A Non Profiled and Profiled Side Channel Attack Countermeasure through Computation Interleaving

Isabella Piacentini, Alessandro Barenghi, Gerardo Pelosi
Dept. of Electronics, Information and Bioengineering - DEIB
Politecnico di Milano, 20133, Milano, Italy
{isabella.piacentini, alessandro.barenghi, gerardo.pelosi}@polimi.it

***Abstract*—Side channel attacks analyse devices to retrieve secret informations. These attacks can be performed using either a synthetic model or by profiling a specific instance of the targeted design. Our proposal is a novel countermeasure characterized by temporal interleaving of the computation. This approach improves upon existing methods by rendering the profiled models of a device non-portable and offering resistance against first-order non-profiled attacks. The assessment of the security of our proposed approach covers scenarios of profiled attacks with feature reduction techniques, both under a single-device and multi-device model, as well as first-order non-profiled attacks. Our design demonstrates improved results in terms of resource consumption and timing when compared to alternative solutions.**

***Index Terms*—Hardware Security, Countermeasures, Side Channel Attacks, Profiled Attacks**

I. Introduction

Side channel attacks (SCAs) are one of the principal threats to the security of digital devices. Their ability to extract secret keys from a computing device executing the implementation of a target cipher, regardless of the correctness of the implementation itself, and the absence of mathematical vulnerabilities, makes them a source of prime concern. The key working principle of an SCA is to model the expected, secret-data dependent behavior of the side channel of a device, and match it to the actual measurements taken from the device itself. Traditionally SCAs are split into two categories, depending on whether the model is synthetically obtained (a.k.a. *non-profiled* SCAs) or derived in a data driven fashion from a device identical to the one under attack, but fully controlled by the attacker (a.k.a. *profiled* SCAs). Profiled SCAs are often described as the most powerful ones, as they derive a perfectly fitting model for the behaviour of the device during the execution of a given cipher implementation, known as a *profile* or *template*. Indeed, an a posteriori, data-driven modeling method allows an attacker to capture also the effects on the side channel of many countermeasures, albeit at the cost of an increased number of measurements. One notable hindrance to profiled attacks is the so-called *portability* of the profiles from the device they are obtained from to the one being attacked due to, e.g., process variability and measurement setup differences. These facts may hinder or prevent altogether the execution of a profiled SCA. However, it has been experimentally shown in [1]–[5] that building a profile employing measurements coming from multiple device instances (Multi Device Model, MDM) and reducing the measurement setup differences, leads to successful profiled SCAs, regardless of the specific profiling technique (i.e., Bayesian templates or machine learning/neural networks based classifiers). While natural inter-device variability was proven to be manageable by an attacker, the authors of [6] report a countermeasure against profiled attacks named Scramble Suit, which introduces an approach to amplify the said inter-device variability to the point where the profile distortion prevents attacks in a systematic fashion. In [6], such amplification is obtained computing two identical instances of the cipher to be protected, one of which acts on a scrambling secret key derived blending together a device-dependent element while the other employs the actual user-supplied key. This approach superimposes a device-and-computation dependent noise onto the side channel signal coming from the computation with the user key. Scramble Suit does not provide protection against non profiled SCAs (the user supplied and the scrambling key can be recovered), and was not validated against MDM attacks.

In this work, we propose an SCA resistant design for block ciphers, that improves on Scramble Suit [6], as it combines the resistance against profiled SCAs with a significant amount of resistance against non profiled ones. The main idea is to exploit the temporal interleaving of computations of the same cipher with different scrambling keys, all-but-one of which depend on both the device and the actual user-supplied key. Such a computation interleaving also performs (pseudo) random precharging of the datapath and computation state registers, providing resistance to first-order non-profiled SCAs. Our approach can be augmented with spatial redundancy, should the designer be willing to raise the amount of distance among the profiles of different devices. We also explore this avenue, providing quantitative results. Comparing our approach with SCA countermeasures oriented at preventing non-profiled SCAs (which also hinder to some extent profiled SCAs), such as the combination of shuffling and (high-order) masking, we report that our approach is a more efficient alternative to first-order masking, obtained as a combination of random precharging and redundant computation shuffling. Finally, our design requires, as Scramble Suit [6] does, the presence of a side channel resistant element generating a scrambling key blending the user-supplied key value with a unique feature of the device in a non-extractable fashion, even via side channel. The authors of [6] noted that a strong PUF [7] was required for

this purpose, employing the user-supplied key as a challenge and using the response as the scrambling secret key. While there is a flourishing research on SCA resistant PUFs [7], we highlight that a recent line of research proposed a design of a component, which fulfills the requirement of generating a scrambling key [8] without involving the process variability of the device. Indeed, such a component can be used to generate fresh scrambling keys starting from a randomly chosen secret parameter securely stored on the device. The resistance against SCAs of [8] hinges on the fact that extracting information from side-channel data is equivalent to the computationally hard problem known as Learning With Physical Errors (LWPE), for which no efficient solution is known.

Contributions. We introduce a novel design aimed at countering *both profiled and non-profiled* SCAs with a single countermeasure, by exploiting temporal computational redundancy. We validate our security claims with an implementation onto an FPGA target, employing as a case study the AES-128/-256 cipher. We show that our design protected against profiled SCAs exhibits a 50.7% classification accuracy across the board in a two-class (single bit) classification with single- and multi-device-models. At the same time, our design increases the number of measurements to disclose security metric, against non-profiled SCAs, up to $333\times$ w.r.t. an unprotected design. We evaluate the performance of our design, reporting a resource (LUT+FF) overhead, w.r.t. an unprotected design, equal to 66%, comparing favourably with the overheads of common masking countermeasures, which are in the 219% – 274% range [9]–[11]. We note that our design has a $38\times$ to $104\times$ lower pressure on the Random Number Generator (RNG) throughput with respect to traditional masking schemes [9], [10], and halvens the requirements with respect to the state-of-the-art randomness reusing schemes [11]. Finally, our countermeasure can also be augmented with spatial redundancy, in turn providing a tunable increase of the security margin.

II. Background

Side Channel Attacks exploit the link between the data being processed by a device and one or more environmental parameters, such as the power consumption, Electro-Magnetic (EM) radiations or computation time. The device combines the cryptographic key with known inputs (or producing known outputs) when computing a publicly known implementation of the cryptographic primitive. The attacker models a portion of the side channel signal corresponding to the computation of a sensitive intermediate value, which depends only on a small portion of the secret key. In the following, we consider such a portion to be a single bit for the sake of clarity. Consider a cryptographic primitive with an l-bit key k and a set of inputs $\mathcal{P} = \{P_j,\, 0 \leq j \leq |\mathcal{P}|-1\}$ (e.g., plaintexts), $|\mathcal{P}| \gg 1$. A time series of $s > 0$ side channel measurements (samples) obtained from the device executing the whole cryptographic primitive is known as a *trace*. Traces are measured feeding the device an input P_j and an (unknown) value for a single bit of the key k_i, $1 \leq i \leq l$, We thus denote a trace as $\hat{T}_j^{(k_i)} = \{\hat{T}_j(t) \mid 1 \leq t \leq s\}$, with $1 \leq j \leq |\mathcal{P}|$ The sequence of measurements of the leakage from the intermediate value of choice, while computing on different plaintexts is identified by the time instant $t = t^*$, i.e., $\hat{T}^{(k_i)}(t^*) = \{\hat{T}_j(t^*),\, 1 \leq j \leq |\mathcal{P}|\}$. An SCA is usually split into two phases: modeling and exploitation. In the modeling phase the attacker builds as many models of the side channel leakage as the possible values of k_i (two in our case, i.e., $\hat{M}^{(k_i=0)}$, $\hat{M}^{(k_i=1)}$) during the computation of the chosen sensitive intermediate value. In the exploitation phase, the attacker determines which model fits best the behavior of the device as observed in a set of traces $\hat{T}^{(k_i)}$, revealing k_i.

Non-profiled Attack. In a non-profiled attack, the attacker builds the models $\hat{M}^{(k_i=a)}, a \in \{0, 1\}$ according to a synthetic computation. Considering the case of power consumption and EM emission side channel attacks, the most common model is the toggle-count of a portion of the logic circuit. This corresponds to the Hamming distance between the values held by the computing circuit before and after the sensitive intermediate value is computed. To this end, the attacker picks a sequence of random plaintexts $\mathcal{P}$, computes the intermediate computation values according to the postulated key value a, and obtains $\hat{M}^{(k_i=a)}$ as a set of expected power consumptions. In the exploitation phase, the attacker collects the set of traces $\hat{T}_j^{(k_i)}$, employing the sequence of plaintexts $\mathcal{P}$ used for the computation of $\hat{M}^{(k_i=a)}$. The attacker now considers sequence of samples of a given time instant $\hat{T}^{(k_i)}(t)$, $1 \leq t \leq s$, and the values obtained through the models $\hat{M}^{(k_i=a)}, a \in \{0, 1\}$ as samples from random variables, and tries to determine for which value of k_i $\hat{M}^{(k_i=a)}$ fits best $\hat{T}^{(k_i)}(t)$. To quantify the fitness, a statistical tool, also known as *distinguisher* is applied to the samples of the aforementioned random variables. Pearson's linear correlation coefficient and the Mutual Information (MI) are distinguishers applied by SCAs known as Correlation Power Attack (CPA) [12] and Mutual Information Analysis (MIA) [13], respectively. A large amount of research effort was devoted to understand which distinguisher allows to derive the secret key value with the least amount of traces.

Heuser et al. in [14] showed that, for 1st-order non-profiled attacks, Pearson's correlation coefficient is the information theoretic optimal distinguisher, if the consumption model is known on a proportional scale, while the noise follows a Gaussian zero-average statistical distribution. In [15] the authors derive relations binding together the Success Rate, SR [16] metric (equivalently, the Guessing Entropy, GE [16]) and the mutual information (MI) distinguisher. They also derive an upper bound to the MI starting from the Signal-to-Noise (SNR) ratio of the measurements, and link the SNR to the minimum number of traces (a.k.a. Measurements-To-Disclosure, MTD) required to extract the secret key, with a probability of success decided a priori. Since SR, GE, MI and the SNR can all be expressed as a function of MTD, we will employ the MTD with Pearson's correlation coefficient as figure of performance in withstanding non-profiled attacks. As reported in [17], while MTD (in unprofiled attacks) and the accuracy (in profiled ones) provide the same information as the GE when the number of traces grows asymptotically, they may

yield different results if the amount of information available to the attackers is relatively low. We will also measure the effectiveness of the attacks in terms of GE.

A popular method to test for side channel vulnerability of an implementation is the Test Vector Leakage Assessment (TVLA), first proposed in [18]. TVLA assesses if two trace populations have the same or a different time-wise mean employing Student's t-test. In the *non-specific* variant the two populations are obtained as i) a repeated encryption of the same plaintext under a fixed key and ii) the encryption of random plaintexts under the same key. In the *specific* variant the two populations are obtained classifying, according to an intermediate computation value, a single set of traces obtained encrypting random plaintexts under a fixed key. The specific t-test conveys less information than a CPA attack: indeed its results are not scale-invariant. Therefore, we do not report the results of TVLA procedures on our implementations.

The described attack is denoted as a 1st-order *non-profiled* attack. A d-th order attack, $d > 1$, is executed to overcome the countermeasures against non-profiled SCAs in the cryptographic implementation and consists in combining properly the measurements in each trace coming from d distinct time instants in order to define/compute properly the values of both $\hat{T}^{(k_i)}(t)$ and $\hat{M}^{(k_i=a)}$. An implementation protected with a d-th order masking countermeasure processes the sensitive values of the implementation at hand through splitting the original values in d randomized shares each. Thus, the higher the value of d, the more difficult is to envision a recombination function able to put together the measurements at different time instants and the higher is the computational overhead.

Profiled Attack. In profiled attacks, the attacker derives $M^{(k_i=a)}$ applying Bayesian or machine learning strategies. In the modeling phase, the attacker employs a copy of the target device where the secret key k can be changed at will. The attacker collects traces from the device for a large set of plaintexts $\mathcal{P}$ and all possible values of k_i, and obtains $M^{(k_i=a)}$ through either descriptive statistics methods, or trains a ML model with the traces. The *a posteriori* model $M^{(k_i=a)}$ of the device obtained is also commonly known as a *profile* or *template* of the device. In the *exploitation phase* the attacker collects traces from the target device where the value of k_i is unknown, and tries to determine the best fitting model for the collected traces belong, revealing the value of k_i.

The first profiled attack was first introduced in [19] with the name of *template attack* (TA). In a TA, the attacker employs a multivariate Gaussian distribution, which is fully described by its s-dimensional mean vector $\mu^{(k_i)}$,($s \gg 1$), and covariance matrix $\Sigma^{(k_i)}$, as the model $M^{(k_i=a)}$ of an entire trace. We thus have $T^{(k_i)} \sim \mathcal{N}(\mu^{(k_i)}, \Sigma^{(k_i)})$, with probability density function

$$\Pr(T^{(k_i)} = x) = \frac{\exp\big(-\frac{1}{2}(x-\mu^{(k_i)})(\Sigma^{(k_i)})^{-1}(x-\mu^{(k_i)})^{\mathrm{tr}}\big)}{\sqrt{(2\pi)^n \det(\Sigma^{(k_i)})}}.$$

TAs assume that the additive zero-average Gaussian measurement noise affects each component independently.

In the modeling phase, the attacker derives, for each value of k_i, a profile through a sample estimate of both $\hat{\mu}^{(k_i)}$ and $\hat{\Sigma}^{(k_i)}$, from sets of traces $T^{(k_i=a)}$ collected from the controlled device, setting $k_i = a$ and feeding it with uniformly randomly selected plaintexts. In the exploitation phase, the attacker determines the likelihood of a trace $\hat{T}$ of being an instance (i.e., sample) of one of the random vector variables representing the models $M^{(k_i=a)} = T^{(k_i)} = \{T(t) \mid 1 \le t \le s\}$, exploiting Bayes' theorem: $\Pr(k_i \mid \hat{T}) = \frac{\Pr(T^{(k_i)}=\hat{T}) \cdot \Pr(k_i)}{\sum_{h=1}^{l} \Pr(T^{(k_h)}=\hat{T}) \cdot \Pr(k_h)}$ where $\Pr(k_i)$ is the a priori probability associated to the specific key-bit value k_i that does not consider $\hat{T}$. Typically, all key values are equally likely, therefore $\Pr(k_i) = \frac{1}{2}$. The value k_i which maximizes the aforementioned a posteriori probability is selected as the value of the i-th bit of the secret key employed by the device under attack.

While TAs are information theoretically optimal, they have constraints on the measurement setup. The sets of traces in the modeling phase and the one(s) in the exploitation phase should be perfectly aligned, measured on the same vertical scale, and the clock jitter should be negligible. Furthermore, no DC drifts (e.g., due to thermal effects) should be present [1]. TAs are also influenced by the profile variability coming from process variation across different devices. Finally, TAs suffer from a superlinear increase in the amount of traces for the modeling phase when dealing with $d > 1$ masked implementations.

Profiled attacks executed applying machine/deep learning techniques were employed to overcoming the adverse effects on TAs provided by masked implementations of cryptographic primitives [20] as well as to well manage difficulties due to trace misalignments, clock jitter of the device [21] and even insertion of artificial independent noise [22], during the modeling or the exploitation phases. The authors of [2]–[4] consider the cases where a mismatch between the modeling and exploitation phase is caused by the inter-device process variability, and no clock jitter and misalignments are present. In this case, the mismatch between modeling and exploitation can be compensated building a leakage model that makes use of traces collected from multiple attacker-controlled devices (Multi-Device Model, MDM). The same conclusions is also confirmed by [5], where the authors recommend a MDM approach to overcome the issue of device portability of profiled attacks even when making use of deep learning strategies. The authors suggest to use three devices: two in the modeling phase, one in the exploitation phase.

Since the aim to amplify the intrinsic inter-device variability we will evaluate its effectiveness against TAs [19], while physically removing all the setup-dependent factors being a hindrance for them. To this end, our setup is untouched during all the measurements, is done on a single physical FPGA device, clock jitter and misalignment effects are carefully avoided, and we experimentally checked that no DC drift takes place. We also do not consider the combined action of our countermeasure and masking techniques, as it would prevent from singling out the effectiveness of our countermeasure alone. In our setting, we did not encounter numerical issues in

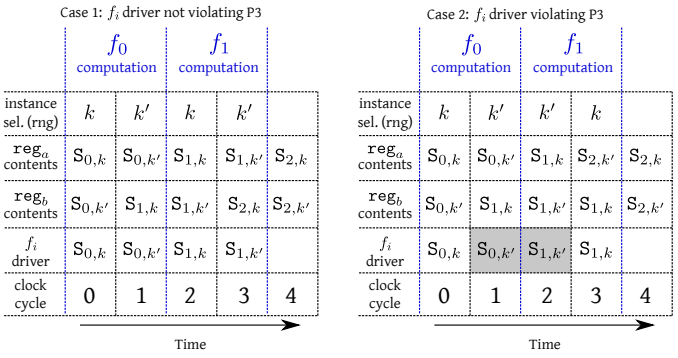

Fig. 1. Simplified timing diagram showing the contents of an iteratively computed block cipher interleaving two cipher executions, for the computation of f_0 and f_1 on both states. The grey highlight underlines the only transition requiring random precharging to remove leakage to non-profiled attacks

estimating the full covariance matrices of the profiles, in turn allowing the use TAs as our security gauge.

Feature Selection. A concrete way to improve the overall Signal-to-Noise Ratio (SNR) of the traces employed in both the modeling and exploitation phase is to perform a *feature selection pass*. Feature selection is performed either keeping only the samples of a trace in time instants where the SNR is high (known as points of interest (PoIs)), or combining together the trace points to obtain synthetic values with higher SNR. The first approach requires to compute either the SNR or an equivalent score for each time instant. To this purpose, Bhasin et al [23] proved that the use of the Normalized Inter-Class Variance (NICV) index is a low computational footprint method to detect PoIs for which it holds NICV $= 1/(1+\frac{1}{\text{SNR}})$. A score similar to NICV, widely used, albeit not directly relatable to the SNR is the Sum Of Squared T (SOST) differences proposed by [24], deriving its name to the relation with the score of Student's test. The algorithm considered to be the most effective in the literature for the second approach is the Principal Component Analysis (PCA). PCA considers the samples within a trace as a set of interrelated random variables, and aims to reduce their number while retaining the information included in their variation. This is done transforming the original data into a new set of variables, known as *principal components*, which are mutually uncorrelated, and sorted in decreasing order of their variation. This linear transformation has the potential to single out into few components the signal variation due to the computations, decoupling it from the signal variation due to signal-independent additive noise.

III. Proposed Architecture

We describe the principles of our countermeasure relying on introducing algorithmic noise in the side channel signal, interleaving in time two computations of a block cipher. We define the reference computation pattern of a block cipher with iterative structure. Subsequently, we define the properties that our countermeasure will fulfill, and detail the realization with AES-128/-256 as a case study.

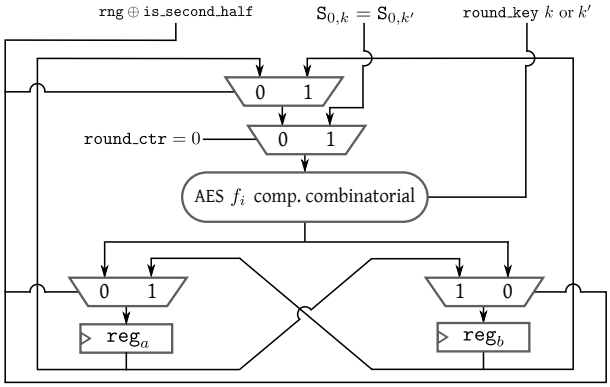

Fig. 2. Architectural view of of a single CIU unit for AES

A. SCA Countermeasure Strategy

Definition 1 (Reference computation pattern of an iteratively computed block cipher): An iteratively computed block cipher is realized through the execution of a sequence of $n > 0$ Boolean functions $f_0(\cdot), \ldots, f_{n-1}(\cdot)$, each of which is applied in a single clock cycle to a binary string $\mathtt{S}$ known as the cipher state. Therefore, function $f_{i+1}(\cdot)$ takes as input the output value of $f_i(\cdot)$, $0 \leq i \leq n-2$. The sequential application of the n functions is logically organized into r rounds, each of which may prescribe the application of a different number of functions. One or more functions per round take as additional input the bitstring corresponding to the so-called *round key*s: k, $k.1$, $k.2$, $\ldots$ $k.r$, which are in turn prescribed by the KeySchedule algorithm of the cipher, starting from the user key k. We will denote the states traversed by a cipher computation as $\mathtt{S}_{i,k}$, where i is an integer matching the one of the Boolean function to be applied next, and k is the user key bound to the current computation of the cipher. The states $\mathtt{S}_{0,k}$ and $\mathtt{S}_{n,k}$ denote the initial input and the final output state when employing the user key k (i.e., the plaintext/ciphertext pair in an encryption computation). The $\mathtt{S}_{i,k}$ values are stored in a memory element, which we denote as $\texttt{reg}$: its contents drive the datapath containing the combinatorial implementations of the Boolean functions at each function computation.

Secure Design Properties. To obtain the desired effect of blending data dependent noise into the computation of a block cipher as per Definition 1, and providing protection against first-order side channel attacks, we interleave two block cipher computations acting on the same plaintext and on two different keys, k, and k'. While k is the actual user key, k' is derived from a device-dependent computation such that: i) given k it is not feasible for an attacker to derive k', ii) if k is generated as uniformly distributed over $\{0,1\}^{|k|}$, then so is k'. Such a key generation can be obtained either using k as a challenge for a strong PUF, which emits k' as a response, or feeding k as the key to be refreshed into an LWPE protected key refreshing mechanism [8], where the secret parameter of the LWPE module acts as the device- dependent component. Considerations on the most efficient strategy to implement the derivation of k' given k are out of scope for this work.

We consider an interleaving of two block cipher computations on the same datapath, thus requiring the duplication of the state-value-holding register. We will denote the two registers holding the states as $\texttt{reg}_a$ and $\texttt{reg}_b$. To achieve the effectiveness of our countermeasure, the architectural interleaving realization should respect the following properties:

Property 1 (P1): the order in which any given function f_i is sequentially applied to $\texttt{S}_{i,k}$ and $\texttt{S}_{i,k'}$ must be randomized, as doing otherwise would allow an attacker to easily separate their contribution in time

Property 2 (P2):it must never hold that two subsequent states of the same cipher computation, e.g., $\texttt{S}_{i,k}$ and $\texttt{S}_{i+1,k}$, are stored one after the other in the same register, as this would result in a first-order non profiled attack vulnerability from the register switching activity;

Property 3 (P3):each combinatorial Boolean function in the datapath should never be driven by two subsequent values of the computation of the same cipher, e.g., $\texttt{S}_{i,k}$ and $\texttt{S}_{i+1,k}$, in two consecutive clock cycles, as this would result in a first-order non profiled attack vulnerability from the combinatorial logic switching activity.

Computation Interleaving Unit. We design a Computation Interleaving Unit (CIU) so that it enjoys the aforementioned properties, while minimizing the number of involved registers. A simplified timing diagram of the hardware unit behaviour is depicted in Fig. 1. First of all, fulfilling P1 requires that the order in which $f_i(\texttt{S}_{i,k})$ and $f_i(\texttt{S}_{i,k'})$ are computed is randomly chosen: we therefore employ a RNG to select which one of the two computations is performed first. We note that every other computation of f_i is uniquely determined: if $f_i(\texttt{S}_{i,k})$ was computed first, $f_i(\texttt{S}_{i,k'})$ will be computed afterwards and vice-versa. This requires one random bit every two cycles.

Fulfilling P2 is achieved by our design taking care of alternating which computation is being stored in a given register at each clock cycle. This is achieved with two fixed update rules, depending on which computation between $f_i(\texttt{S}_{i,k})$ and $f_i(\texttt{S}_{i,k'})$ goes first. In particular, if $f_i(\texttt{S}_{i,k})$ is selected to be the first computation (left timing diagram in Fig. 1), $\texttt{reg}_a$ is filled with the contents of $\texttt{reg}_b$, i.e., $\texttt{S}_{i,k'}$, while $\texttt{reg}_b$ receives the result of $f_i(\texttt{S}_{i,k}) = \texttt{S}_{i+1,k}$. The same update rule is applied again during the subsequent computation of $f_i(\texttt{S}_{i,k'})$, bringing back the computation on k in $\texttt{reg}_a$, and the one on k' in $\texttt{reg}_b$. If $f_i(\texttt{S}_{i,k'})$ is the first one to be computed in the computation of f_i (as in the computation of f_1 in the right timing diagram in Fig. 1), the destination registers in the previous rule are swapped, yielding the behaviour depicted in clock cycles 2 and 3 in the right diagram of Fig. 1. As a result, regardless of the order being chosen for the computation of $f_i(\texttt{S}_{i,k})$ and $f_i(\texttt{S}_{i,k'})$ no register experiences a transition from a value of the computation on k and a value of the computation on k' directly, thus resulting in a switching activity (and information transmission) always depending on both computations preventing first-order non-profiled leakage. The correctness of the strategy can be easily observed considering all f_is identity functions. In this case, the alternating rule just keeps on swapping the contents of $\texttt{reg}_a$ and $\texttt{reg}_b$, which start by containing states belonging to different computations.

Finally, willing to fulfill P3, we analyze the behaviour of our CIU for two subsequent f_i, f_{i+1} computations. Figure 1 reports two of the four possible sequences of computations for f_i and f_{i+1}, namely the one where the same ordering between the computation on k and on k' takes place both for f_i and f_{i+1} (left) and the one where opposite orderings are selected (on the right). We note that the same line of reasoning we now follow applies to the remaining two cases thanks to the symmetry of the operating logic: they correspond to a simple renaming of the keys. Observing the sequence of values driving the combinatorial component which computes f_i (fourth row from top of the timing diagram) we note that P3 is violated whenever the last state on which f_i is computed belongs to the same computation of the first state on which f_{i+1} (highlighted in grey in Fig. 1). Preventing this unwanted violation by restricting the possible sequences of computations is not possible, as it would imply violating P1. We solve this issue adopting a strategy from the *random precharging* countermeasure. Random precharging consists in briefly driving the inputs of either combinatorial cones or write lines of a register with random values, before actually driving them with relevant values. This causes an uncorrelated switching activity for the component, removing first-order leakage. In our case, we precharge the combinatorial component with the contents of $\texttt{reg}_b$ whenever the first computation of a pair is the one on k' and with $\texttt{reg}_a$ otherwise. This precharging has no adverse effect when P3 is not violated, while it actually removes the P3 violation whenever happening.

B. Architectural Countermeasure Design

We now consider a concrete implementation of a CIU, taking as our case study the AES-128/-256 block cipher. AES-128 (equiv. AES-256) is composed of $r = 10$ (equiv. $r = 14$) rounds, of which the first 9 (equiv. 13) have the same structure, while the last round computes a different function. The r rounds are preceded by a single addition of the first round key, via xor, to the plaintext. As our CIU design is generic, we chose the same unprotected AES implementation as the one of Scramble Suit, as it is both compliant with Definition 1, and allows a fair comparison. This implementation considers each one of the r rounds of AES as split into five functions each: the first four compute the SUBBYTES transform on four state bytes at once, while the last function computes all the remaining AES round primitives (SHIFTROWS, MIXCOLUMNS and ADDROUNDKEY) as a single combinatorial net. The key addition before the first round is treated as the first Boolean function f_0 to be computed, while the last function of the last AES round omits the MIXCOLUMNS as prescribed by the standard. Figure 2 reports the structure of our CIU unit, omitting the control signals, which select which AES f_i function is to be computed depending on the round counter for the sake of clarity. The design relies on the value of a single RNG bit driving the selection of which computation should be performed, in combination with a

TABLE I

FPGA RESOURCE USAGE, WITH AND WITHOUT THE RESOURCES FOR THE SCRAMBLING KEY GENERATOR. RESOURCES ARE SHOWN AS ABSOLUTE VALUES AND PERCENTAGES OF THE ENTIRE DEVICE RESOURCES, WITH THE FREQUENCY (f) AND THE CRITICAL PATH SLACK (CPS)

Circuit	Scrambling key gen.	LUTs No. (%)	FFs No. (%)	f (MHz)	CPS (ns)
plain AES-128/-256	–	3,997 (6.3 %)	3,392 (2.9%)	100	0.85
Our AES-128/-256	×	5,938 (9.4 %)	5,644 (4.5%)	100	0.12
1 **CIU unit**	✓	7,690 (12.1%)	8,629 (6.8%)	100	0.24
Our AES-128/-256	×	12,022 (18.9%)	11,207 (8.8%)	95	0.66
2 **CIU units**	✓	17,189 (27.1%)	18,610 (14.7%)	95	0.68
Our AES-128/-256	×	25,828 (40.7%)	22,831 (18.0%)	85	0.44
4 **CIU units**	✓	38,357 (60.5%)	38,565 (30.4%)	85	0.48
plain AES [6] **Spartan** 6 **FPGA** ***AES***-128 ***Encryption only***	–	8,700 (18%)	3,081 (3%)	48	1.72
Scramble Suit [6] **Spartan** 6 **FPGA** ***AES***-128 ***Encryption only***	×	11,201 (24.0%)	5,464 (5.0%)	48	3.70

single-bit counter, `is_second_half`, to select both the driving value for the combinatorial path, and the location into which the result should be stored. The precharging required to fulfill P3 is performed acting on the control of the topmost mux of Fig. 2, so that the AES combinatorial logic is driven by the correct register. The plaintext $\mathrm{S}_{0,k} = \mathrm{S}_{0,k'}$ is loaded with a dedicated path, which is only enabled at the beginning of the computation to avoid potential glitches in the multiplexer at the end of the AES f_i combinatorial component.

Adding Spatial Redundancy. Given the depicted CIU design, it is possible to scale-out the amount of device dependent side channel distortion introduced by means of multiple CIU units running in parallel. All but one of the computations being performed in this case will be employing keys generated from a user key, and a separate PUF/device dependent key generator. To prevent an attacker from separating the contribution of the parallel CIU units, the mapping of the computations onto the CIU modules is randomized after each AES execution. To achieve this, a register keeping the locations of the current computations is generated at synthesis time, fitting the number of CIUs. The indexes contained in it are randomized employing a modified bitonic sorting network, which, instead of comparing the index values, employs a randomly generated bit as the comparison outcome. The sorting network architecture allows us to compute a random permutation in a single clock cycle. This in turn raises the randomness requirement by 2 bits per AES computation, if two CIUs are present, and by 4 bits per computation with four CIUs, in addition to the baseline cost of 0.5 per clock cycle for each CIU to fulfill P1.

IV. EXPERIMENTAL RESULTS

We chose as target platform the Digilent Arty A7 board, based on the Xilinx Artix-7 A100 (`xc7a100tcsg324-1`) FPGA. We implement our countermeasure in the SystemVerilog HDL, performing a classic manual design of the CIU, with a controller-datapath structure. We employ the on-board 100 MHz quartz oscillator for clock generation, performing both the synthesis and implementation of our SystemVerilog design with AMD/Xilinx Vivado HLx 2020.2.

TABLE II

COMPARISON OF MULTIPLE STATE-OF-THE-ART FIRST-ORDER MASKING IMPLEMENTATIONS AND OUR WORK. WE REPORT THE AREA OVERHEAD WITH RESPECT TO THE UNPROTECTED IMPLEMENTATION AS PERCENTAGE, THE RANDOMNESS PER CLOCK CYCLE AND THE CLOCK LATENCY.

Countermeasure approach	Datapath width	Area overhead(%)	Randomness (bit/clock cycle)	Clock cycles
De Cnudde *et al.* [9]	8	256	54	276
De Meyer *et al.* [10]	8	219	19	256
Shahmirzadi *et al.* [11]	8	274	1	246
Our work	32	66	0.5	102

We instructed Vivado with the following directives. Synthesis phase: keep equivalent registers, avoid resource sharing and no hierarchy flattening; implementation phase: `explore` optimization directive. We avoided BRAMs in our design with the intent of providing a design which may be fit both on FPGAs and ASIC targets alike, as we have no requirements for large memories. Our choice directly exposes the cost for the memory elements as FFs in table Table I. All our designs were tested for functionality deploying the bitstream on the actual FPGA. Our AES design includes encryption, decryption, and key scheduling for both 128 and 256 bit key sizes with the same design. To perform our evaluation we implemented two ancillary components: the scrambling key generator, and a RNG. The scrambling key generator is realized with an AES component taking as plaintext the user key, and employing as device-specific element a fixed key for each instance. The RNG is built with a 32b long LFSR with maximum period.

Table I reports the required resources and attained working frequency of our design. A single CIU protected design requires 48% more LUTs and 66% more FFs than its unprotected counterpart, while reaching the same working frequency (100 MHz). To provide an overview on the efficiency of our approach, Table II provides a comparison with the state-of-the art results in masking countermeasures, providing protection against both profiled and non-profiled attacks. All the reported masking schemes have an area overhead in the 219% – 274% range, i.e., three to four times higher than our solution. Comparing the requirements on the random number generator, we note that our CIU approach has a $38\times$ – $104\times$ lower pressure than traditional masking schemes such as the ones of [9] and [10], and is also able to halven the requirements with respect to masking schemes designed to employ the minumum amount of randomness such as [11]. Finally, we note that our CIU based design allows us to realize a 32-bit datapath AES engine with less resources than the masked counterparts, in turn obtaining a better latency with respect to the 8-bit datapaths employed in designs in Table II.

Multiple-CIU designs scale almost linearly in resource demands, with the 4 CIU design requiring little less than twice the resources of a 2 CIU design. The higher resource gap between a single CIU design and a multiple one is due to the external randomization structure distributing computations across CIUs. No direct comparison can be made with [6], due to target device differences: Scramble Suit [6] is implemented with Xilinx ISE on a Spartan-6 XC6SLX75-2 FPGA, which is

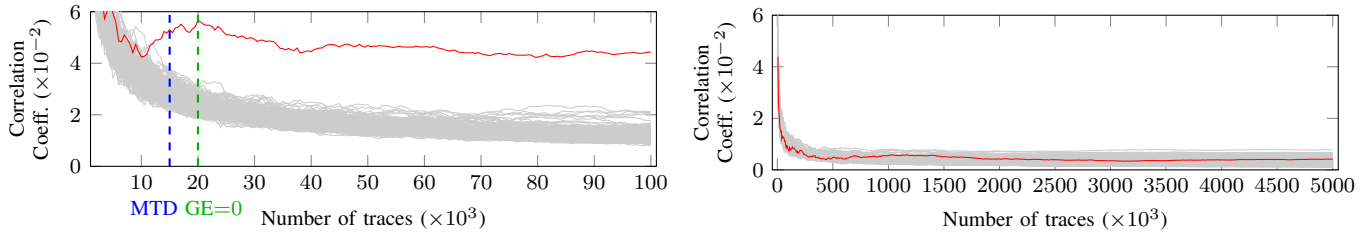

Fig. 3. Non-profiled CPA against an unprotected (left) and protected implementation (right): correlation coefficient as a function of the number of traces. Measurements to disclose for the unprotected implementation (99.9% confidence level): 15k, blue line; amount of traces where the GE is null for all 5 experiments: 20k, green line. For the protected implementation 5M traces do not allow the CPA to succeed; GE > 0 in 20 experiments with 250k traces

not supported by Vivado. However, we note that we obtain a lower overhead in FFs (66% vs 77%), while Scramble Suit [6] achieves a better margin with respect to LUTs usage(28% vs 48%). We also note that the combinatorial logic of the base design in Scramble Suit appears to have been less optimized by ISE, as the critical path slack of the protected implementation is twice as high as the one of the unprotected one. The gap in the FF usage between us and Scramble Suit is to be ascribed to the longer keyschedule required by AES-256. Finally, that our design achieves roughly twice the working frequency of Scramble Suit (albeit on a newer FPGA model).

Experimental workbench. We validate the effectiveness of our countermeasure collecting EM emission traces with a 100A Beehive near-field probe placed on top of the Artix-7 A100 package front. The signal is amplified via two cascaded Agilent INA-02186 (31.5 dB gain each) and measured by a Picoscope 5244D DSO sampling at 500 Msamples/s. The trigger signal for the oscilloscope is generated driving a GPIO pin of the Arty board. To avoid signal pollution due to the EM emissions of the GPIO lines, we employed a hardware counter to delay the start of the computation by a fixed amount of cycles. We experimentally observed a perfect trace alignment at 500 MSamples/s and no visible clock jitter. For each one of our design instantiations ($1, 2$ and 4 CIUs), we synthesized three different bitstreams (labelled as a,b, and c in the following), randomly generating the device-dependent keys employed in the scrambling key generation unit.

Non-profiled attack security validation. We collected 5M measurements for each bitstream and performed a CPA. We employed as a synthetic model the Hamming distance between the first byte in input to the first SUBBYTES primitive and the output byte of the same SUBBYTES, matching the switching activity of the unprotected state register. Figure 3 reports the results of the CPA against the unprotected baseline (left) and the 2-CIU design, bitstream a (right). The MTD value for the unprotected design is 15k traces, considering a 99.9% confidence in the correlation of the correct key being the highest. By contrast, performing a CPA on the protected implementation does not disclose the key value even when all the 5M traces are employed, thus yielding a $> 333\times$ increase in the MTD metric. The same results are achieved on all protected bitstreams, we omit the figures for space reasons. Concerning the security evaluation with the GE metric, we performed CPA attacks with single key byte hypothesis: we thus have that the possible values taken by the GE are between 0 (successful attack) and 255 (the correct key is systematically reported with the lowest rank). We note that uniformly randomly drawn key-ranks in our scenario (i.e., the result of a perfectly unsuccessful attack) have a GE of 127.5, with a standard deviation for the key rank of 73.61. We report that the GE obtained performing 5 experiments (i.e., CPAs) on the unprofiled device, each one with 20k traces is 0 with standard deviation 0, i.e., all attacks are systematically successful. Conducting 20 CPAs, each one with 250k traces each on our single-CIU protected device, we obtain a GE of 129.9, with a standard deviation of the key rank equal to 68.62 (with negligible variance across the three single CIU devices).

Profiled attack security validation. We evaluate the resistance against profiled attacks employing TAs, and four different feature reduction approaches: sample selection with the maximum SOST/NICV score, PCA, and taking all the samples. We note that the only obstacle to template portability in our setup is our countermeasure: the setup employed to collect traces is untouched between different collections, no clock jitter or misalignment is present, we verified the absence of DC drift due to thermal effects, and no process-variation is present as we employ the same Arty A7 board to record all the measurements. For all device instances, we build two device profiles, selecting the first round key bit as the target of our attack, employing 2k traces to build the device profile. We build multi-device models computing the profiles from the interleaving of 2k traces for each protected device instance pair. Subsequently, we classify 20k traces (none in common with the set used in the modeling phase) coming from different device instances. Table III reports the outcome of both single- and multi-device model TA results. TAs succeed with perfect accuracy against the unprotected device instance, while they obtain an accuracy in the 47.9%–51.7% range for all protected implementations, regardless of the use of a single-device model or an MDM. This provides strong evidence of the non-effectiveness of TAs against our countermeasure, as the fluctuations around the accuracy of a random guess are within a range of plausibility for statistical artifacts. Table III reports the effectiveness results against TAs of Scramble Suit: we achieve the same effectiveness, as it can be noticed by the small fluctuations in the results of Scramble Suit (when

TABLE III

CLASSIFICATION ACCURACY OF TAS VS. AN UNPROTECTED ($\mathbf{u}$) AND 3 PROTECTED DEVICES ($\mathbf{a}, \mathbf{b}, \mathbf{c}$) WITH 1, 2 AND 4 CIU EACH. MOD. IS THE DEVICE(S) EMPLOYED TO BUILD THE PROFILE; EXP. LISTS ATTACKED ONES. TEMPLATES WITH 2K TRACES; ATTACKS PERFORMED AGAINST 20K TRACES PER DEVICE. RESULTS FROM [6] FOR COMPARISON

No. of CIU	Mod.	Exp.	Accuracy of TA with feature selection (%) None	NICV	SOST	PCA
0	**u**	**u**	100.0	100.0	100.0	100.0
1	**a**	**b, c**	47.9, 49.4	50.3, 49.9	49.7, 50.1	49.9, 50.4
	b	**a, c**	48.7, 50.0	50.5, 49.1	50.2, 51.3	49.9, 50.0
	c	**a, b**	49.9, 49.9	49.9, 50.1	50.0, 50.0	50.0, 50.0
	ab	**c**	50.1	49.7	48.5	49.8
	bc	**a**	49.2	48.3	50.2	50.0
	ac	**b**	49.1	49.3	50.1	50.0
2	**a**	**b, c**	50.1, 50.1	50.2, 49.6	50.7, 50.5	50.0, 50.0
	b	**a, c**	50.7, 50.5	50.0, 49.9	50.2, 49.7	50.0, 50.0
	c	**a, b**	50.2, 49.9	49.7, 50.1	49.5, 49.8	50.0, 50.0
	ab	**c**	50.1	48.8	50.1	50.0
	bc	**a**	49.7	50.0	51.0	49.9
	ac	**b**	50.1	49.3	49.6	50.0
4	**a**	**b, c**	49.8, 51.7	50.3, 50.4	51.1, 49.9	48.7,50.0
	b	**a, c**	49.2, 48.8	49.9, 49.2	50.8, 50.0	49.8,50.0
	c	**a, b**	50.9, 50.3	49.6, 50.0	49.2, 49.7	50.0,49.8
	ab	**c**	49.0	50.0	49.8	50.0
	bc	**a**	49.8	49.2	49.9	49.7
	ac	**b**	50.1	50.1	50.4	49.9
Scramble Suit [6] (1 equiv. CIU)	**a**	**a**	n.a.	n.a.	98.5	100.0
	b	**b**	n.a.	n.a.	50.3	93.5
	a	**b**	n.a.	n.a.	50.0	50.2
	b	**a**	n.a.	n.a.	50.0	50.0

employing their best feature selection approach, PCA). We note that, in our scenario where single-bit (i.e., two-class) template attacks are performed, the Guessing Entropy and the accuracy scores coincide. Indeed, computing the GE in our template attacks (with the possible key ranks being only 0 and 1) yields 0.5 as the average rank.

Security scalability. While our solution with a single CIU already withstands first-order non-profiled and profiled attacks, as multi-CIU designs do, we note that adding multiple CIU units further increases the inter-virtual device distance between the profiles. To provide quantitative substantiation to this claim, we computed the Euclidean distance between the mean vectors of the templates for our virtual devices. Increasing the number of CIUs increases the Euclidean distance between the means of the templates by $\approx 50\%$. To put this increase in perspective, we report that the Euclidean distance between two means of the templates of the same device with different key values is roughly $10\times$ smaller than any of the distances between the means of the templates coming from different devices with a single CIUs. As a consequence, we have that devices with two CIUs have profiles which are $15\times$ farther apart than the distance caused by key values, while devices with 4 CIUs have template means which are $\approx 33\times$ farther apart than the distance caused by the key values.

V. CONCLUDING REMARKS.

We presented a SCA countermeasure proving it robust both against single/multi device model TAs and non-profiled attacks. Our countermeasure yields a $\geq 333\times$ increase in the MTD metric vs. an unprotected implementation, and results in TAs having an $\approx 50\%$ accuracy in retrieving a correct key bit. Our solution provides a less expensive alternative, when put in the ballpark of masked implementations. In particular, we obtain a 66% area overhead, comparing favourably with the 219% – 274% overheads of masking countermeasures [9]–[11], and require $38\times$ to $104\times$ less RNG throughput w.r.t. traditional masking schemes [9], [10], and $2\times$ less RNG throughput w.r.t. the state-of-the-art [11].

REFERENCES

[1] M. A. Elaabid and S. Guilley, "Portability of templates," *J. Cryptogr. Eng.*, vol. 2, no. 1, 2012.

[2] D. P. Montminy, R. O. Baldwin, M. A. Temple, and E. D. Laspe, "Improving cross-device attacks using zero-mean unit-variance normalization," *J. Cryptogr. Eng.*, vol. 3, no. 2, 2013.

[3] O. Choudary and M. G. Kuhn, "Template Attacks on Different Devices," in *COSADE 2014*, ser. LNCS, vol. 8622. Springer, 2014.

[4] D. Das, A. Golder, J. Danial, S. Ghosh, A. Raychowdhury, and S. Sen, "X-DeepSCA: Cross-Device Deep Learning Side Channel Attacks," in *DAC 2019*. ACM, 2019.

[5] S. Bhasin, et al., "Mind the Portability: A Warriors Guide through Realistic Profiled Side-channel Analysis," in *NDSS 2020*. The Internet Society, 2020.

[6] A. Barenghi, W. Fornaciari, G. Pelosi, and D. Zoni, "Scramble Suit: A Profile Differentiation Countermeasure to Prevent Template Attacks," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 39, no. 9, 2020.

[7] R. Maes and I. Verbauwhede, "Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions," in *Towards Hardware-Intrinsic Security*. Springer, 2010.

[8] D. Bellizia, C. Hoffmann, D. Kamel, P. Méaux, and F. Standaert, "When Bad News Become Good News Towards Usable Instances of Learning with Physical Errors," *IACR TCHES*, vol. 2022, no. 4, 2022.

[9] T. De Cnudde, O. Reparaz, B. Bilgin, S. Nikova, V. Nikov, and V. Rijmen, "Masking aes with shares in hardware," in *CHES 2016, Santa Barbara, CA, USA, August 17-19*. Springer, 2016, pp. 194–212.

[10] L. De Meyer, O. Reparaz, and B. Bilgin, "Multiplicative masking for aes in hardware," *IACR TCHES*, pp. 431–468, 2018.

[11] A. R. Shahmirzadi and A. Moradi, "Re-consolidating first-order masking schemes: Nullifying fresh randomness," *IACR TCHES*, 2021.

[12] E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model," in *CHES'04*, ser. LNCS, vol. 3156. Springer, 2004.

[13] L. Batina, B. Gierlichs, E. Prouff, M. Rivain, F. Standaert, and N. Veyrat-Charvillon, "Mutual Information Analysis: a Comprehensive Study," *J. Cryptol.*, vol. 24, no. 2, 2011.

[14] A. Heuser et al., "Good Is Not Good Enough - Deriving Optimal Distinguishers from Communication Theory," in *CHES'14*, ser. LNCS, vol. 8731. Springer, 2014.

[15] E. de Chérisey, et al., "Best information is most successful mutual information and success rate in side-channel analysis," *IACR TCHES*, vol. 2019, no. 2, 2019.

[16] M. Rivain, "On the Exact Success Rate of Side Channel Analysis in the Gaussian Model," in *SAC 2008*, ser. LNCS, vol. 5381. Springer, 2008.

[17] S. Picek, A. Heuser, A. Jovic, S. Bhasin, and F. Regazzoni, "The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations," *IACR TCHES*, vol. 2019, no. 1, 2019.

[18] J. Coron, D. Naccache, and P. C. Kocher, "Statistics and secret leakage," *ACM Trans. Embed. Comput. Syst.*, vol. 3, no. 3, pp. 492–508, 2004.

[19] S. Chari, J. R. Rao, and P. Rohatgi, "Template Attacks," in *CHES'02*, ser. LNCS, vol. 2523. Springer, 2002.

[20] H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking Cryptographic Implementations Using Deep Learning Techniques," in *SPACE 2016*, ser. LNCS, vol. 10076. Springer, 2016.

[21] E. Cagli, C. Dumas, and E. Prouff, "Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Pre-processing," in *CHES'17*, ser. LNCS, vol. 10529. Springer, 2017.

[22] J. Kim, S. Picek, A. Heuser, S. Bhasin, and A. Hanjalic, "Make Some Noise. Unleashing the Power of Convolutional Neural Networks for Profiled Side-channel Analysis," *IACR TCHES*, vol. 2019, no. 3, 2019.

[23] S. Bhasin, et al., "Side-channel leakage and trace compression using normalized inter-class variance," in *HASP 2014*. ACM, 2014.

[24] B. Gierlichs, K. Lemke-Rust, and C. Paar, "Templates vs. Stochastic Methods," in *CHES'06*, ser. LNCS, vol. 4249. Springer, 2006.