $1|\overline{F_i(X)}$:

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

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:

$$\Omega^{Secure}(P) = F_k^{SIMD}(F_{k-1}^{SIMD}(\dots F_1^{SIMD}1(0|P, 1|\bar{P})..)) = (0|C, 1|\bar{C}) \quad (5)$$

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

FI detection is achievable by comparing the original and complementary data as we will explain later in Sections V. CONFISCA's methodology is generic, simple and extremely effective based on the results in the next sections. However, large input functions necessitate large tables. In this case, either we break the function into smaller sub-functions then apply CONFISCA or use another protection methods.

IV. A PRESENT CASE STUDY

In this section, we demonstrate the SIMD protection idea in a case study on the PRESENT cipher [17]. 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 [15] and secure them using CONFISCA. We left it as a future work for 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 a 8-bit function, we need an SIMD table two times bigger than PRESENT.

A) PRESENT S-box Layer & 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 32 we have to store 12 and 19, respectively. 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.

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 $32\times5 = 160$ 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 by SCA.

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 provides us with *NEON*. Neon is an advanced SIMD architecture extension and presents several SIMD operations on vectors and matrices [14].

Table 1 - PRESENT S-box Layer protected memory content

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
01100	00101	00110	01011	01001	00000	01010	01101	00011	01110	01111	01000	00100	00111	00001	00010

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
11101	11110	11000	11011	10111	10000	10001	11100	10010	10101	11111	10110	10100	11001	11010	10011

Table 2 - PRESENT AddRoundKey protected memory content

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
01010	01011	01000	01001	01110	01111	01100	01101	00010	00011	00000	00001	00110	00111	00100	00101

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
11010	11011	11000	11001	11110	11111	11100	11101	10010	10011	10000	10001	10110	10111	10100	10101

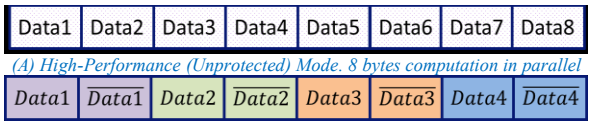

Fig - 2 – Data configuration in SIMD table look-up

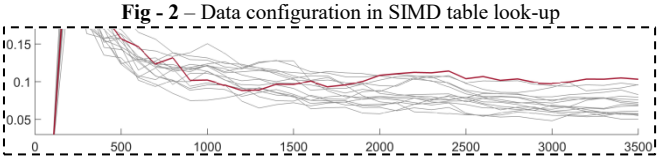

Fig 3 - SCA against the *conventional* (no SIMD) PRESENT cipher

The Unprotected version: NEON extension for table look-up includes 8 parallel look-up operations from a 32 bytes 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 8 parallel instances of the cipher algorithm. Fig 2-A illustrates the 8 look-up registers which contain 8 different data from 8 different instances of the cipher.

The Protected version: We can apply the CONFISCA method on 4 pairs of the NEON SIMD look-up. Fig 2-B shows the structure. Each pair is shown in 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 applied to them. Therefore in the *Protected* Mode we have 4 instances of the cipher.

V. Countermeasure Evaluation

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

SCA analysis: Due to the noise from the switching power supply of Xilinx Zybo Zynq-7000 board, our preliminary trial for power SCA needed extremely large number of power acquisitions. Therefore we decided to conduct electromagnetic SCA (EM-SCA) against the proposed method. The EM traces from the device acquired on 5 GS/Sec. In order to have higher level of accuracy we utilized two EM amplifiers: one from HackMyMCU board, and another from ChipWhisperer[16]. HackMyMCU is a SCA analysis tool developed in our laboratory, LCIS. It is designed for low-noise capture of electromagnetic or power traces, high-accuracy quantization of the captured signals and fast SCA analysis. We performed an Electromagnetic Attack against the implemented design using two amplifiers: AD8000 from Analog Devices installed on HackMyMCU and BGA2801 from NXP installed on ChipWhisperer. The attack tries to guess the correct 4-bit key on each look-up table operation of the PRESENT cipher.

First, we implemented a *conventional* (no SIMD) version of the unprotected cipher implemented using simple XOR (in AddRoundKey) and look-up table (in sBox). The PRESENT cipher is broken after 2000 traces, when the correct key stands out among the other key hypothesis as shown in Fig-3. This number forms a basis for our comparison later on these series of experiments. The attacks to SIMD implementation are shown from Fig 4-A to Fig4-H. Fig 4-A & B illustrate SCA on NEON implementation without the CONFISCA countermeasure, for HackMyMCU and ChipWhisperer, respectively. Moving from conventional implementation to NEON, we can observe that SCA needs around 3000 to 4000 traces to break the cipher.

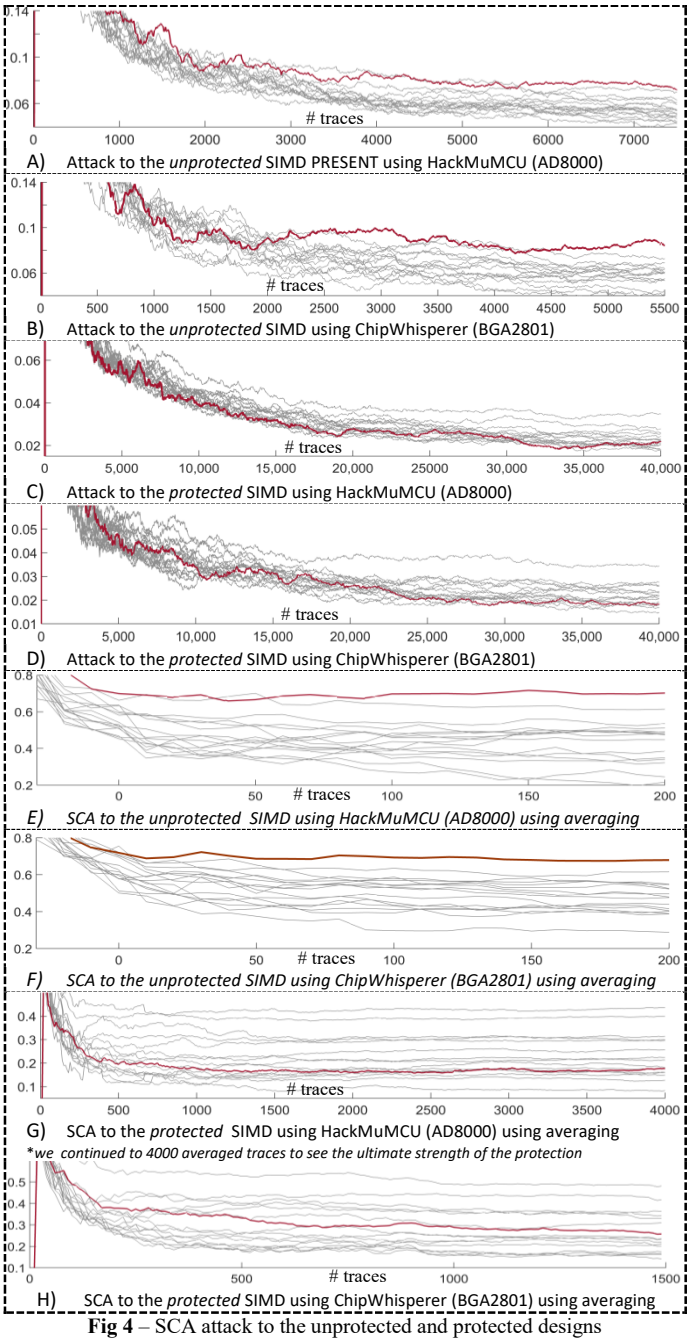

Fig 4 – SCA attack to the unprotected and protected designs

Fig 4-C & D show SCA against the countermeasure on both devices. After acquiring 40K traces we were not able to find the key. Therefore, we decided to mount stronger and faster attacks using averaging feature on our oscilloscope.

We set the oscilloscope to average the last 2048 traces in order to reduce the capturing noise. Fig 4-E & F shows the SCA against the unprotected NEON implementation. Obviously the correlation coefficients are higher (between 0.3 to 0.7) than normal acquisition (around 0.1).The key stands out after the first 10 or 20 acquisitions (2048 averaged traces each). HackMyMCU performs slightly better on these figures. Using the same averaging approach, Fig 4-G & H illustrate the SCA against the protected mode. It should be noted that in Fig 4-G, we let the SCA continue until 4K×2048≈ 8M traces on 5 GS/s to see the ultimate strength of CONFISCA. But the correct key is well hidden among the other keys and does not seem to stands out upon this order of magnitude of the number of traces.

FI Detection Analysis: To theoretically evaluate the fault detection capabilities, we 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 an undetected multiple faults is to inject the exact same bit-flip faults in the two duplicates of the computation. The total amount of possible faults is 2^{32}-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 pairs of the computations have at least one (pairs of) fault. Then the probability to not detect a fault is:

$$\frac{(2^4-1)^4 + \binom{4}{1}\times(2^4-1)^3 + \binom{4}{2}\times(2^4-1)^2 + \binom{4}{3}\times(2^4-1)}{2^{32}}$$

$$= \frac{2^{16}-1}{2^{32}} \approx 2^{-16} \approx 1.5259 \times 10^{-5} = 0{,}00001525 \qquad (7)$$

It is also possible to add capability to detect faults on the control flow (e.g: *instruction skip* faults). To this aim, one pair of the SIMD operations is devoted for computing a constant encryption (fixed plain-text). By comparing the corresponding output with the expected value we can detect instruction-skip FI. Fig 5 illustrates the constant computation for the protected mode.

Overheads: Comparing the protected and unprotected modes in Fig-2, they use the same code and only their data configurations are different. Therefore, there is no code size overhead over the unprotected mode. Comparing SIMD and conventional implementations, applying countermeasure needs twice memory to have the complementary outputs (Fig-1).

On the performance, Table 3 shows the source codes and performances for both NEON and conventional PRESENT. In the NEON 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 conventional 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 NEON is about %33 *faster* than conventional implementation. As for the unprotected mode, half of the data used for protection, the protected mode is about %33 *slower* than the conventional implementation. Obviously, the unprotected mode is two times faster than the protected mode.

Finally Table 4 provides a comparison between the overheads of *DPL*, *encoding* and CONFISCA approaches. There is an explanation on the overheads of the related work in [15]. Aside from the lower overheads of CONFISCA, about the strength of the related woks: DPL in [9] implementation was 34 times more resistant (broken after 4800 traces) and for encoding in [11] about 100 times more resistant (broken around 10K); While CONFISCA was not broken after 8M averaging traces.

VI. Conclusion

We presented CONFISCA, a concurrent countermeasure against SCA and FI based on SIMD look-up table. On a case study on ARM Cortex-A9 and its NEON SIMD feature, we secured PRESENT cipher. The SCA protection was evaluated by 8 million EM averaged traces while the secret key was still hidden. We discussed its strength against data-path FI and instruction skip. The CONFISCA overheads are relatively lower than the related works. The proposed method could be deployed for AES with the same respects but with two times bigger tables.

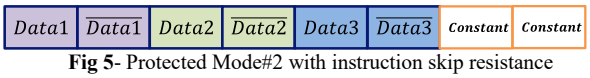

Fig 5- Protected Mode#2 with instruction skip resistance

Table 3 – Performance of the NEON and conventional PRESENT

Duration		Code	

32
171
171
32
	1
	2
	3
	4
		`Vectors = vld1_u8(input);`
		`Vectors = vld4_u8(ARK_Table,Vectors);`
		`Vectors = vld4_u8(SBOX_Table,Vectors);`
		`vst1_u8(vectors,N_output);`
			NEON

608
total
	1
	2
	3
	4
		`for(int i=0; i <8;i++){`
		`buffer[i]=key[i]^input[i];`
		`output[i] = SBOXTable[buffer[i]];`
		`}`
			CONVENTIONAL

Table-4 – Comparison of the overheads

Method Ref.	Cipher	Overheads		
		Performance	Memory	Code
DPL [8]	PRESENT	%800	%200	
DPL [9]	PRESENT	%200	%20	%188
Encoding [11]	Prince	%767	%1966	%235
CONFISCA	PRESENT	%33	%100	-

References

[1] Peeters, E., 2013. Advanced DPA theory and practice: towards the security limits of secure embedded circuits. Springer Science & Business Media.

[2] . Joye, M. and Tunstall, M. eds., 2012. Fault analysis in cryptography (Vol. 147). Heidelberg: Springer.

[3] Pahlevanzadeh, H., Dofe, J. and Yu, Q., 2016, January. Assessing CPA resistance of AES with different fault tolerance mechanisms. In Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific (pp. 661-666). IEEE.

[4] Luo, P., Fei, Y., Zhang, L. and Ding, A.A., 2014, December. Side-channel power analysis of different protection schemes against fault attacks on AES. In ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on (pp. 1-6). IEEE.

[5] Wanderley, E., Vaslin, R., Crenne, J., Cotret, P., Gogniat, G., Diguet, J.P., Danger, J.L., Maurine, P., Fischer, V., Badrignans, B. and Barthe, L., 2011. Security fpga analysis. In Security Trends for FPGAS (pp. 7-46). Springer, Dordrecht.

[6] Aerabi, E., Papadimitriou, A. and Hely, D., 2019, July. On a Side Channel and Fault Attack Concurrent Countermeasure Methodology for MCU-based Byte-sliced Cipher Implementations. In 2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS) (pp. 103-108). IEEE.

[7] Danger, J.L., Guilley, S., Bhasin, S. and Nassar, M., 2009, November. Overview of dual rail with precharge logic styles to thwart implementation-level attacks on hardware cryptoprocessors. In *2009 3rd International Conference on Signals, Circuits and Systems (SCS)* (pp. 1-8). IEEE.

[8] Hoogvorst, P., Duc, G. and Danger, J.L., 2011. Software implementation of dualrail representation. COSADE, February, pp.24-25.

[9] Rauzy, P., Guilley, S. and Najm, Z., 2016. Formally proved security of assembly code against power analysis. Journal of Cryptographic Engineering, 6(3), pp.201-216.

[10] Breier, J., Jap, D. and Bhasin, S., 2016, May. The other side of the coin: Analyzing software encoding schemes against fault injection attacks. In Hardware Oriented Security and Trust (HOST), 2016 IEEE International Symposium on (pp. 209-216). IEEE.

[11] Chen, C., Eisenbarth, T., Shahverdi, A. and Ye, X., 2014, November. Balanced encoding to mitigate power analysis: a case study. In International Conference on Smart Card Research and Advanced Applications (pp. 49-63). Springer, Cham.

[12] Reparaz, O., De Meyer, L., Bilgin, B., Arribas, V., Nikova, S., Nikov, V. and Smart, N., 2018, August. CAPA: the spirit of beaver against physical attacks. In Annual International Cryptology Conference (pp. 121-151). Springer, Cham.

[13] De Meyer, L., Arribas, V., Nikova, S., Nikov, V. and Rijmen, V., 2019. M&M: Masks and Macs against physical attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems, pp.25-50.

[14] https://developer.arm.com/architectures/instruction-sets/simd-isas/neon/intrinsics

[15] Franchetti, F. and Püschel, M., 2008, March. Generating SIMD vectorized permutations. In International Conference on Compiler Construction (pp. 116-131). Springer, Berlin, Heidelberg.

[16] O'Flynn, C. and Chen, Z.D., 2014, April. Chipwhisperer: An open-source platform for hardware embedded security research. In International Workshop on Constructive Side-Channel Analysis and Secure Design (pp. 243-260). Springer, Cham.

[17] Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J., Seurin, Y. and Vikkelsoe, C., 2007, September. PRESENT: An ultra-lightweight block cipher. In International workshop on cryptographic hardware and embedded systems (pp. 450-466). Springer, Berlin, Heidelberg.