Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2022.0122113

A Tale of Resilience: On the Practical Security of Masked Software Implementations

LORENZO CASALINO[1], NICOLAS BELLEVILLE[1], DAMIEN COUROUSSÉ[1], KARINE HEYDEMANN[2][3]
[1]Univ. Grenoble Alpes, CEA, List, F-38000 Grenoble, France (e-mail: firstname.lastname@cea.fr)
[2]Thales DIS, France (e-mail: firstname.lastname@thalesgroup.com)
[3]Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Corresponding author: Nicolas Belleville (e-mail: nicolas.belleville@cea.fr).

This work was partially funded by the French National Research Agency (ANR) under grant agreement ANR-20-CE39-0010.

ABSTRACT Masking constitutes a provably-secure approach against side-channel attacks. However, *recombination effects* (e.g., transitions) severely reduce the proven security. Concerning the software domain, CPU microarchitectures encompass techniques improving the execution performances. Several studies show that such techniques induce recombination effects. Furthermore, these techniques implicitly induce some form of parallelism, and the potential associated threat has never been investigated. In addition, the *practical* security of masking relies on the chosen masking scheme. Few works analysed the security of software protected by different masking schemes, and none considered the parallelism threat. Thus, literature lacks of a more comprehensive investigation on the *practical* security of software implementations relying on various masking schemes in presence of micro-architecture-induced recombination effects and parallelism. This work performs a first step to fill this gap. Specifically, we evaluate the practical security offered by first-order *Boolean*, *arithmetic-sum* and *inner-product* masking against transitions and parallelism in software. We firstly assess the presence of transition and parallel-based leakages in software. Secondly, we evaluate the security of the encodings of the selected masking schemes with respect to each leakage source via micro-benchmarks. Thirdly, we assess the practical security of different AES-128 software implementations, one for each selected masking scheme. We carry out the investigation on the STM32F215 and STM32F303 micro-controllers. We show that (1) CPU's parallel features allow successful attacks against masked implementations resistant to transition-based leakages; (2) implementation choices (e.g., finite field multiplication) impact on the practical security of masked software implementations in presence of recombination effects.

INDEX TERMS masking, processor micro-architecture, side-channel analysis, software masking

I. INTRODUCTION

Side-channel attacks threat the security of embedded hardware and software components, in particular cryptographic primitives. To counteract a side-channel attacker, an nth-order masking countermeasure encodes secret-dependent data into $n+1$ random values, called *shares*. Under the assumption of independently leaking shares (ILA) and of sufficient noise, a successful attack requires the computation of higher-order statistical moments of the encoding's distribution. The difficulty of this task increases exponentially in n [1], defining the *security order* of masking. In practice, physical phenomena such as (memory-state) transitions [2], glitches [3] and wire cross-talking [4] *recombine* shares, reducing the expected security order of a masked implementation [3], [5].

Several works highlight the pervasiveness of transition-based leakages in CPU micro-architectures [6], [7], [3], [2], [8]. Still, few works show the *practical* security difference between different maskings in presence of transition-based leakages [9] [10].

Besides transitions, the threat posed by parallel processing of shares (PPS) is overlooked for software-masked implementations[1], although Moos and Moradi show a simple pre-processing technique to efficiently exploit it in hardware [12]. In software, the micro-architecture of modern CPUs relies

[1]To the best of our knowledge, the only work mentioning the existence of PPS in software is [11], in footnote 1, but this work does not study the security implications further.

on techniques to increase execution performance [13], potentially handling multiple shares per clock cycle. As such, the PPS implies new potential vulnerabilities in masked software implementations. To the best of our knowledge, the study of such vulnerabilities remains unexplored.

This work explores the practical security of several first-order software masked implementations in presence of *both recombination effects and PPS-based leakages*. We study three masking schemes: the most studied *Boolean* (BM) [14] and *inner-product* (IPM) [15], and the *arithmetic-sum* (ASM) [16]. Our investigation firstly assesses the potential sources of vulnerabilities in software due to transition-based and PPS-based leakages, and then evaluates the practical security of masked software with respect to the identified vulnerabilities. In more details, our methodology develops in three steps:

1) We characterise micro-architectural leakage effects: we carefully handcraft micro-benchmarks to assess the presence of transition-based and PPS-based leakages in software (Section IV).
2) We characterise the impact of the observed leakage effects on masking encodings: we quantify the leaked information and investigate its exploitability (Section V, Section VI).
3) We characterise the impact of the observed leakage effects on masked implementations: once evaluated leakage impact on the encodings, we assess the practical security of fully masked software implementations (Section VII). Specifically, we target as a use-case the AES-128 block-cipher [17].

To provide a comprehensive analysis, we split the security assessment in a first *information leakage assessment*, to analyse the information leaked by the encoding or the fully masked implementation, and in a *information leakage exploitation*, to evaluate the exploitability of such information. In addition, as the design and implementation of the execution platform potentially impacts the observed leakage [2], [18], [19], we lead our investigation on two different micro-controllers, an STM32F215 and STM32F303.

CONTRIBUTIONS

To the best of our knowledge, we provide the first investigation on the practical security of different masking schemes in software in presence of both recombination effects and PPS. In particular we show that:

- PPS-based leakage is observable in the software context (Section IV).
- PPS induces information leakage for all the considered masking encodings (Section V).
- Such leakage can be exploited against all the considered masking schemes by slightly adapting Moos and Moradi methodology [12] (Section VI).
- Transitions and PPS-based leakages lead to successful attacks against all the considered masked implementations. In particular, we exhibit two attacks against inner-product masking: one exploiting PPS leakages and one exploiting a vulnerability due to transition-based leakage on the logarithm representation of the encodings within the finite field multiplication implementation (Section VII).

II. RELATED WORK

The nature of this work touches different areas of the side-channel domain. This section compares our work with the most relevant ones from each area.

a: Micro-architecture-induced Leakages

Several works investigate the different micro-architectural sources of leakage, spanning through different micro-architectures and processors. Table 1 summarises the related state of the art, highlighting the investigated leakage sources, types of micro-architecture and the CPU use-case(s). Papagiannopoulos and Veshchikov assess some recombination effects (i.e., register overwrite and memory persistence) violating the ILA on a simple AVR ATMega163 micro-controller [20]. Marshall et al. assess the presence of multiple transition-based leakages on different platforms [2]. They highlight how similar platforms, executing the very same piece of code, may exhibit or not a transition-based leakage. Furthermore, they highlight how speculative execution potentially introduces unexpected transition-based leakage. With ARMISTICE, de Grandmaison et al. show how also the encoding of instructions potentially affects the variability of the observable leakages [8]. Concerning platforms provided with *superscalar* capabilities, Barenghi and Pelosi show, on the ARM Cortex-A7 and ARM Cortex-M7, that the increased parallelism provided by such micro-architectures increase the sources of transition-based leakage [6], [7]. Besides the pervasiveness of transition-based leakage, a micro-architecture potentially encompasses other recombination effects. Gao et al. show that intra-register leakage interaction can break the security of share-slicing implementations [21], suggesting that leakage is due to glitch-based recombinations in the barrel-shifter unit of ARM Cortex-M0 and Cortex-M3 implementations. On the contrary, Gigerl et al. show how signal glitches in the forwarding logic of the superscalar RISC-V SweRV CPU recombine multiple shares, reducing the masking security order beyond the factor of 2 predicted by the theory [3]. As a matter of fact, the current state of the art focuses on micro-architecture-induced recombination effects. Our work represents a novel and orthogonal effort: we show that PPS-based leakage can be observed in software implementations, even on in-order scalar processors.

b: Practical Security of Software Masked Implementations

Few works explore the practical security of masking against micro-architectural leakages. Table 2 summarises the related state of the art, highlighting the investigated masking, the considered leakage sources and the analyses carried on. Beckers et al. show that several deemed-to-be-secure software

TABLE 1: Summary of the state of the art concerning micro-architectural leakage investigations. For each work, we report the targeted leakage source, the type of investigated micro-architecture and the CPUs analysed. With "?" we mark works for which it is unknown whether a given leakage source is targeted.

Work	Leakage Source			μArch Type		CPU(s)			
	Transition	Glitch	PPS	Scalar	Superscalar	AVR	ARM (Cortex)	RISC-V	MicroBlaze
Papagiannopoulos [20]	✓	?	-	✓	-	ATMega163	-	-	-
Barenghi [6]	✓	-	-	-	✓	-	M4, A7	-	-
Gao [21]	-	✓	-	✓	-	-	M0, M4	-	-
Barenghi [7]	✓	-	-	✓	✓	-	M4, M7	-	-
Gigerl [3]	✓	✓	-	-	✓	-	-	SweRV (EH1)	-
Marshall [2]	✓	✓	-	✓	-	-	M0, M0+, M3, M4	PicoRV32	v.10.0.0
de Grandmaison [8]	✓	-	-	✓	-	-	M3	-	-
This paper	✓	-	✓	✓	-	-	M3, M4	-	-

TABLE 2: Summary of the state of the art concerning the practical security analyses of masking in software. For each work, we report the targeted masking scheme, the leakage source against which we evaluate the practical security and the analyses carried on.

Work	Masking			Leakage Source		Analyses		
	BM	ASM	IPM	Transition	PPS	TVLA	CPA	TA
Beckers [10]	✓	-	✓	✓ (BM)	-	✓	✓	-
Wu [9]	✓	-	✓	✓ (BM)	-	✓	✓	✓
This paper	✓	✓	✓	✓	✓	✓	✓	-

implementations, either masked via first-order BM or IPM, are vulnerable to simple first-order analyses (i.e., CPA and TVLA) [10]. Although we follow a similar investigation approach, our goal is different: whereas they aim to verify the claims concerning the security of open-source implementations, we evaluate the practical security of different masking schemes against transition-based and PPS-based leakages. The work of Wu et al. is closer to ours: they analyse the practical security of different *code-based* software instances of BM and IPM, up to the third masking order [9]. They rely on first-order analyses (i.e., CPA and TVLA), as well as Template Attacks (TA) and bivariate CPA. Interestingly, they analyse the practical security of code-based IPM with respect to different public vectors (Section III-B), providing a better characterisation of the enconding's security. Both works do not (explicitly) target micro-architectural leakages, although the leakages they observe for BM implementations probably result from transition-based leakages. In contrast, we explicitly take advantage of transition-based and PPS-based leakages against each studied masking (Section VI and Section VII). Finally, with respect to the two previous works, we consider the ASM, a masking scheme employed for ARX ciphers (e.g., Speck [22]) and post-quantum cryptosystems (e.g., Kyber [23]).

III. BACKGROUND

This section provides the essential background to understand our methodology. We first introduce the notations employed throughout our work. Then, we introduce the masking countermeasures we study, the necessary security concepts and the potential threat implied by so-called *physical effects*. We follow with an overview of three statistical tools we employ to assess and exploit the information leakage from the investigated software implementations. Finally, we overview a trace preprocessing technique able to exploit PPS-based leakages.

A. NOTATIONS

We refer to a random variable with a capital italic letter, e.g., X. We denote the sampling space of X as $\mathcal{X}_{2^k}$, where $k > 0$. We refer to the distribution of a random variable X as $\mathcal{D}_X$. We refer to a realisation of the random variable $X \in \mathcal{X}_{2^k}$ as $x \in \mathbb{F}_{2^k}$, where $\mathbb{F}_{2^k}$ is a finite field. We implicitly consider any value $x \in \mathbb{F}_{2^k}$ in binary form. We denote the i-th bit of x as x^i, where $i \in [0, k)$. We refer to vectors in bold-face style, e.g., $\mathbf{X}$. We refer to the j-th component of a vector $\mathbf{X}$ as $\mathbf{X}_j$. We refer to a set of n traces, each of m samples, with $\mathbf{T}_{n\times m}$, and to the subset of traces at sample $0 \leq i < m$ with $\mathbf{T}^i_{n\times m}$.

B. MASKING

A side-channel attacker exploits the statistical link between an observed physical quantity (e.g., the instantaneous power consumption) and secret-dependent data, which the target implementation manipulates. The masking countermeasure counteracts such attacks by breaking this statistical link. That is, given any secret-dependent datum X, masking *encodes* it with a so-called *(probabilistic) encoding* (Def. 1).

Definition 1. ***(Encoding).*** *Given a random variable $X \in \mathcal{X}_{2^k}$, where $k \geq 1$, the tuple $\mathbf{X} = (X_i)_{i=0}^{n} \in \mathcal{X}_{2^k}^{(n+1)}$ is an encoding of X. The random variables $\mathbf{X}_i \in \mathcal{X}_{2^k}$ are called shares. n defines the masking* order.

The encoding of X is built from an *nth-order masking scheme* M. Informally, an nth-order masking scheme is a vector-valued function $\mathrm{M} : \mathcal{X}_{2^k} \mapsto \mathcal{X}_{2^k}^{(n+1)}$, such that it satisfies *correctness* (i.e., the M function is invertible) and *dth-order security* (Def. 2).

Definition 2. ***(dth-Order Security).*** *Let* M *be an nth-order masking scheme.* M *satisfies* d*th-order security if and only if, for each* $X \in \mathcal{X}_{2^k}$*, any subset of (at most)* d *shares of* $\mathbf{X} = (X_i)_{i=0}^n = \mathrm{M}(X)$ *does not depend on* X. $d \leq n$ *defines the* security *order of* M.

Examples of masking schemes are the *Boolean* masking (BM) [14], the *arithmetic-sum* masking (ASM) [16] and the *inner-product* masking (IPM) [15]. Respectively, they generate *Boolean* (Def. 3), *arithmetic-sum* (Def. 4) and *inner-product* (Def. 5) encodings.

Definition 3. ***(Boolean Encoding).*** *Let us consider* $X \in \mathcal{X}_{2^k}$*, where* $k \geq 1$ *and* $\mathbf{X} = (X_i)_{i=0}^n = \mathrm{BM}(X)$ *the Boolean encoding of* X*. Then* $X = \bigoplus_{i=0}^n \mathbf{X}_i$*, where* $\oplus$ *is the eXclusive OR.*

Definition 4. ***(Arithmetic-Sum Encoding).*** *Let us consider* $X \in \mathcal{X}_{2^k}$*, where* $k \geq 1$ *and* $\mathbf{X} = (X_i)_{i=0}^n = \mathrm{ASM}(X)$ *the arithmetic-sum encoding of* X*. Then* $X = \boxplus_{i=0}^n \mathbf{X}_i$*, where* $\boxplus$ *is the arithmetic sum.*

Definition 5. ***(Inner-Product Encoding).*** *Let us consider* $X \in \mathcal{X}_{2^k}$*, where* $k \geq 1$ *and* $\mathbf{X} = (X_i)_{i=0}^n = \mathrm{IPM}(X)$ *the inner-product encoding of* X*. Then* $X = \langle \mathbf{L}; \mathbf{X} \rangle$*.* $\mathbf{L} = (1, L_i)_{i=1}^n \in \mathcal{X}_{2^k}^{(n+1)}$ *is a public random vector, and* $\langle \cdot; \cdot \rangle$ *is the inner-product operator.*

The appeal in the masking countermeasure lies within the provable security framework, composed of:

- **Leakage Model**: it describes how an implementation leaks information through a given side channel. Typically, the leakage model takes the form of an *Additive Gaussian Noise* (AGN) function:

$$\mathrm{L}(v) = \mathrm{L}(v)_{\mathrm{d}} + \mathcal{N}(0, \sigma) \tag{1}$$

  where $\mathcal{N}(0, \sigma)$ is a *Gaussian* noise, and $\mathrm{L}(\cdot)_{\mathrm{d}}$ is a deterministic function. Typically, $\mathrm{L}(\cdot)$ is a *value-based* leakage function (Def. 6), such as the *Hamming-Weight* function:

$$\mathrm{L}(v) = \mathrm{HW}(v) + \mathcal{N}(0, \sigma) \tag{2}$$

  where HW is:

$$\mathrm{HW}(x) = \sum_{0 \leq i < k} x^i. \tag{3}$$

- **Attacker Model**: it describes how many intermediate variables the attacker can observe (we distinguish between univariate and multivariate) and the maximum statistical moment order the attacker can compute.

Definition 6. ***(Value-based Leakage Function [5]).*** *Let* $\mathbb{V}$ *be a finite set of intermediate variables and* $\mathrm{L}(\cdot) = \mathrm{L}(\cdot)_{\mathrm{d}} + \mathcal{N}(0, \sigma)$ *be a leakage function made of a deterministic part* $\mathrm{L}(\cdot)_{\mathrm{d}}$ *and an (additive) random noise* $\mathcal{N}(0, \sigma)$*. This leakage function is value-based if its deterministic part can only take a value* $v \in \mathbb{V}$ *as argument.*

Under the so-called *Noisy Leakage* security model (parallel computation, value-based leakage, univariated attacker), Chari et al. proved that the masking countermeasure exponentially amplifies (in the number n of shares) the difficulty of an attack, expressed in number of traces to collect and analyse [1]. Ishai et al. defined the *d-probing security model* (value-based leakage, multivariated attacker), under which an implementation is secure against any d-variated attacker [24]. Barthe et al. defined the *Bounded-Moment* (value-based leakage, univariated attacker), which proves the security of masked implementations against attackers able to compute statistical moments of order (up to) d [11].

C. PHYSICAL EFFECTS

Security proofs of masking schemes typically assume a value-based leakage model, i.e., each share leaks independently of the others. The literature refers to it as the *Independent Leakage Assumption* (ILA). However, in practice, masked implementations do not comply with such hypothesis. Indeed, several *physical effects*, such as memory transitions, glitches and coupling, *recombine* the shares, hence violating the ILA.

A typical class of models capturing such effects are the so-called *Transition-based* Leakage Functions (Def. 7). A well-known example is the *Hamming-Distance* leakage function (Eq. 4).

Definition 7. ***(Transition-based Leakage Function [5]).*** *Let* $\mathbb{V}$ *be a set of intermediate variables, and* $\mathbb{T} := \{v \oplus v' \mid \forall\, v, v' \in \mathbb{V}\} \cup \mathbb{V}$ *the set of all the transitions between these intermediate variables. A leakage function* $\mathrm{L}(\cdot)$ *is transition-based if its deterministic part* $\mathrm{L}(\cdot)_{\mathrm{d}}$ *takes values* $t \in \mathbb{T}$ *as argument.*

$$\mathrm{L}(X, Y) = \mathrm{HW}(X \oplus Y) + \mathcal{N}(0, \sigma) = \mathrm{HD}(X, Y) + \mathcal{N}(0, \sigma) \tag{4}$$

Balasch et al. proved that the security order of a d-probing secure implementation in the value-based leakage model is halved in the transition-based leakage model (i.e., $\lfloor \frac{d}{2} \rfloor$). The literature refers to this as the *security-order reduction theorem* [5]. Specific to the context of masked software implementations, a CPU micro-architecture exposes many elements violating the ILA, such as micro-architectural registers (e.g. the inter-stage pipeline registers or the *Memory Data Register* of the *Load-Store Unit* [2], [6]), the forwarding logic, the *Barrel-Shifter Unit*, the *Arithmetic-Logic Unit*, and the *Load-Store Unit* [2], [3], [21]. Gigerl et al. show that, in presence of glitch-based recombinations, the order reduction exceeds the reduction factor of 2 considered in presence of transition-based leakage [3]. Furthermore, the micro-architectural properties of complex CPUs also imply that the security reduction order is also greater than 2 [3].

The CMOS technology is still mainstream in digital design, and the overall power consumption of a CMOS-based circuit is the superposition of the power consumptions of its subelements [25]. We can describe the induced leakage via the *Sum-of-Hamming-Weights* leakage function:

$$\mathrm{L}(X,Y) = \mathrm{SHW}(X,Y) + \mathcal{N}(0,\sigma). \tag{5}$$

Such AGN model assumes as its deterministic component the SHW function:

$$\mathrm{SHW}(X,Y) = \mathrm{HW}(X) + \mathrm{HW}(Y). \tag{6}$$

In this paper we use the binary form of the SHW function, which can be readily extended to accept an arbitrary number of arguments.

D. PEARSON'S CORRELATION COEFFICIENT

The Pearson's Correlation Coefficient (PCorrl) is a statistical tool which quantifies the *linear* relationship between two random variables. Given two arbitrary random variable X, Y, the PCorrl is defined as:

$$\rho(X,Y) = \frac{\mathbb{E}[(X-\mu_X)\cdot(Y-\mu_Y)]}{\sigma_X \cdot \sigma_Y} \tag{7}$$

where μ and σ represent, respectively, the mean value and standard deviation of the given random variable. The coefficient takes values in the interval $[-1,+1]$, where the extremes indicate perfect linear dependency between the two variables, whereas a coefficient of 0 indicates no linear dependency.

E. TVLA

When performing security evaluations of a cryptographic implementation, the evaluator ideally aims to provide the most possible general answer regarding the security of the implementation (i.e., *is the implementation secure?*). The *Test Vector Leakage Assessment* (TVLA) [26] reduces the problem at testing whether two sets of side-channel traces $\mathbb{S}_{\text{fixed}}$ and $\mathbb{S}_{\text{random}}$ can be distinguished by their statistical moments (alternative hypothesis) or not (null hypothesis). $\mathbb{S}_{\text{random}}$ refers to side-channel traces collected while the implementation processes a different plaintext for each trace, whereas $\mathbb{S}_{\text{fixed}}$ refers to the usage of the same plaintext for each trace. In the case of univariate first-order TVLA, the evaluator computes the t-statistic t:

$$t = \frac{\hat{\mu}_{\text{fixed}} - \hat{\mu}_{\text{random}}}{\sqrt{\frac{\hat{\sigma}^2{}_{\text{fixed}}}{n_{\text{fixed}}} + \frac{\hat{\sigma}^2{}_{\text{random}}}{n_{\text{random}}}}} \tag{8}$$

where $\hat{\mu}_{\text{fixed}}, \hat{\mu}_{\text{random}}$ refer to the sample mean, $\hat{\sigma}^2{}_{\text{fixed}}, \hat{\sigma}^2{}_{\text{random}}$ to the sample variance and $n_{\text{fixed}}, n_{\text{random}}$ to the number of traces of the fixed and random set, respectively.

The implementation leaks information with a certain probability if the t-statistic overcomes a given *t-threshold*. The *t*-threshold is normally set to ± 4.5, which means we can reject the null hypothesis with a probability confidence of 99.999%.

At its core, TVLA relies on hypothesis testing. As such, it is affected by statistical errors too. We distinguish between Type-I errors (or false positives) and Type-II errors (or false negatives) [27]. Type-I errors refer to the cases where the test fails (null hypothesis rejected), although the implementation does not leak. Type-II errors, on the other hand, refer to the acceptance of the null hypothesis, although the implementation actually leaks. Type-II errors are the most troublesome, as they would report an implementation as leakage-free when it is not. As a mitigation technique against these types of errors, a strategy is to repeat the TVLA several times, each with a distinct fixed key [26].

F. MUTUAL INFORMATION

The Mutual Information (MI) is an information-theoretic tool for the quantification of *linear* and *non-linear* relationship between two random variables. The metric has different definitions, according to the nature (discrete or continuous) of the random variable. Equation 9 reports the definition of MI in the case of two discrete random variables X and Y.

$$\mathrm{MI}(X,Y) = \sum_{x\in\mathbb{X}}\sum_{y\in\mathbb{Y}} p(x,y)\cdot\log_2\frac{p(x,y)}{p(x)\cdot p(y)} \tag{9}$$

Although it can capture any type of relationship, the computation of the MI relies on the knowledge of the joint probability distribution $p(x,y)$. Generally, the distribution is unknown and can only be estimated. Therefore, MI cannot be directly computed, requiring the employment of estimators. Such estimators rely on different techniques such as histograms, Gaussian mixtures, k-nearest neighbours, or neural networks [28]–[30]. Among these estimators, the empirical *Hypothetical Information* (HI) provides an upper-bound to the MI [31], while converging towards MI as the number of traces increases. As such, HI fits in those contexts where a *conservative* analysis of the security of an implementation (i.e., overestimate the information leakage) is preferable.

G. BIASING LEAKAGE DISTRIBUTIONS (BLD) TO ATTACK MASKED PARALLEL IMPLEMENTATIONS

The strength behind masking stands in the need, for an attacker, to compute higher-order statistical moments and/or to perform multivariate statistical analyses. When considering hardware masked implementations, security evaluators assume a parallel computation model. Under this computation model, the implementation can treat related shares at the same time sample. Considering a nth-order masking scheme, the attacker, which observes all the $n+1$ shares of a key-dependent encoded value, needs, at least, the statistical moment of order $n+1$ to detect any key-dependent information. Moos and Moradi proposed a preprocessing technique to reduce such minimal key-dependent order moment [12]. Informally, the technique consists in selecting, for each trace sample, a subset of the measured traces, preserving only certain leakage values. Such *Biasing Leakage Distribution* (BLD) preprocessing biases the leakage distribution of each trace sample, converting higher-order leakages to lower-order leakages.

To exemplify this technique, let us consider a first-order BM encoding of $X \in \mathcal{X}_{2^2}$. Further, let us assume that the two shares $\mathbf{X}_0, \mathbf{X}_1$ are processed in parallel, and that the implementation leaks according to a noise-free SHW model

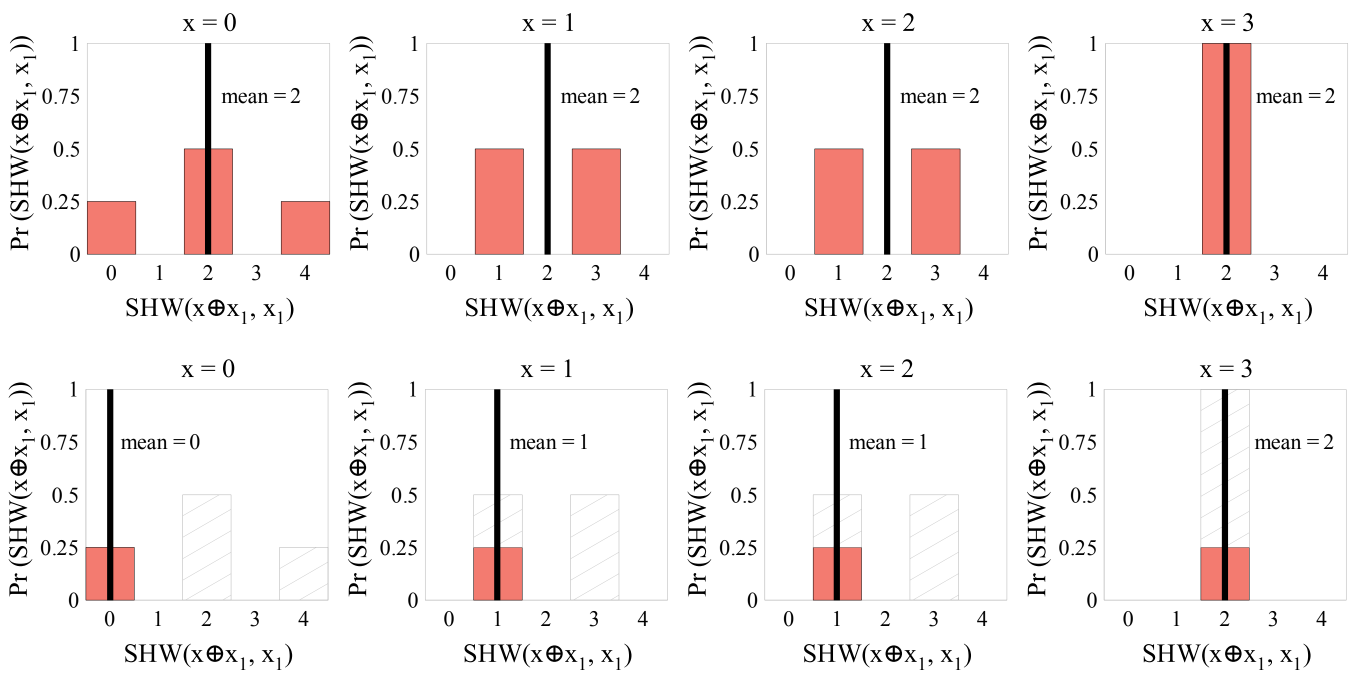


FIGURE 1: SHW distributions obtained for various secret values masked with first-order Boolean masking. x is the secret value, and x_1 a random value used for Boolean masking. Top row: distributions of SHW without preprocessing. Bottom row: distribution obtained when keeping only the lowest k% values ($k = 25\%$ here). While the mean is independent of the secret without preprocessing, it becomes dependent on the secret when only the lowest k% samples are kept.

(Eq. 5). Fig. 1, top row, reports the marginal distributions of each realisation of X. Each marginal distribution exhibits the same first-order moment (e.g., mean). That is, the first-order moment is independent on the encoded value X, as it is expected for a first-order masking scheme. Fig. 1, bottom row, reports the marginal distributions of the realisations of X after a preprocessing keeping the $k = 25\%$ of samples with the lowest values of the leakage distributions [12]. The preprocessed first-order moments of the marginal distributions depend on the secret value, making possible to mount first-order attacks. In practice, the resulting order reduction varies depending on the value of *threshold* k, and on the heuristic used for traces pruning (e.g., keeping the ones with the lowest leakage values) [12].

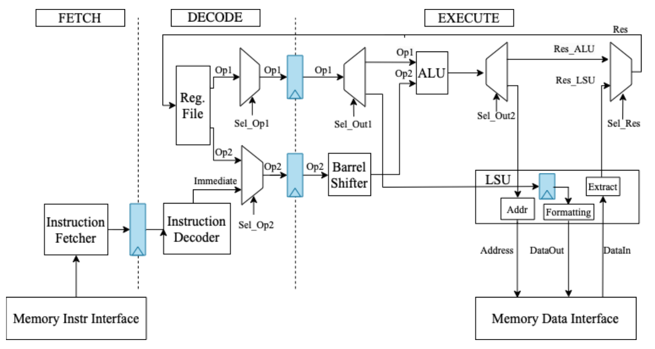


FIGURE 2: Simplified model of a 3-stage, in-order micro-architecture.

IV. PARALLEL PROCESSING OF SHARES IN SOFTWARE

As our goal is to evaluate the practical security of masked software implementations (Section V, Section VI, Section VII), we need first to assess the *potential* sources of leakage. To this end, we proceed as follows: we firstly provide a rationale explaining how the complexity of a CPU micro-architecture potentially induces PPS (Section IV-A). Then, we describe the three carefully hand-crafted assembler code (called micro-benchmarks, or *UBenches*) that we designed to investigate the presented rationale (Section IV-B). To confirm or reject the presence of PPS, we run side-channel analyses on each UBench (Section IV-D).

As presented in Section III-C, the micro-architecture of modern CPUs constitutes a rich source of recombination effects; in particular, of transition-based leakages. Hence, we also include a UBench exercising a transition-based leakage originating within the micro-architecture. We have released these micro-benchmarks (C and binary code) as publication artefacts (https://zenodo.org/record/8094516).

A. RATIONALE

The micro-architecture of modern CPUs extensively relies on hardware-oriented techniques to increase the instruction throughput [13]. Due to *instruction pipelining*, the micro-architecture is partitioned into several *stages*, where each stage takes care of a part of the instruction life cycle. Fig. 2 depicts a simplified 3-stage, in-order, micro-architecture. In such example, the *Instruction Fetch* (`IF`) stage fetches the next instruction to be executed, the *Instruction Decode* (DE) interprets the instruction (e.g., selecting operands from the Register File), whereas the *Instruction Execute* (EXE) executes the instruction. We remark that, in such example, the execution of memory-related instructions (e.g., load and store) requires 2 clock cycles, whereas arithmetic-logic instructions require 1. For memory accesses, the target address is sent to the memory in the first cycle of the EXE stage. During the second cycle, the data to be stored is sent to the memory, or the data to be read is received from the memory. The address computation phase employs the ALU. To avoid any resource conflict, during the address computation phase, the fetch and decode stages are stalled. Although being quite simple, such model captures the micro-architecture organisation of real micro-controller-graded CPUs (e.g., ARM Cortex-M3 and Cortex-M4 [7], [8]). With such model in mind, it gets easy to understand how PPS can happen in software. Indeed, as mentioned above, each stage takes care of one part of the instruction life cycle: the execution of the DE stage happens *in parallel* with the execution of the EXE stage. As a consequence, whenever the two stages of the simplified micro-architecture manipulate related shares, the micro-architecture processes shares *in parallel*.

B. MICRO-BENCHMARKS

We design three distinct micro-benchmarks, one for each potential PPS case we identified. Each UBench shares the same structure: a preamble followed by a workload (Listing 1). We implement the UBenches in Thumb-2 assembler, targeting ARM-based target platforms (Section IV-C).

The UBench preamble consists in a sequence of machine instructions preparing the architectural and micro-architectural states and the inputs for the workload. The preparation of the micro-architectural state consists in the randomization of the state of specific elements (e.g., micro-architectural registers, memory data-path), which may otherwise induce unintended leakage. The workload consists in a sequence of machine instructions, which attempts to exercise a desired leakage effect. The trigger_high() and trigger_low() functions, which surround the workload, respectively start and stop the collection of power-based side-channel traces. To clearly identify the workload-induced leakage effect, we pad the workload's beginning and ending with eor.w instructions provided with random inputs. To make clear the handling of these values, we comment each UBench instruction with its effect.

a: Notation

We denote the UBench target words as X0 and X1, whereas rndN refers to one of the UBench random input values. We denote with R_val a generic 32-bit register containing the value val. As a special case, we denote with R_destN a 32-bit register containing the result of the N-th UBench instruction. We refer to the immediate address of a value val with addr[val]. We denote a constant value const with #const.

Listing 1: Common Structure of Leakage Micro-Benchmarks

```
<ubench_template>:
  <preamble_ubench> ; Prepare UArch state
  <padding>         ; eor with random inputs x4
  bl <trigger_high> ; begin traces collection
  <padding>         ; eor with random inputs x8
  <workload_ubench> ; See Listing 2, 3, 4, 5
  <padding>         ; eor with random inputs x8
  bl <trigger_low>  ; end traces collection
```

b: PPS-related UBench #1

The first PPS-related UBench stimulates the parallel manipulation of bytes when loading a specific one from a given memory address. The preamble crafts a 32-bit word and stores it on the memory stack. Such word contains the least-significant byte (LSB) of both X0 and X1. The workload reads the X0's LSB by accessing to its address. The manipulation of the LSB of each share allows different word's layouts. Listing 2 reports the workload employed in our evaluations, hereafter referred to as UB-SHW-LDRB. We comment the workload with a byte-oriented representation of the word's layout (LSB on the right).

Listing 2: UB-SHW-LDRB workload

```
<workload_ubench_shw>:
  ; [R_addr] = [ LSB(X1) | 0 | LSB(X0) |  0      ]
  ; R_dst0  <- [ 0       | 0 | 0       | LSB(X0) ]
  ldrb.w R_dst0, [R_addr, #1]
```

c: PPS-related UBench #2

The second PPS-related UBench stimulates the parallel manipulation of values during the readings of X0 and X1 from the memory and the register file, respectively. Listing 3 reports the corresponding workload, hereafter referred to as UB-SHW-LDR-EOR. The ldr.w instruction enters the EXE stage at clock cycle #k. X0 enters the micro-architecture at clock cycle #k+1. Due to the pipeline stall inserted during the address generation, the eor.w instruction passes the DE stage at clock cycle #k+1. During the DE stage, the X1 is read from the register file. As a consequence, at clock cycle #k+1, the values X0 and X1 are simultaneously alive in the micro-architecture.

Listing 3: UB-SHW-LDR-EOR workload

```
<workload_ubench_shw_ldr_eor>:
  ldr.w R_dst0, addr[X0]      ; R_dst0 <- X0
  eor.w R_dst1, R_rnd0, R_X1  ; R_dst1 <- rnd0 ^ X1
```

d: PPS-related UBench #3

The third PPS-related UBench stimulates the parallel manipulation of values by processing X0 and X1, each handled by a distinct ALU instruction. Listing 4 reports the corresponding workload, hereafter referred to as UB-SHW-MOV-EOR. The mov.w instruction (and thus, its input operand X0) enters in the EXE stage at clock cycle #k. At the same clock cycle, the eor.w instruction enters the DE stage, where the target value

X1 is read from the register file. As a consequence, the values X0 and X1 will be both in the micro-architecture within the same clock cycle #k.

Listing 4: UB-SHW-MOV-EOR workload

```
<workload_ubench_shw_mov_eor>:
  mov.w R_dst0, R_X0      ; R_dst0 <- X0
  eor.w R_dst1, R_X1, #0 ; R_dst1 <- X1
```

e: *Transition-related UBench*

This UBench tests the transition-based leakage stemming from the update of the inter-stage pipeline registers. Listing 5 reports the corresponding workload, hereafter referred to as UB-HD. At clock cycle #k, the first eor.w instruction enters the DE stage. X0 is read from the register file and stored in the DE/EXE inter-stage register. At clock cycle #k+1 the second eor.w enters the DE stage. X1 is read from the register file, and it is stored in the DE/EXE register. The update of the DE/EXE potentially causes a transition-based leakage.

Listing 5: UB-HD workload

```
<workload_ubench_hd>:
  eor.w R_dst0, R_X0, R_rnd0 ; R_dst0 <- X0 ^ rnd0
  eor.w R_dst1, R_X1, R_rnd1 ; R_dst1 <- X1 ^ rnd1
```

C. EXPERIMENTAL SETUP

We execute the UBenches on the STM32F215 and STM32F303 micro-controllers. The former hosts a ARM Cortex-M3 CPU, whereas the latter a ARM Cortex-M4 CPU.

We compile each UBench with arm-none-eabi-gcc version 9.2.1. We tune the compilation with -Os, -mthumb, and -mcpu=cortex-m3 and -mcpu=cortex-m4 for the STM32F215 and the STM32F303, respectively. To minimise execution time variability across runs of the same code, we fetch code from the Flash, disable the instruction and data cache and set the Flash access latency to 0 clock cycles. We collect power-based side-channel traces via the ChipWhisperer setup, with an acquisition board CW-308 UFO and the CW-1200 oscilloscope [32]. We set the micro-controllers clock frequency to 7.384 MHz, and the oscilloscope samples the power consumption at a rate of 29.538 MHz. Hence, 4 samples per clock cycle are measured. The STM32F215 comes with an internal voltage regulator, which we left turned-on and set to 1.2V [33].

D. EVALUATION

For each UBench, we generate two datasets of randomly chosen input values: the *test* dataset and the *control* dataset. Then, for each input dataset, we collect a trace set of 30,000 power-consumption traces, each of 90 samples. Finally, for *both* the collected traces sets, we compute $\rho(\mathrm{L}(\mathtt{X0},\mathtt{X1})_{\mathrm{d}}, \mathbf{T}^{i}_{30\mathrm{k}\times 90})$, where $i \in [0, 90)$ and X0, X1 belong to the *test* dataset (i.e., the *control* input dataset is unused). With this procedure, we verify that any correlation stems from X0 and X1 manipulation, not from other experimental factors.

The two leftmost columns of Fig. 3 report the results under the SHW leakage model (Eq. 5), whereas the two rightmost columns report the results under the HD leakage model (Eq. 4). Except for the UB-SHW-LDRB on the STM32F215, we observe that, when using the proper leakage model (i.e., SHW and HD for PPS-oriented and transition-oriented UBenches, respectively) we observe a higher correlation in the *test* traces, confirming the presence of the targeted leakage effect. When looking for other effects (i.e., transitions in the PPS-oriented UBenches or PPS in transition-oriented UBenches), we do not observe any significant correlation, indicating that the searched effect is negligible. Concerning the UB-SHW-LDRB, as explained in Section IV-B, we test all the different word layouts. For the sake of brievity, we only report the results of UB-SHW-LDRB for the layout illustrated in Listing 2, but all the other word layouts give similar results.

Finally, we observe lower correlation values for the STM32F215 as compared to the STM32F303. Such difference, potentially stemming from micro-architectural differences and/or the noise generated by the STM32F215's internal regulator (Section IV-C), provides us two distinct noise settings for the same leakage model. We will take advantage of this difference to explore the practical resilience of BM, ASM and IPM in different noise settings.

In this section, we have experimentally shown that both transition-based and PPS-based leakages potentially occur in software. In the following section, we employ the developed UBenches to assess the security of masking encodings against transition-based and PPS-based leakages.

V. EVALUATION OF THE PRACTICAL RESILIENCE OF MASKING ENCODINGS

In the previous section, we verified the presence of both transition-based and PPS-based leakages on our two target micro-controllers. This section evaluates the practical resilience of first-order masking encodings against such leakage sources. We develop the evaluation in two settings: an *ideal* one (leakage model and leakage effect match); a *real* one (leakage model and leakage effect potentially differ). For the latter case, we rely on the UBenches designed to assess the presence of PPS and transition-based leakages (Section IV-B). We analyse the encodings' resilience in two steps: (a) quantification and comparison of the leaked information (Section V-A); (b) exploitation of the leaked information through first-order analyses (Section V-B).

A. THEORETICAL EVALUATION

As remarked in Section IV-D, the SHW and HD leakage models might not perfectly describe the actual behaviour of our target boards. In order to evaluate the leakage resilience in the case such models capture the leakage behaviour, we firstly conduct an information-theoretic analysis. For such purpose, we numerically estimate $\mathrm{MI}(X, \mathrm{L}(\mathbf{X}_0, \mathbf{X}_1))$, where $X \in \mathcal{X}_{2^4}$ and the shares $\mathbf{X}_0, \mathbf{X}_1 \in \mathcal{X}_{2^4}$ encode X according to BM, ASM or IPM. For IPM, we arbitrarily select $\mathbf{L} = (1, 6) \in \mathbb{F}^2_{2^4}$. We describe the leakage L via an AGN leakage model

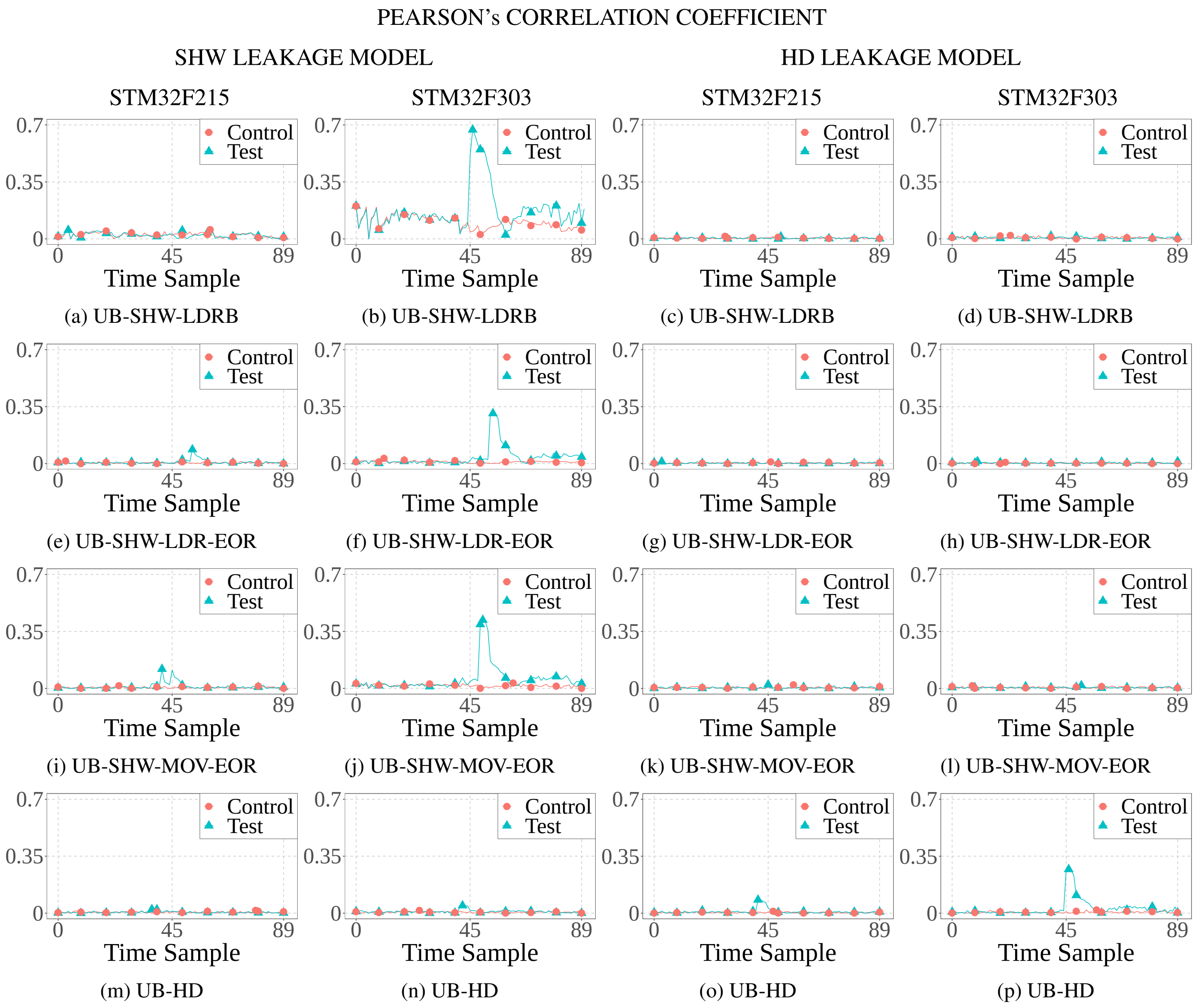


FIGURE 3: PCorrl-based evaluation of PPS-based and transition-based leakages. Each row reports the PCorrl from a different UBench: first row for UB-SHW-LDRB (Listing 2), second row for UB-SHW-LDR-EOR (Listing 3), third row for UB-SHW-MOV-EOR (Listing 4), fourth row for UB-HD (Listing 5). The two first columns report the results under the SHW leakage model, and the two last columns under the HD leakage model. The first and third column report the results for the STM32F215 board, whereas the second and fourth ones for the STM32F303 board. Each UBench is evaluated on two sets (*test* and *control*) of $30,000$ power-consumption traces.

(Eq. 1). According to the targeted leakage effect, we employ either $\mathrm{L}(\cdot)_\mathrm{d} = \mathrm{SHW}$ or $\mathrm{L}(\cdot)_\mathrm{d} = \mathrm{HD}$.

Fig. 4 reports the results of the information-theoretic leakage evaluation. We observe that the BM encoding leaks the most, while the IPM one leaks the least. Comparing the information leakage between the two leakage models, the SHW model not only provides the least information quantity, but it decreases faster. This is witnessed by the slope of the curves, as the SHW curve reports a slope of -2, whereas the HD one reports a slope of -1. As reported by Duc et al., such slope reports the minimal statistical moment to break the encoding [34].

We verify this observation by mounting a first-order correlation analysis on simulated power-consumption traces. Specifically, we generate $1,000,000$ traces, each of 1 sample, via an AGN leakage model, and we compute $\rho(\mathrm{HW}(X), \mathrm{L}(X_0, X_1))$, where $X, \mathbf{X}_0, \mathbf{X}_1 \in \mathcal{X}_{2^4}$. Fig. 5 reports the results of the first-order analyses. As expected, under the HD model, we detect correlation for both the BM and ASM encodings. Consistently with the information-theoretic analysis, we do not detect correlation for the IPM encoding. Concerning the SHW case, the first-order analysis does not identify correlation with the encoded value X. Such evidence illustrates the need of, at least, a second-order statistical

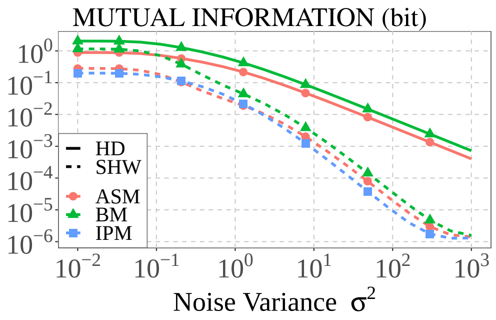


FIGURE 4: Information-Theoretic leakage resilience analyses results. The plot reports the numerically estimated $\mathrm{MI}(X, \mathrm{L}(\mathbf{X}_0, \mathbf{X}_1))$ evolution according to an increasing noise variance σ^2 (both in Log10 scale). We describe the leakage L as an AGN leakage model (Eq. 1), where $\mathrm{L}(\cdot)_\mathrm{d} = \mathrm{SHW}$ or $\mathrm{L}(\cdot)_\mathrm{d} = \mathrm{HD}$, for PPS-based and transition-based leakages, respectively. Due to estimation errors, for $\sigma^2 \geq 10^2$, the SHW curve diverges from the expected straight line. As IPM reaches perfect independence from X in the HD case, we omit the related curve.

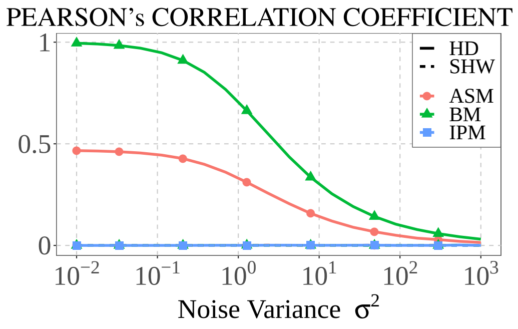


FIGURE 5: PCorrl-based leakage resilience analyses results on simulated traces. The plot reports $\rho(\mathrm{HW}(X), \mathrm{L}(\mathbf{X}_0, \mathbf{X}_1))$ according to an increasing noise variance σ^2 (Log_{10} scale), for the HD and SHW models. We generate $1,000,000$ power-consumption traces, each of 1 sample. We simulate the traces according to an AGN leakage model (Eq. 1), where $\mathrm{L}(\cdot)_\mathrm{d} = \mathrm{SHW}$ or $\mathrm{L}(\cdot)_\mathrm{d} = \mathrm{HD}$, for PPS-based and transition-based leakages, respectively. The metric does not detect correlation with X under the SHW for BM, ASM and IPM.

moment to correlate with X.

From the information-theoretic analyses, we observed that ASM and IPM encodings tend to better mitigate transition-based and PPS-based leakages. We corroborated such analyses by first-order moment analyses, evaluating the correlation between the encoded value and the simulated power-consumption. We observed results consistent with the information-theoretic ones. Furthermore, we highlighted how first-order moments cannot detect any information in the presence of PPS-based leakage.

Having obtained an overview of the resilience of first-order BM, ASM and IPM in an ideal setting (i.e., the leakage models perfectly describes the target leakage effect), in the next section we evaluate the resilience of such encodings in a more realistic context.

B. EXPERIMENTAL EVALUATION

In the previous sub-section, we analysed the information-theoretic resilience of first-order BM, ASM and IPM. We completed the analyses with a PCorrl-based evaluation on simulated traces. Such evaluation remarked the better leakage resilience of ASM and IPM encodings. Although the interest provided by an ideal setting (i.e., simulated traces), masked software implementations are executed in an imperfect one, where the leakage behaviour potentially deviates from the hypothetical one. As such, in this section we evaluate the leakage resilience of the three masking schemes when the first-order encodings are manipulated on our two boards, the STM32F215 and STM32F303. For this purpose, we re-use the UBenches of Section IV-B, which stimulate PPS-based and transition-based leakages. Differently from the information-theoretic analyses, for IPM we arbitrarily select $\mathbf{L} = (1, 170) \in \mathbb{F}_{2^8}^2$.

For each UBench, we capture $4,000,000$ traces, each of 90 samples. We first quantify the leaked information by computing $\mathrm{HI}(X, \mathbf{T}^i_{4\mathrm{M}\times 90})$. We set the random target inputs `X0`, `X1`, manipulated by each UBench, to the realisation of the shares $\mathbf{X}_0$, $\mathbf{X}_1 \in \mathcal{X}_{2^8}$, in each of the studied masking encodings BM, ASM, and IPM. As explained in Section III-F, the HI provides an upper bound of MI. This property is of particular interest in our case as we want to assess conservatively the amount of leakage. HI also converges towards the true MI as the number of traces gets higher [31].

The first two columns of Fig. 6 present the results of the HI analysis for the considered masking encodings and UBench. We compute the HI via the `ENNEMI` Python library [35] which implements a k-nearest-neighbour algorithm. Although the high number of traces and the univariate setting which favours HI convergence, we observe weak information leakage on the STM32F215 for both UB-SHW-LDR-EOR and UB-SHW-LDRB. As shown in Fig. 3, PPS leakage seems very low on this board, which may explain this result. On the STM32F303, UB-SHW-LDR-EOR and UB-SHW-LDRB show a tiny peak of information, whose significance is uncertain. By contrast, peaks of information are clearly visible for UB-SHW-MOV-EOR and the UB-HD on both boards. As expected, the BM encoding leaks the most information, while leakage is hardly visible for the IPM for the given number of traces.

For completeness, we also run first-order moment analyses on the same traces sets. Specifically, we compute $\rho(\mathrm{HW}(X), \mathbf{T}^i_{4\mathrm{M}\times 90})$ where $i \in [0, 90)$. The last two columns of Fig. 6 report the results.

Unexpectedly, we observe a correlation peak for the UB-SHW-MOV-EOR. As explained in Section V-A, a first-order moment cannot detect correlation with an encoded value via PPS-based leakage. Still, the peak takes place at the same time sample where we verified the presence of PPS-based leakage (Section 3). Hence, we ascribe the observed correlation to a

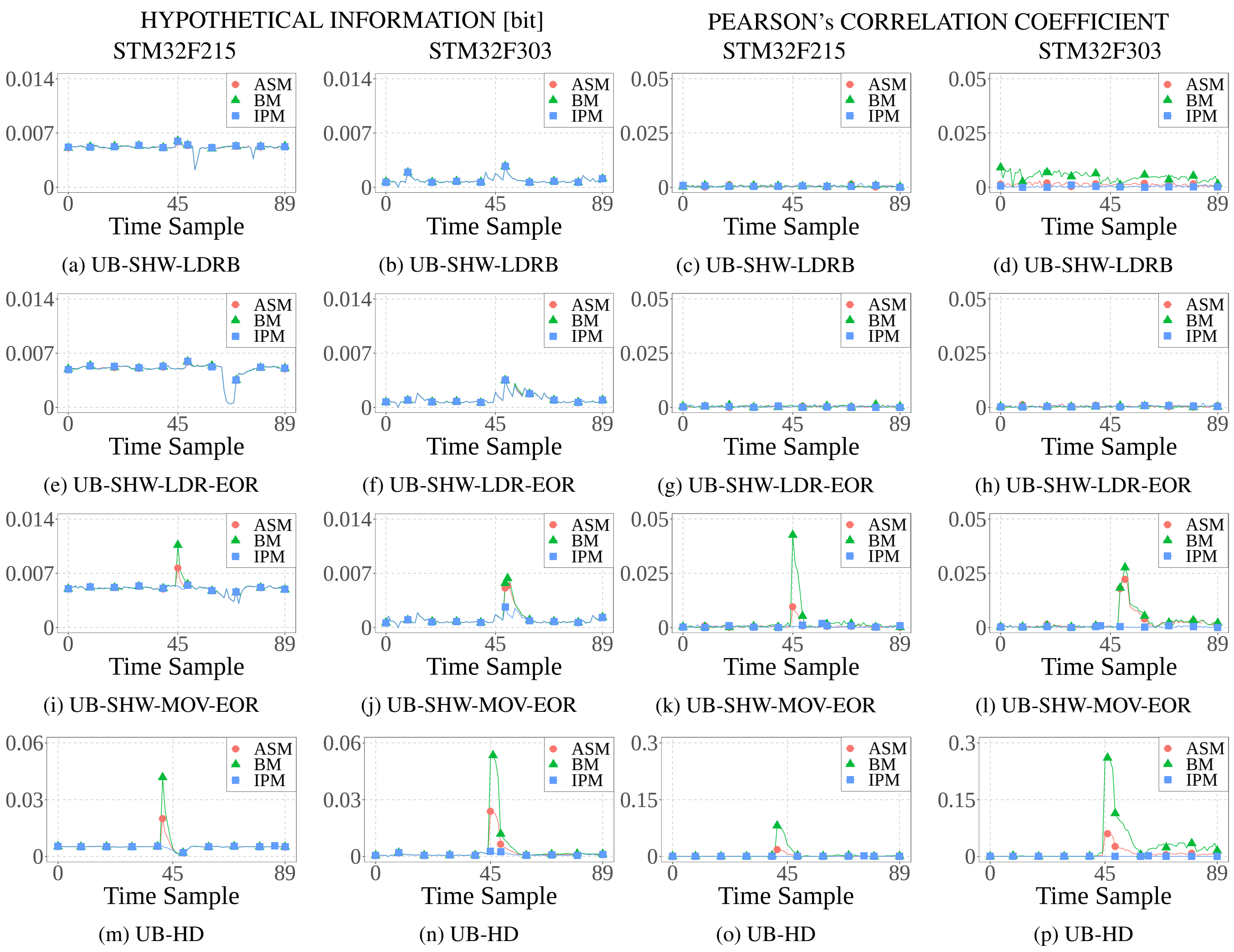


FIGURE 6: Experiment-based quantification of the transition-based and PPS-based leakages. Each row reports the PCorrl from a different UBench: first row for UB-SHW-LDRB (Listing 2), second row for UB-SHW-LDR-EOR (Listing 3), third row for UB-SHW-MOV-EOR (Listing 4), fourth row for UB-HD (Listing 5). The first two columns report the HI metric, whereas the last two report the PCorrl metric. The first and third column reports the results for the STM32F215 board, whereas the second and fourth one for the STM32F303 board. For each UBench and board, we compute the PCorrl on a $4,000,000$ power-consumption trace set.

recombination effect that occurs *simultaneously* with the PPS event.

Up to now, we evaluated the leakage resilience of different masking encodings against transition-based and PPS-based leakages. Concerning transition-based leakages, the results highlight the better leakage resilience of ASM and IPM encodings. Concerning the PPS-based ones, although the use of $4,000,000$ traces, the HI-based analyses hardly identify any PPS-based information leakage. Nonetheless, a different approach e.g., use of the BLD preprocessing [12], could better take advantage of the existing information leakage.

With this last remark, we employ the BLD preprocessing proposed by Moos and Moradi [12]. Their approach takes advantage of the PPS, converting higher-order leakages into lower-order ones, reducing the security order of the encoding (Section III-G). We directly focus on experimental analyses, as simulation-based ones are extensively provided in the original work [12]. Due to its high correlation with the PPS-based leakage (Fig. 3), we limit our analysis to the trace set collected with the UB-SHW-LDRB execution on the STM32F303. From experimental attempts, we identified $k = 10\%$ (i.e., $400,000$ traces per sample) as a good threshold. Fig. 7 provides the correlation curves from the BLD-based analyses. This time, we detect correlation peaks for both BM and ASM encodings, confirming the potential exploitability of PPS-based leakage.

This section has shown that transition-based and PPS-based leakages represent a concrete vulnerability in software masking implementations, leaking exploitable information through simple first-order analyses. Among the selected can-

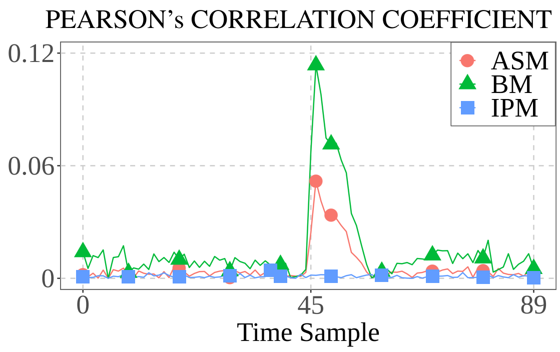


FIGURE 7: Evaluation of the BLD approach (Section III-G). We collect $4,000,000$ power-consumption traces and apply the BLD approach for $k = 10$. We compute the PCorrl by means of the HW model. We collect the traces during the execution of the UB-SHW-LDRB (Listing 2) on the STM32F303 board.

didates, the IPM was found to be the least vulnerable, preventing even the exploitation of higher-order leakages by means of the BLD approach. Yet, such approach relies on the HW model's distribution of the encoded value X, independently of the targeted masking scheme and leakage source. In reality, the distribution of HD and SHW model changes with the masking encoding. In the following section, we take advantage of this observation to break all the evaluated software masked implementations of AES with first-order analysis.

VI. EXPLOITATION OF LEAKAGE MODEL DISTRIBUTION IN IMPROVED CORRELATION ATTACKS

In the previous section, we have evaluated the resilience of BM, ASM and IPM first-order encodings, remarking the better leakage resilience of ASM and IPM ones. This result stems from the consistent employment of the HW to model the leakage of the encoded variable X. In general, such model provides low discrimination capabilities when targeting recombination effects as transitions. For instance, given a first-order IPM encoding of an arbitrary X, $\mathrm{HD}(\mathbf{X}_0, \mathbf{X}_1) \neq \mathrm{HW}(X)$. The same observation holds for PPS-based leakages. In this section, we take advantage of the above remark to enhance the practical security investigation of masking encodings. We proceed as follow: we first elaborate on the unsuitability of the HW model when targeting transition-based and PPS-based leakages; we discuss how to exploit the leakage model's distribution to build more efficient ones. Then, we put in practice the developed models, mounting first-order analyses and compare the new security results with the previous ones.

A. RATIONALE

When targeting leakages involving multiple shares, generally the HW model provides low discrimination capabilities. Considering the case of transitions and PPS-based leakages, the HD and SHW distributions are different from the HW's (Eq. 10, Eq. 11).

$$\mathcal{D}_{(\mathrm{HD}(\mathbf{X}_0,\mathbf{X}_1),X)} \neq \mathcal{D}_{(\mathrm{HW}(X),X)} \tag{10}$$

$$\mathcal{D}_{(\mathrm{SHW}(\mathbf{X}_0,\mathbf{X}_1),X)} \neq \mathcal{D}_{(\mathrm{HW}(X),X)} \tag{11}$$

Fig. 8 reports $\mathcal{D}_{(\mathrm{HD}(\mathbf{X}_0,\mathbf{X}_1),X)}$ and $\mathcal{D}_{(\mathrm{SHW}(\mathbf{X}_0,\mathbf{X}_1),X)}$ for BM, ASM and IPM. As the distributions differ, so the marginal distributions do. It is possible to exploit such difference to define (statistical-)moment-based leakage models.

For instance, we can associate to each X's realisation the first-order moment of the marginal $\mathcal{D}_{(\mathrm{HD}(\mathbf{X}_0,\mathbf{X}_1),X=x)}$:

$$\mathrm{HD}_{\mathrm{fo}}(x) = \frac{1}{|\mathbb{F}_{2^8}|^2} \sum_{\mathbf{x}_i \in \mathbb{F}_{2^8}, x = \bigodot_i x_i} \mathrm{HD}(\mathbf{x}_0, \mathbf{x}_1) \tag{12}$$

Nevertheless, such moment-based approach cannot improve the PCorrl results in the IPM case, as the $\mathcal{D}_{(\mathrm{HD}(\mathbf{X}_0,\mathbf{X}_1),X)}$ is independent of X (and thus, any statistical moment is independent of X).

Concerning the SHW model, which we use to model the PPS-based leakages, the $\mathcal{D}_{(\mathrm{SHW}(\mathbf{X}_0,\mathbf{X}_1),X)}$'s first-order moment is independent of X, for all the three masking schemes. Thus, we can not straightforwardly employ $\mathrm{SHW}_{\mathrm{fo}}$ (Eq. 13) model.

$$\mathrm{SHW}_{\mathrm{fo}}(x) = \frac{1}{|\mathbb{F}_{2^8}|^2} \sum_{\mathbf{x}_i \in \mathbb{F}_{2^8}, x = \bigodot_i x_i} \mathrm{SHW}(\mathbf{x}_0, \mathbf{x}_1) \tag{13}$$

Yet, we can resort on the BLD preprocessing to make $\mathcal{D}_{(\mathrm{HD}(\mathbf{X}_0,\mathbf{X}_1),X)}$'s mean secret-dependent. We define the biased version of the $\mathrm{SHW}_{\mathrm{fo}}$ model:

$$\mathrm{SHW}_{\mathrm{fo},k\%}(x) = \frac{1}{|\mathbb{F}_{2^8}|^2} \sum_{\mathbf{x}_i \in \mathbb{F}_{2^8}, x = \bigodot_i x_i} \mathrm{SHW}_{k\%}(\mathbf{x}_0, \mathbf{x}_1) \tag{14}$$

where

$$\mathrm{SHW}_{k\%}(\mathbf{x}_0, \mathbf{x}_1) = \begin{cases} \mathrm{SHW}(\mathbf{x}_0, \mathbf{x}_1), \text{if } \mathrm{SHW}(\mathbf{x}_0, \mathbf{x}_1) \in \mathbb{O}_{k\%}(x) \\ 0, \textit{otherwise} \end{cases}$$

and $\mathbb{O}_{k\%}(x)$ contains the $k\%$ lowest (or highest) realisation of $\mathrm{SHW}(\mathbf{X}_0, \mathbf{X}_1)$ when $X = x$.

B. EVALUATION

We start with a first evaluation of the $\mathrm{HD}_{\mathrm{fo}}$ leakage model for transition-based leakages. We target the ASM scheme, as for BM we cannot improve the results, and the IPM is intrinsically immune to this leakage type. We compute $\rho(\mathrm{HD}_{\mathrm{fo}}(X), \mathbf{T}^i_{4\mathrm{M}\times 90})$, with $\mathbf{T}_{4\mathrm{M}\times 90}$ the trace set collected from the STM32F303 board executing UB-HD. Fig. 9 confirms the better suitability of the first-order moment leakage model, as we get a higher PCorrl value with respect to the HW model. Then, we test the improvements concerning the exploitation of PPS-based leakages. We employ the $\mathrm{SHW}_{\mathrm{fo},k\%}$ model against each masking scheme, computing $\rho(\mathrm{SHW}_{\mathrm{fo},k\%}(X), \mathbf{T}^i_{4\mathrm{M}\times 90})$, with $\mathbf{T}_{4\mathrm{M}\times 90}$ the trace set collected from the STM32F303 board executing UB-SHW-LDRB. We experimentally select $k = 10\%$ (i.e., $400,000$

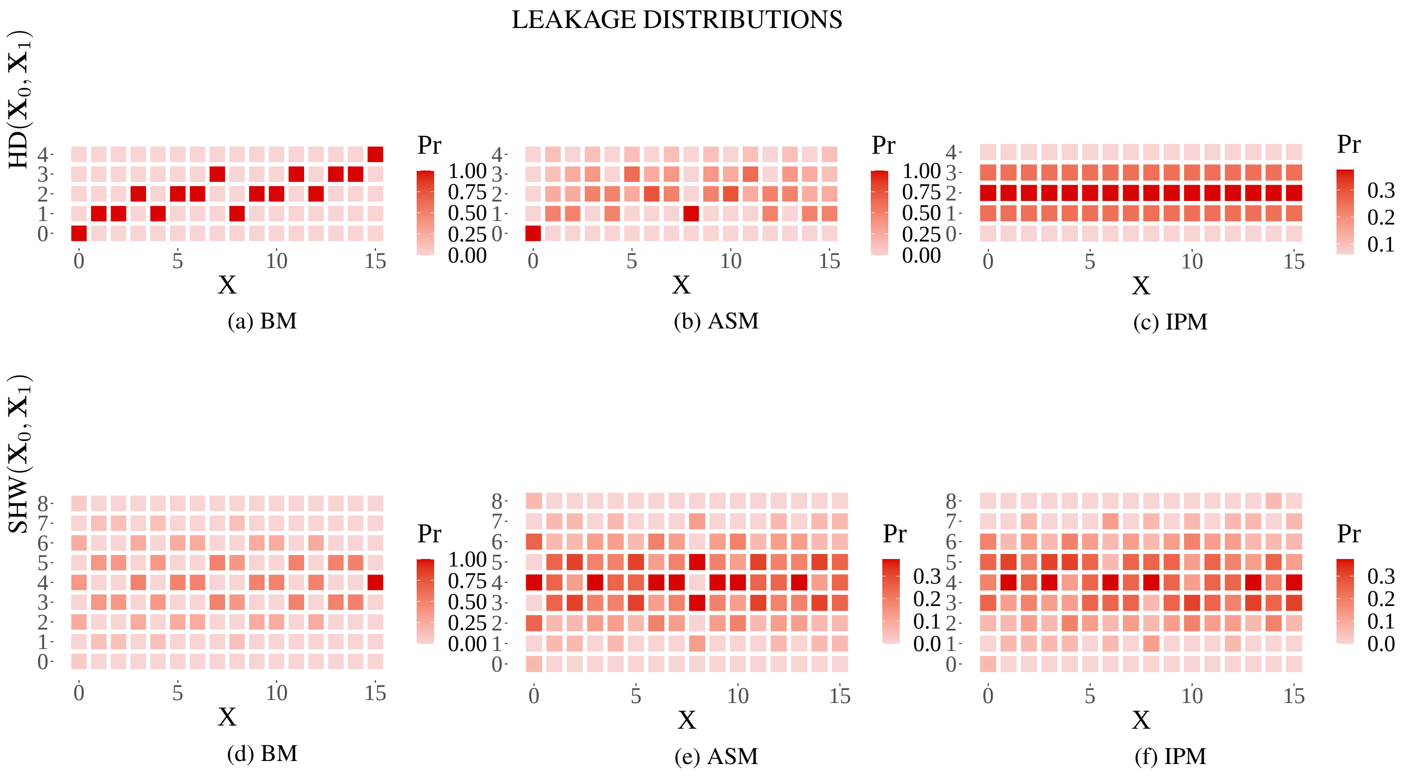


FIGURE 8: Distribution of the HD and SHW leakage models. Given $X \in \mathcal{X}_{2^4}$, the first row reports $\mathcal{D}_{(\mathrm{HD}(\mathbf{X}_0,\mathbf{X}_1),X)}$, whereas the second one reports $\mathcal{D}_{(\mathrm{SHW}(\mathbf{X}_0,\mathbf{X}_1),X)}$, where $\mathbf{X}_0, \mathbf{X}_1 \in \mathcal{X}_{2^4}$ represent the shares obtained from the application of BM (first column), ASM (second column) or IPM (third column) to X.

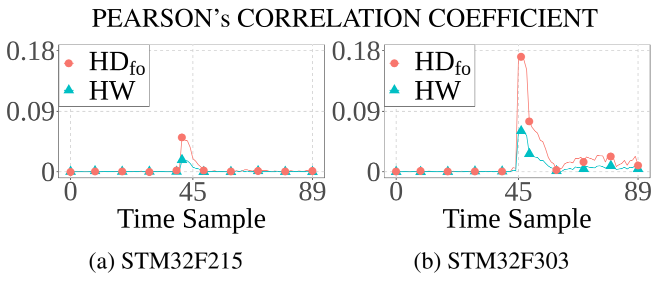


FIGURE 9: Experiment-based comparison of the $\mathrm{HD}_{\mathrm{fo}}$ and the HW leakage models. We consider the ASM case. We compute PCorrl on $4,000,000$ power-consumption traces. We collect traces during the execution of the UB-HD (Listing 5) on the STM32F215 and the STM32F303 boards.

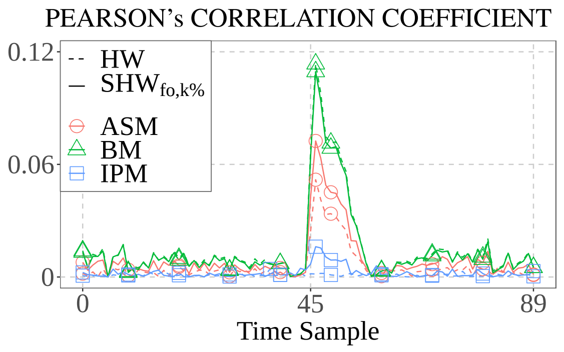


FIGURE 10: Experiment-based comparison of the HW and the $\mathrm{SHW}_{\mathrm{fo},k\%}$ model. We consider the case of the BLD-based PCorrl analyses, for $k = 10$ (Section III-G). For the $\mathrm{SHW}_{\mathrm{fo},k\%}$ model, we set $k = 10$. We compute PCorrl over $4,000,000$ power-consumption traces. We collect the traces during the execution of the UB-SHW-LDRB (Listing 2) on the STM32F303 board.

traces per each $0 \leq i < 90$ sample) as it works well for BM, ASM and IPM. Fig. 10 compares the PCorrl when employing the HW model and our moment-based leakage model. The HW allows the detection of correlation peaks in the case of BM and ASM schemes, but none in the IPM case. In contrast, our moment-based model not only improves the correlation results for the ASM, but it detects a correlation peak in the IPM case.

Such results corroborate the observations made in Section V, remarking the better leakage resilience of ASM and IPM encodings against transition-based and PPS-based leakages.

VII. SIDE-CHANNEL RESILIENCE OF SOFTWARE MASKED AES-128

With Section V and Section VI we assessed the practical security of different first-order masking encodings. Such analyses are fundamental get insights on the achievable security of masked implementations. Inner-product encoding showed perfect resistance against transition-based leakage, while Boolean and arithmetic encodings were more vulnerable. All masking encodings showed vulnerability to PPS-based leakage. We question how these findings translate on a full implementation.

This section aims at evaluating the impact of transition-based and PPS-based leakages on 4 software implementations of the AES-128 block-cipher: an unprotected version (*vanilla* from now on) and three masked ones, one for each masking scheme investigated. We have released all our investigated implementations (both C and binary codes) as publication artefacts (https://zenodo.org/record/8094516).

Our security assessment splits in two phases: at first, we evaluate whether the masked implementations leak information; the second one assess the resistance of such implementation against the exploitation of the (potential) leakage. The first phase relies on the TVLA methodology (Section III-E) to provide an assessment independent on the class of attacker. The second phase relies on the same techniques employed to analyse the masking encodings (Section V, Section VI). Specifically, we evaluate the security with and without the BLD technique (Section III-G). We start by exploiting univariate first-order moment leakages, then we exploit univariate higher-order moment leakages with filtering. This last phase is particularly important to assess the practical security against PPS, since its first-order moment leakages can't be directly exploited (Section V-A).

A. EXPERIMENTAL SETUP

The vanilla implementation follows the FIPS-PUB-197 specification [17], except for the key-scheduling: the implementation generates the next round key between the SubByte and the MixColumns steps.

Each first-order masked implementation follows by the manual application of the related masking scheme to the vanilla implementation. In particular, the BM and IPM ones follow the specification of Rivain et al. [36] and Balasch et al. [15], respectively. For the IPM version, we resort to $\mathbf{L} = (1, 170) \in \mathbb{F}_{2^8}^2$, the same we employed for the experiment-based analyses (Section V, Section VI). We implement the finite field multiplication using log/exp tables [37].

Concerning the ASM implementation, an inherent difficulty is the masking of the field addition (i.e., the eXclusive-OR, XOR). Indeed, the XOR is *non-linear* with respect to the arithmetic-sum operation. We mask the XOR operation by means of a masked look-up table. A straightforward tabulation of the operation would require 2^{16} byte of memory. To reduce the memory consumption, we tabulate the XOR on 4 bits, where the concatenation of the least (and most) significant inputs' nibbles indexes the table. We compute the XOR between two 8-bit inputs as a double access to such table: one to process the least significant nibbles of the inputs, and one to process the most significant ones. We remark that, the output carry of the arithmetic-sum potentially leaks information on the processed values. To prevent such leakage, we pre-charge the landing bit of the output-carry with a fresh random value.

In the vanilla implementation, for performance reasons, we tabulate the SBOX and the XTIME functions. In the ASM implementation, we implement the same functions by means of masked look-up tables. Concerning the BM and IPM implementations, we compute those functions on the fly.

We resort to the experimental setup introduced in Section IV-C (software toolchain and side-channel measurement setup). We develop each implementation in C language, and compile them with the compiler toolchain and compilation options reported in Section IV-C. Table 3 reports the mean execution time, number of PRNG calls, and memory impact of each AES-128 implementation. We report such parameters for both STM32F215 and STM32F303. Each masked implementation draws fresh randomness from the xoroshiro64** 1.0 PRNG [38]. The execution time from Table 3 includes time spent in the PRNG. We remark the long execution time ($500, 000$ clock cycles on the STM32F215) for the ASM implementation. We ascribe it to the MixColumns step, which performs several accesses to the table-based XOR implementation. We remark that our experimental setup provides us with correctly-aligned side-channel traces. Hence, we do not require any re-alignment of the side-channel traces.

For the purpose of our analyses (e.g., leakage resilience against physical effects), we have to guarantee the correct application of the masking scheme. Each of the selected scheme considers a *value-based* leakage model. Thus, we verify that no value-based leakage can be detected from each implementation. To this end, we run TVLA analyses on simulated power-traces collected during the execution of each implementation on a ISA-level simulator of the ARMv7 profile. Specifically, we simulate the power consumption stemming from the usage of the register file and memory requests via load and store instructions. For all the implementations, we accept the null hypothesis (i.e., the implementation does not leak in the value-based model), proving the correct application of the three considered masking schemes.

B. INFORMATION LEAKAGE EVALUATION

As a first step in the leakage resilience assessment of our AES-128 implementations, we proceed with the TVLA methodology. Precisely, we analyse the full first round of each implementation, except for the ASM implementation: as pointed out in Section VII-A, the MixColumns step counts for the largest part of the execution time. To reduce the trace collection time without compromising the validity of our results, we exclude the ASM's MixColumns from the leakage evaluation. As introduced in Section III-E, the TVLA allows an evaluator to determine whether an implementation leaks or not, independently on the particular attack or leakage model.

TABLE 3: Mean execution time (in clock cycles), number of calls to the PRNG, and segment size (in bytes) of each AES-128 implementation

Version	Execution time		PRNG	Segment size		
	STM32F303	STM32F215	calls	`.text`	`.data`	`.bss`
Vanilla	3,524	4,180	0	1,016	525	0
BM	70,310	73,478	35	4,384	276	324
ASM	469,615	498,805	13,463	14,936	280	840
IPM	202,318	213,580	3,234	12,836	8,756	152

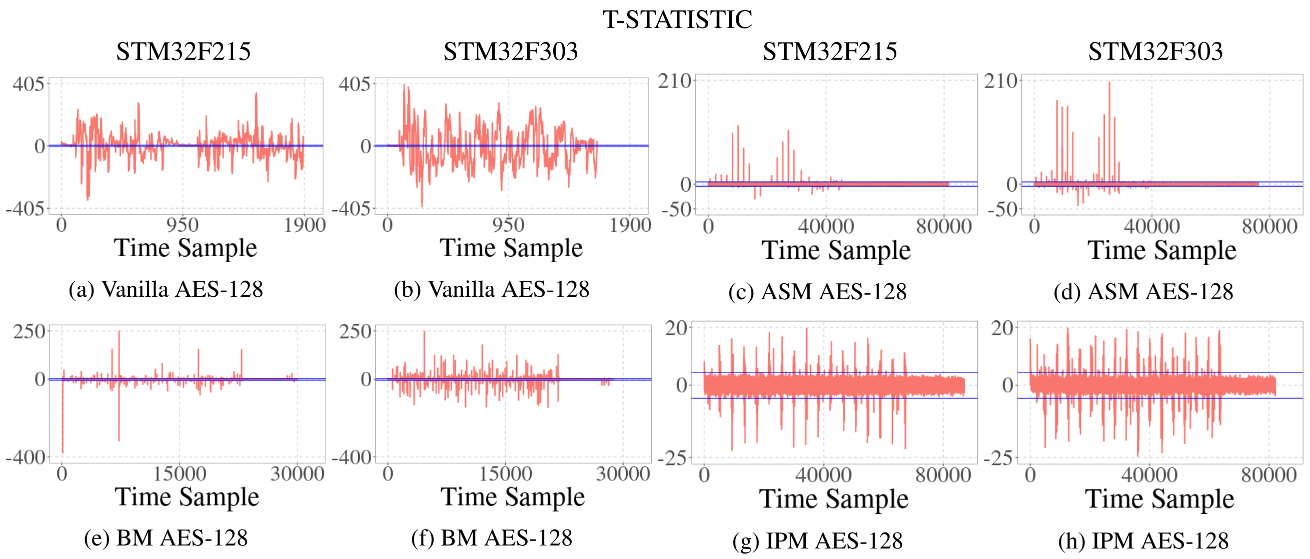


FIGURE 11: TVLA results on the 4 AES-128 implementations. In red, we report the *maximum* t-statistic between two t-tests. In blue, the t-statistic threshold (± 4.5) for the null hypothesis rejection. We execute each t-test by using a distinct fixed key. The first and third columns refer to the STM32F215 board, whereas the second and fourth ones to the STM32F303 board. Each plot refers to a $15,000$-vs-$15,000$ t-test, except for the IPM AES-128, which refers to a $90,000$-vs-$90,000$ t-test.

For the vanilla, BM and ASM implementations, we collect $15,000$ power-consumption traces for both fixed and random sets, respectively. Concerning the IPM implementation, we observed that it is characterised by a higher leakage resilience (Section V, Section VI). To be more confident in its evaluation, perform the same assessment with $90,000$ power-based traces for both the fixed and random trace set. As explained in Section III-E, the TVLA methodology is prone to errors of type I and II, where the latter represents the most problematic ones. To cope with them, for each implementation, we repeat the TVLA assessment two times, each with a distinct fixed key, and we measure the maximum absolute t-statistic for each sample point of the traces. Fig. 11 reports the TVLA results for each AES-128 implementation and each target board.

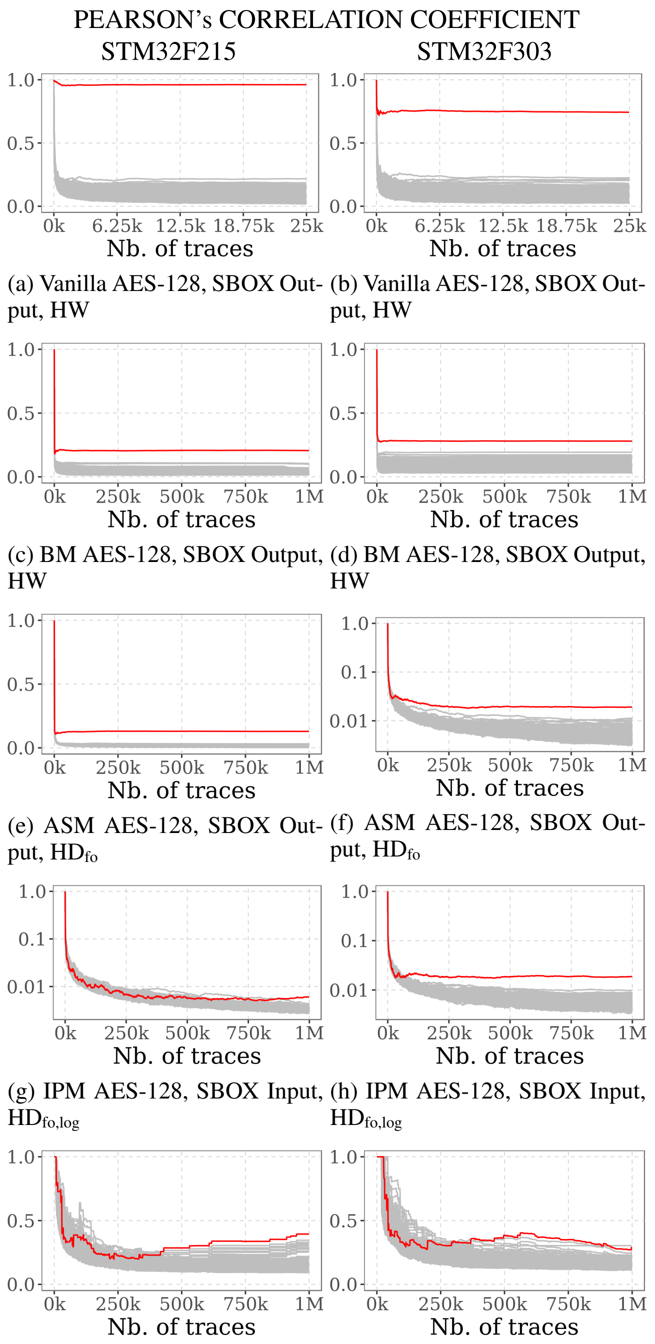
The vanilla, BM and ASM implementations leak information along the whole first round. As we verified that the masking countermeasure is correctly applied at binary level, and as first-order statistical moments cannot detect leakage from PPS, we ascribe such leakage to recombination effects (e.g., transitions).

We remark that the ASM implementation presents fewer leaking samples than the BM. The algebraic structure of the ASM encoding potentially contribute to such observation.

Unexpectedly, the leakage assessment on the IPM implementations reveal several leakage points along the full first round. We found out that the source of such leakages stem from recombination effects that impact the log/exp-based field multiplication. Specifically, we verified the statistical dependence between $\mathrm{HD}(\log_3(\mathbf{X}_0), \log_3(\mathbf{X}_1))$ and the encoded value x. We conjecture that the non-linear nature of the logarithm function introduces some *bit-interaction* effect between the share's bits. Such effect counteracts the *randomness diffusion* of the IPM, making transition-based leakage again exploitable. Yet, we remark that, despite the higher number of employed traces, we observe a way lower magnitude of the t-statistic with respect to the one of the other implementations.

C. INFORMATION LEAKAGE EXPLOITATION

In the previous section, we assessed the leakage resilience of our AES-128 implementations. We observed results consistent to the encoding analyses (Section V, Section VI), except for the IPM. In fact, we observed unexpected leakage stemming from the finite field multiplication. Despite the presence of leakage, the TVLA methodology does not provide any clue concerning the *exploitability* of the leaked information.



(a) Vanilla AES-128, SBOX Output, HW

(b) Vanilla AES-128, SBOX Output, HW

(c) BM AES-128, SBOX Output, HW

(d) BM AES-128, SBOX Output, HW

(e) ASM AES-128, SBOX Output, $\mathrm{HD_{fo}}$

(f) ASM AES-128, SBOX Output, $\mathrm{HD_{fo}}$

(g) IPM AES-128, SBOX Input, $\mathrm{HD_{fo,log}}$

(h) IPM AES-128, SBOX Input, $\mathrm{HD_{fo,log}}$

(i) IPM AES-128, SBOX Output, $\mathrm{SHW}_{\mathrm{fo},k\%}$, $k = 0.5\%$

(j) IPM AES-128, SBOX Output, $\mathrm{SHW}_{\mathrm{fo},k\%}$, $k = 0.5\%$

FIGURE 12: CPA results for the four AES-128 implementations. In grey, the wrong key hypotheses, whereas in red the correct one. Fig. 12f, 12g and12h report the PCorrl in Log10 scale. For each implementation, we employ a different leakage model (Table 4). For the $\mathrm{SHW}_{\mathrm{fo},k\%}$ model, the X-axis reports the number of collected traces (i.e., before trace filtering). Each row refers to a different implementation/leakage model combination. First and second columns refer, respectively, to the STM32F215 and STM32F303 board.

TABLE 4: Summary of the leakage models used for the side-channel analysis of each AES-128 implementation.

Implementation(s)	Leakage Model: Transitions	Leakage Model: PPS
Vanilla, BM	HW (Eq. 3)	-
ASM	HW, $\mathrm{HD_{fo}}$ (Eq. 12)	-
IPM	HW, $\mathrm{HD_{fo,log}}$ (Eq. 15)	$\mathrm{SHW}_{\mathrm{fo},k\%}$ (Eq. 14)

With this section, we explore the resilience of our software masked implementations against information leakage exploitation; specifically, against univariate side-channel attacks.

To this end, we rely on standard, BLD-based (Section III-G) and moment-based-model (Section VI) CPA attacks. For each implementation and target board, we measure $1,000,000$ power traces.

The side-channel analysis proceeds as follows. We analyse the usage of the first secret key byte during the SubByte step of the first round, and we compute $\rho(\mathrm{L}(X)_{\mathrm{d}}, \mathbf{T}^{j}_{1\mathrm{M}\times m})$, where m varies according to the target implementation. Table 4 summarises the leakage models $\mathrm{L}(\cdot)_{\mathrm{d}}$ employed to attack each implementation.

For the IPM implementations, we also target the SubByte's input, which comes as result of the field implementation. We employ the first-order-moment leakage model $\mathrm{HD_{fo,log}}$:

$$\mathrm{HD_{fo,log}}(x) = \frac{1}{|\mathbb{F}_{2^8}|^2} \sum_{\mathbf{x}_i \in \mathbb{F}_{2^8}} \mathrm{HD}(\log_3(\mathbf{x}_0), \log_3(\mathbf{x}_1)) \quad (15)$$

Fig. 12 reports the results of the different CPA attacks, and Table 5 reports the minimum number of traces required to mount a successful CPA attack. Despite the correct application of the masking scheme on the binaries, we exploit only 140 and $241,000$ traces to break the BM and ASM implementations, respectively. Consistently with the result from Section VI, the $\mathrm{HD_{fo}}$ model improves the attack efficiency against the ASM implementation, reducing up to $\times 8.6$ times the minimum number of traces to mount a successful CPA attack, with respect to a plain use of the HW model.

By targeting the SBOX input, we successfully retrieve the target key byte on IPM implementations. This suggests that the design of masking schemes should also consider the implementation of the employed algorithms (e.g., finite field multiplication). We remark that the attack on the STM32F215 takes longer to succeed. This may be due to the lower accuracy of the HD model for this device and/or the higher noise affecting the platform (Section IV-D).

We conclude the leakage exploitation analyses with the BLD-based CPA attacks (Section III-G). We evaluate the resilience of each implementation according to several k values. Table 6 reports the rank of the correct key hypothesis with $1,000,000$ traces, and the minimum traces number to reach that rank. On the STM32F303, the correct key hypothesis frequently appears among the best correlated key candidates. Table 6 reports the number of traces necessary to observe the correct key byte hypothesis among the 4 best correlated key candidates. Then, an attacker can brute-force the 4^{16} possible 128-bit keys.

We remark that (1) the choice of the threshold value k is relevant to mount a successful CPA attack, (2) that low k values increase the probabilities of a successful side-channel attack. We ascribe this to the higher noise setting compared to more controlled context of the encoding analyses (Section V, Section VI).

Our results emphasise the threat that PPS and recombination effects represent. Also, we highlight the practical security impact of different representations of data in a masked software implementation (e.g., logarithm of a share). As a first guideline to mitigate PPS-based leakages exploitation, developers should avoid packing shares within the same word (Listing 2). However, such condition is necessary, but not sufficient, as PPS potentially stems from other sources (Section IV-B).

VIII. DISCUSSION

In this section, we warn about unanticipated sources of weaknesses in masked implementations, then we discuss how parallel-oriented architectures and programming models can introduce PPS in software, and we give some principles to prevent the vulnerabilities created by PPS.

A. ON THE RESILIENCE OF IPM TO TRANSITION-BASED LEAKAGE

In Section V, we have shown that IPM encodings are immune to transition-based leakages, which is consistent with literature knowledge. Yet, in Section VII we were able to successfully attack IPM masked implementations through a leakage model targeting such leakages. We found the root cause in the use of logarithms in the finite field multiplication implementation. Transition-based leakages on logarithm representation of the encodings induced exploitable leakage. Such gap underlines the importance of studying the masking resistance both theoretically and practically. It suggests that the different representations of masked encodings used in an implementation should all be considered for security assessment.

B. PPS AND PARALLEL-ORIENTED ARCHITECTURES

The PPS threat emerges whenever *data processing parallelism* can be achieved. From a hardware point-of-view, PPS readily extends to any architecture encompassing any kind of feature implying data parallelism. In our work, we focused on simple micro-architectures encompassing instruction pipelining, which implies a sort of data parallelism. Gigerl et al. show that super-scalar micro-architectures exhibit more sources of transition-based leakage [3] due to pipeline depth and multiple issuing of instructions. In such micro-architectures, data parallelism is exacerbated, and so the possible occurrence of PPS.

Instruction Set Extensions (ISE) play an important role in the introduction of PPS. Miayjan et al. suggest the employment of SIMD (Single Instruction Multiple Data) ISE to provide efficient and secure software masked implementations [39]. The SIMD ISE enables *data-level* parallel processing, handling multiple data via a single instruction [13]. The explicit data parallelism naturally implies PPS. Such remark extends also to GPU architectures, designed to intrinsically support data-level parallelism. Still, we are not aware of any work concerning their usage to accelerate software masked implementations. Finally, FPGAs represent an interesting case: they can be employed for either the implementation of hardware implementations, or for the implementation of full CPUs [40]. In both cases, the designs might rely on some parallel features, e.g., [41], potentially introducing the PPS vulnerability.

C. PREVENTING PPS IN SOFTWARE

PPS emerges whenever the micro-architecture handles related shares in parallel. As discussed, architectures encompassing parallel features and certain programming models potentially introduce the PPS threat. As a naïve solution, the programmer should rely on programming techniques which do not promote data parallelism, and execute the implementation on architecture not endowed with parallel features. Yet, such approach would increase the already high cost of a masked implementation, in particular for masked instances of order $n > 1$.

Instead, we advocate for a more principled approach, based on the concept of *Non-Completeness*. Non-Completeness is a security property defined in the context of *Threshold Implementations* [42]. Informally, by seeing an n-th order masked algorithm as a composition of sub-functions, each sub-function has to handle no more than n shares. Gaspoz and Dhooghe extend this property to provide *necessary* security properties against micro-architecture-induced recombination effects [43]. In particular, we remark their *Horizontal Register Non-Completeness* as a necessary condition to avoid PPS. Such property contrasts certain programming techniques, e.g., *share-slicing* [21], which aim at the efficient implementation of masked software implementations.

Yet, their notion of non-completeness does not take into consideration the PPS stemming from the pipeline's depth (i.e., number of pipeline stages). Indeed, PPS originates also from related shares manipulated in different pipeline stages. It is possible to extend the non-completeness property at pipeline level, requiring that the pipeline does not process more than n shares at a time. Gigerl et al. suggest a stricter version of this *Pipeline Non-Completeness* property, separating the processing of related shares according to the pipeline's depth and number of instructions that can be executed in parallel to prevent glitch-based leakage [3].

Admittedly, register and pipeline non-completeness might not be sufficient to prevent PPS. Indeed, the register file, caches and memory, potentially store all the shares of an encoding. Static power leakage potentially allows an attacker to observe these shares, enabling successful attacks [44]. The

TABLE 5: Minimum number of traces to mount a successful CPA attack against the AES-128 implementations. We report `failed` in case of attack failure with $1,000,000$ traces.

Device	STM32F215				STM32F303			
Version	Vanilla	BM	ASM	IPM	Vanilla	BM	ASM	IPM
Number of traces	5	140	2970 (HW) 1710 (HD_{fo})	`failed` (HW) 867k ($HD_{fo,log}$)	16	49	241k (HW) 28k (HD_{fo})	`failed` (HW) 82k ($HD_{fo,log}$)

TABLE 6: Key ranking of correct key guess when employing the $SHW_{fo,k\%}$ against IPM implementations. We report the correct key-guess rank and related number of traces for $k \in \{0.1\%, 0.2\%, \dots, 1\%, 2\%\}$. We omit the entries for $k > 2\%$, as we did not succeed in the attack. The number of traces corresponds to the number of *collected* traces (i.e., not the number of traces actually analysed).

	Threshold (%)	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	2
STM32F215	Key rank at 1M traces	1	1	1	1	1	1	1	1	6	7	1
	Number of traces to get to this rank	430k	608k	417k	333k	430k	512k	430k	426k	417k	417k	814k
	Number of traces to get to top 4 rank	416k	417k	235k	333k	412k	401k	417k	417k	—	—	531k
STM32F303	Key rank at 1M traces	3	6	10	7	2	1	4	4	7	7	4
	Number of traces to get to this rank	460k	998k	997k	368k	368k	997k	997k	994k	974k	987k	784k
	Number of traces to get to top 4 rank	460k	—	—	—	324k	494k	997k	994k	—	—	784k

risk implied by static power leakage is still unexplored in the software context.

We conclude this discussion by remarking that the IPM scheme (more generally, the family of *code-based* masking) can *amplify* the security order naturally expected [9], [45], [46]. That is, given a masking of order n, according to the particular public vector $\mathbf{L}$, the security order can be higher than n. Although we analysed IPM instantiated with *non-optimal* codes (i.e., which do not amplify the security order), the use of optimal codes can be a sound way to better mitigate PPS-based leakage. We leave as an interesting future work the investigation of the practical security guarantees of optimal code-based software masked implementations when register and pipeline non-completeness are satisfied.

IX. CONCLUSION

Recent literature has highlighted the CPU micro-architecture as a rich source of recombination effects (e.g., transitions), which severely decrease the security of masking. Although the pervasiveness of such effects, our work shows that they do not represent the only threat to the practical security of masking in software: the parallel processing of share (PPS), exercised by a CPU micro-architecture, represents a potential threat too. Relying on an adaptation of the preprocessing technique proposed by Moos and Moradi [12], we show how to exploit PPS-based leakage against first-order instances of Boolean, arithmetic-sum and inner-product masking. Furthermore, despite the fact that some schemes, such as the inner-product masking, provide immunity to transition-based leakage, particular operations can remove such immunity. Specifically, we show how the employment of the log operation in the field multiplication algorithm allows the successful exploitation of transition-based leakage against the inner-product masking.

ACKNOWLEDGMENT

The authors thank Arnaud de Grandmaison and Emanuele Valea for their helpful comments and many fruitful discussions, Romain Frappier for its contributions to the implementations of masked AES software, and Arnaud de Grandmaison for its contributions to the verification of first-order security in the value-based leakage model.

REFERENCES

[1] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *CRYPTO*, 1999.

[2] B. Marshall, D. Page, and J. Webb, "MIRACLE: micro-architectural leakage evaluation A study of micro-architectural power leakage across many devices," *IACR Trans. Cryptogr. Hardw. Embed. Syst*, 2022.

[3] B. Gigerl, R. Primas, and S. Mangard, "Secure and efficient software masking on superscalar pipelined processors," in *ASIACRYPT*, 2021.

[4] T. D. Cnudde, B. Bilgin, B. Gierlichs, V. Nikov, S. Nikova, and V. Rijmen, "Does coupling affect the security of masked implementations?" in *COSADE*, 2017.

[5] J. Balasch, B. Gierlichs, V. Grosso, O. Reparaz, and F. Standaert, "On the cost of lazy engineering for masked software implementations," in *CARDIS*, 2014.

[6] A. Barenghi and G. Pelosi, "Side-channel security of superscalar CPUs: evaluating the impact of micro-architectural features," in *DAC*, 2018.

[7] A. Barenghi, L. Breveglieri, N. Izzo, and G. Pelosi, "Exploring cortex-m microarchitectural side channel information leakage," *IEEE Access*, 2021.

[8] A. de Grandmaison, K. Heydemann, and Q. L. Meunier, "ARMISTICE: microarchitectural leakage modeling for masked software formal verification," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2022.

[9] Q. Wu, W. Cheng, S. Guilley, F. Zhang, and W. Fu, "On efficient and secure code-based masking: A pragmatic evaluation," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022.

[10] A. Beckers, L. Wouters, B. Gierlichs, B. Preneel, and I. Verbauwhede, "Provable secure software masking in the real-world," in *COSADE*, 2022.

[11] G. Barthe, F. Dupressoir, S. Faust, B. Grégoire, F. Standaert, and P. Strub, "Parallel implementations of masking schemes and the bounded moment leakage model," in *EUROCRYPT*, 2017.

[12] T. Moos and A. Moradi, "On the easiness of turning higher-order leakages into first-order," in *COSADE*, 2017.

[13] J. L. Hennessy and D. A. Patterson, *Computer Architecture - A Quantitative Approach, 5th Edition*, 2012.

[14] L. Goubin and J. Patarin, "DES and Differential Power Analysis (The "Duplication" Method)," in *Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems*, ser. CHES '99, 1999.

[15] J. Balasch, S. Faust, and B. Gierlichs, "Inner product masking revisited," in *EUROCRYPT*, 2015.

[16] T. S. Messerges, "Securing the aes finalists against power analysis attacks," in *Fast Software Encryption*, 2001.

[17] *Advanced Encryption Standard (AES)*, NIST, 2001. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf

[18] L. D. Meyer, E. D. Mulder, and M. Tunstall, "On the effect of the (micro)architecture on the development of side-channel resistant software," *IACR Cryptol. ePrint Arch.*, 2020.

[19] V. Arora, I. Buhan, G. Perin, and S. Picek, "A tale of two boards: On the influence of microarchitecture on side-channel leakage," in *CARDIS*, 2021.

[20] K. Papagiannopoulos and N. Veshchikov, "Mind the gap: Towards secure 1st-order masking in software," in *COSADE*, 2017.

[21] S. Gao, B. Marshall, D. Page, and E. Oswald, "Share-slicing: Friend or foe?" *IACR Trans. Cryptogr. Hardw. Embed. Syst*, 2020.

[22] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The SIMON and SPECK families of lightweight block ciphers," *IACR Cryptol. ePrint Arch.*, 2013.

[23] J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS - kyber: A cca-secure module-lattice-based KEM," in *IEEEEuroS&P*, 2018.

[24] Y. Ishai, A. Sahai, and D. A. Wagner, "Private circuits: Securing hardware against probing attacks," in *CRYPTO*, 2003.

[25] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.

[26] T. Schneider and A. Moradi, "Leakage assessment methodology - A clear roadmap for side-channel evaluations," in *CHES*, 2015.

[27] S. M. Ross, *Introductory Statistics*, 3rd ed., 2010.

[28] N. Veyrat-Charvillon and F. Standaert, "Mutual information analysis: How, when and why?" in *CHES*, 2009.

[29] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information," *Phys. Rev. E*, 2004.

[30] V. Cristiani, M. Lecomte, and P. Maurine, "Leakage assessment through neural estimation of the mutual information," in *ACNS*, 2020.

[31] O. Bronchain, J. M. Hendrickx, C. Massart, A. Olshevsky, and F. Standaert, "Leakage certification revisited: Bounding model errors in side-channel security evaluations," in *CRYPTO*, 2019.

[32] *CW1200 ChipWhisperer-Pro*, NewAE. [Online]. Available: https://rtfm.newae.com/Capture/ChipWhisperer-Pro/

[33] *CW308-STM32F2 - VCC Internal Regulator*, NewAE. [Online]. Available: https://rtfm.newae.com/Targets/UFO%20Targets/CW308T-STM32F/#vcc-int-supply

[34] A. Duc, S. Faust, and F. Standaert, "Making masking security proofs concrete (or how to evaluate the security of any leaking device), extended version," *J. Cryptol.*, 2019.

[35] P. Laarne, "polsys/ennemi: 1.1.1," 2022. [Online]. Available: https://doi.org/10.5281/zenodo.5848134

[36] M. Rivain and E. Prouff, "Provably secure higher-order masking of AES," in *CHES*, 2010.

[37] D. Goudarzi and M. Rivain, "How fast can higher-order masking be in software?" in *EUROCRYPT*, 2017.

[38] D. Blackman and S. Vigna, "Scrambled linear pseudorandom number generators," *ACM Trans. Math. Softw.*, 2021.

[39] A. Miyajan, Z. Shi, C. Huang, and T. F. Al-Somani, "Accelerating higher-order masking of AES using composite field and SIMD," in *IEEE ISSPIT*, 2015.

[40] T. Gokulan, A. Muraleedharan, and K. Varghese, "Design of a 32-bit, dual pipeline superscalar RISC-V processor on FPGA," in *23rd Euromicro Conference on Digital System Design*, 2020.

[41] J. Vliegen, O. Reparaz, and N. Mentens, "Maximizing the throughput of threshold-protected AES-GCM implementations on FPGA," in *IEEE IVSW*, 2017.

[42] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen, "Higher-order threshold implementations," *IACR Cryptol. ePrint Arch.*, 2014.

[43] J. Gaspoz and S. Dhooghe, "Threshold implementations in software: Micro-architectural leakages in algorithms," *CHES*, 2023.

[44] A. Moradi, "Side-channel leakage through static power - should we care about in practice?" in *CHES*, 2014.

[45] W. Wang, F. Standaert, Y. Yu, S. Pu, J. Liu, Z. Guo, and D. Gu, "Inner product masking for bitslice ciphers and security order amplification for linear leakages," in *CARDIS*, 2016.

[46] W. Cheng, S. Guilley, C. Carlet, J. Danger, and S. Mesnager, "Information leakages in code-based masking: A unified quantification approach," *IACR Trans. Cryptogr. Hardw. Embed. Syst*, 2021.

LORENZO CASALINO received his master's degree in Computer Science and Engineering at Politecnico di Milano (Italy), in 2020. He is currently pursuing the Ph.D. degree at CEA-List (Grenoble, France). His research interests focus on side-channel analyses, related micro-architecture-aware countermeasures and their automated application.

NICOLAS BELLEVILLE received the PhD degree from the Univ. Grenoble Alpes, France, in 2019. Ever since, he has been working as a researcher in CEA, in Grenoble, France. His research interests include side-channel attacks, their countermeasures, and the automated application of countermeasures during compilation.

DAMIEN COUROUSSÉ is with CEA-List since 2011, as a Research Engineer and Senior Expert. He received the Ph.D. from the Institut National Polytechnique de Grenoble in 2008. His research interests include embedded software and its interaction with hardware, compilation and runtime code generation for performance and security, with a recent focus on hardware security.

KARINE HEYDEMANN has been an Associate Professor at Sorbonne University / LIP6 from 2006 to 2022. She works at Thales DIS as a Senior Expert Architect and is an Associated Researcher of the LIP6. She received the Ph.D. degree in Computer Science from the University of Rennes 1 in 2004. Her areas of expertise encompass hardware micro-architecture, compilation, code optimization, and physical attacks, including modelling of hardware fault injection effects, automated code hardening and robustness analysis.