Leveraging Coprocessors as Noise Engines in Off-the-Shelf Microcontrollers.

Balazs Udvarhelyi and François-Xavier Standaert

UCLouvain, ICTEAM, Crypto Group, Louvain-la-Neuve, Belgium
firstname.lastname@uclouvain.be

Abstract. Securing low-cost microcontrollers against side-channel attacks is an important challenge. One core issue for this purpose is that such devices may exhibit leakages with very limited noise. As a result, standard countermeasures like shuffling or masking, which emulate or amplify noise, have limited effectiveness. In this paper, we investigate the possibility to run hardware coprocessors in parallel to a masked software implementation, in order to generate algorithmic noise. We detail the conditions for such a noise generation to be effective and show experimental evidence that it leads to security improvements compared to masked software implementations running without activated coprocessors. While masking remains expensive, the gains we show in number of traces to recover the key are systematic: an approximate factor two in our experiments, that is raised to the number of masking shares.

1 Introduction

Side-channel attacks are an important threat against which various countermeasures exist. Among generic solutions that can be applied independent of the algorithms, a standard idea is to emulate and amplify the physical noise inherently present in the measurements. Masking [CJRR99,GP99] and shuffling [HOM06,VMKS12] are typical examples of such countermeasures. In particular, it is now established that assuming shares' leakages that are sufficiently independent, masking can amplify the noise (variance) exponentially in the number of shares [PR13,DDF14,DFS15,PGMP19,BCG$^+$23].

In general, such countermeasures can of course only work if there is indeed some noise inherently present in the leakages. This requirement turns out to be quite easily achieved in hardware implementations contexts (e.g., FPGAs, ASICs), since designers then have a good level of control on the design. As a result, the primary constraint in those cases is to ensure the independence of the leakages, which is reflected by the security order (i.e., the smallest statistical moment of the leakage distribution that depends on the target secret). This is still not trivial and requires dealing with physical defaults like glitches or transitions, but it is now quite well understood and design principles exist to deal with such defaults. See for example [NRS11,FGP$^+$18,CGLS21] for glitches and [CGP$^+$12,BGG$^+$14,CS21] for transitions.

Unfortunately, the situation quite significantly differs in the context of software implementations running on Commercially available Off-The-Shelf (COTS) Micro-Controller Units (MCUs). On the one hand, such (small) devices usually exhibit limited physical noise. On the other hand, their serial nature lets the adversary mounting so-called horizontal attacks that combine the leakages of multiple operations in order to further reduce the noise [BCPZ16]. As a result, a recent work showed that state-of-the-art masked implementations of the AES can be broken nearly independently of the security order (i.e., that exploiting the low noise is the best strategy independent of physical defaults like transitions that may show up in software implementations) [BS21].

In this paper, we are therefore interested in possibilities to improve the noise level of COTS devices. Our general idea relies on the presence of peripherals alongside the main CPU of recent MCUs, which could be run in parallel and provide additional algorithmic noise. Such an algorithmic noise could in turn make masking more effective and possibly reduce the number of shares needed to reach a given security level. Natural candidates for this purpose are cryptographic coprocessors as they have the advantage of producing seemingly random computations, can be supplied via the same power supply and may rely on large (parallel) architectures generally leading to large(r) levels of noise.

In order to evaluate the effectiveness of such a possibility, we run state-of-the-art attacks against a masked bitslice implementation of the AES with different number of shares. We show that running dummy coprocessor operations in parallel to the main (masked) computations indeed improves the noise level by a factor $f \approx 2$ in our experiments, leading to an increase of the best attacks' complexity by a factor $\approx 2^n$ for n shares. As a side result, we also show that the MCU we consider, which uses a more advanced technology than the one of [BS21], leads to slightly less informative leakages even without running coprocessors in parallel. These combined observations lead to improved security for masked software implementations. Reaching high security levels remains expensive but overheads are reduced. For example, we were not able to attack a 4-share implementation with additional noise in less than one million traces.

We note that our work is focused on power measurements (or more generally, global leakages). This excludes advanced adversaries taking advantage of high-resolution localized measurements like [UHSS17,UHS+18], which generally work best in an invasive attack setting (i.e., after depackaging). We leave the investigation of such advanced attacks as an interesting open problem.

2 Background

In this section, we describe the necessary tools that are used in the paper. Namely, we first describe out notations, then the Signal-to-Noise Ratio (SNR), the Regression-based Linear Discriminant Analysis (RLDA) leakage model, Soft Analytical Side-Channel Attacks (SASCA) and the Perceived Information (PI) metric. Finally, we describe the masking countermeasure.

2.1 Notations.

We use capital letters X for random variables, bold letters $\boldsymbol{x}$ for vectors, capital bold letters $\boldsymbol{X}$ for matrices and calligraphic letters $\mathcal{X}$ for sets. We use subscript to indicate shares, if relevant. We additionally use the following conventions: n_s is the number of samples in a trace, n_p and n_a are respectively the number of profiling and attack traces, b is the number of bits of the profiled variable, and p is the dimensionality of the subspace used by RLDA.

2.2 Signal-to-Noise Ratio

The SNR is a common univariate metric in side-channel analysis. It was first defined by Mangard in [Man04]. For an intermediate variable X, it models the signal as the variance of the mean leakage of each class $x \in \mathcal{X}$, and the noise as the mean of the variance of the leakage of each class:

$$\hat{\mathrm{SNR}} = \frac{\hat{\mathsf{Var}}_x\left(\hat{\mathsf{E}}_i\left(l_i^x\right)\right)}{\hat{\mathsf{E}}_x\left(\hat{\mathsf{Var}}_i\left(l_i^x\right)\right)}, \tag{1}$$

where l_i^y corresponds to the i^{th} leakage sample of variable y. $\hat{\mathsf{Var}}_i$ and $\hat{\mathsf{E}}_i$ (resp., $\hat{\mathsf{Var}}_x$ and $\hat{\mathsf{E}}_x$) are the sample variance and the sample mean over the leakages (resp. the classes). We note here that the modeled noise is a combination of physical and algorithmic noise. Mangard's SNR is a good estimator for the complexity of univariate attacks like Correlation Power Analysis or (univariate) template attacks as their complexity can be directly linked to the SNR [Man04,MOS11,DFS19]. The SNR can also be used as tool to detect Points-of-Interest (POIs) as the time samples in a measured leakage trace where the SNR is high represent samples for which first-order information can be extracted.

2.3 Regression-Based Linear Discriminant Analysis

RLDA has been introduced in [CK14]. Its core idea was to replace the mean of each class in the equations of Linear Discriminant Analysis (LDA) [SA08] by a value obtained through linear regression [SLP05]. We use the efficient implementation of Cassiers et al. [CDSU23], which combines the efficient profiling of large states enabled by linear regression and the ability to profile models for long traces enabled by the dimensionality reduction embedded into LDA.

Internally, RLDA first fits a linear regression model with n_b basis functions β_i corresponding to the b bits of the target variable and the intercept:

$$\beta_i(x) = \begin{cases} 1 & \text{if } i = 0, \\ 1 & \text{if } \left\lfloor x/2^{i-1} \right\rfloor \mod 2 = 1, \\ -1 & \text{otherwise,} \end{cases} \tag{2}$$

with $n_b = b + 1 = \lceil \log_2(|\mathcal{X}|) \rceil + 1$. The regression model to fit for each leakage sample s is:

$$\mathsf{m}_s(x) = \sum_{i=0}^{n_b-1} a_{i,s}\beta_i(x),$$

where $a_{i,s}$ corresponds to the ith coefficient of leakage sample s. We define the mean vector of x as $\mathsf{m}(x) = \boldsymbol{A}\beta(x)$ with $\boldsymbol{A} \in \mathbb{R}^{n_s \times n_b}$ the matrix of all coefficients. Then, this model is used to calculate the inter and intra-class scatter matrices:

$$\boldsymbol{S_B} = \sum_{x \in \mathcal{X}} |\mathcal{L}(x)| \, (\mathbf{m}(x) - \hat{\boldsymbol{\mu}})(\mathbf{m}(x) - \hat{\boldsymbol{\mu}})^T, \tag{3}$$

$$\boldsymbol{S_W} = \sum_{x \in \mathcal{X}} \sum_{\boldsymbol{l} \in \mathcal{L}(x)} (\boldsymbol{l} - \mathbf{m}(x))(\boldsymbol{l} - \mathbf{m}(x))^T, \tag{4}$$

where $\mathcal{L}(x)$ defines the subset of the trace set $\mathcal{L}$ where $x = X$. By solving the following problem :

$$\boldsymbol{W} = \underset{\boldsymbol{W} \in \mathbb{R}^{n_s \times p}}{\arg\max} \frac{\left|\boldsymbol{W}^T \boldsymbol{S_B} \boldsymbol{W}\right|}{\left|\boldsymbol{W}^T \boldsymbol{S_W} \boldsymbol{W}\right|},$$

we find the projection matrix maximizing the SNR in the projected subspace, where p is a parameter corresponding to the number of dimensions in the subspace [DHS01]. Next, the covariance matrix of the Gaussian model in the subspace is estimated from the intra-class scatter $\hat{\Sigma}_{\boldsymbol{W}} = |\mathcal{L}|^{-1} \boldsymbol{W}^T \boldsymbol{S_W} \boldsymbol{W}$, and a second, normalizing projection is applied such that the covariance matrix becomes identity. To do so, by computing the eigendecomposition of the symmetric positive-definite covariance matrix:

$$\hat{\Sigma}_{\boldsymbol{W}} = \boldsymbol{V} \Lambda \boldsymbol{V}^T,$$

where $\boldsymbol{V}$ and Λ are respectively the matrix of eigenvectors and eigenvalues, we define the normalizing projection $\boldsymbol{W}^{\mathsf{norm}} = \boldsymbol{V}\Lambda^{-1/2}$. Eventually, the RLDA model is obtained by:

$$\hat{\mathsf{f}}[\boldsymbol{l}|X = x] = \frac{1}{\sqrt{(2\pi)^p}} \exp\left(-\frac{1}{2} \left\| \boldsymbol{W}^{\mathsf{RLDA}} \boldsymbol{l} - \boldsymbol{A}^{\mathsf{RLDA}} \beta(x) \right\|^2\right), \tag{5}$$

with $\boldsymbol{W}^{\mathsf{RLDA}} = \boldsymbol{W}\boldsymbol{W}^{\mathsf{norm}}$ the combined projection matrix and $\boldsymbol{A}^{\mathsf{RLDA}} = \boldsymbol{W}^{\mathsf{RLDA}}\boldsymbol{A}$ the projected coefficients. The likelihood of X conditioned on the leakages are obtained using Bayes' law:

$$\hat{\mathsf{f}}[X = x|\boldsymbol{l}] = \frac{\hat{\mathsf{f}}[\boldsymbol{l}|X = x]\mathsf{Pr}[X = x]}{\sum_{x' \in \mathcal{X}} \hat{\mathsf{f}}[\boldsymbol{l}|X = x']\mathsf{Pr}[X = x']}.$$

In the following sections, we also use Gaussian templates with LDA-based dimensionality reduction [CRR02,SA08], of which the calculation is essentially the same except that the means in Equation 3 are calculated exhaustively.

2.4 Perceived Information

Evaluating the success rate of a multivariate attack cannot be done with the SNR (which is a univariate metric). The Mutual Information (MI) between the leakage and target variable is a good candidate for this purpose. However, it is notoriously hard to estimate. We therefore use the PI as a surrogate, which provides an easy to estimate lower bound [BHM+19]. It can be computed by sampling the model $\hat{\mathsf{f}}$ as follows:

$$\hat{\mathsf{PI}}(X, \boldsymbol{L}) = \mathrm{H}(X) + \sum_{x \in \mathcal{X}} \mathsf{Pr}[x] \sum_{\boldsymbol{l}' \in \mathcal{L}'(x)} \frac{1}{|\mathcal{L}'(x)|} \log_2 \hat{\mathsf{f}}[x|\boldsymbol{l}'], \tag{6}$$

where $\mathrm{H}(X)$ is the entropy of X and $\mathcal{L}'$ is the set of traces used to estimate the PI, which must differ from the set of traces $\mathcal{L}$ used to fit the model.

2.5 Boolean Masking

Masking is a common countermeasure against side-channel attacks. It relies on an encoding of any sensitive variable x into a tuple of n shares:

$$x = x_1 \oplus x_2 \oplus \ldots \oplus x_n,$$

where the $n-1$ first shares are selected uniformly at random, so that an adversary needs knowledge of all the shares in order to recover the sensitive information. This security guarantee is easily expressed in the abstract probing model [ISW03], which states security if an adversary who can probe up to $n-1$ wires in the circuit does not learn anything about x.

From an implementation viewpoint, linear operations are performed share-by-share and have a linear complexity. Multiplications of two encodings are more complex: they have quadratic complexity in the number of shares and require additional randomness to remain d-probing secure. Popular algorithms are the ISW multiplication [ISW03] and the PINI gadgets introduced in [CS20].

Probing security reduces to noisy leakage security [DDF14], which is closer to real-world measurements. The noisy leakage model assumes that all the shares leak noisy observations to the adversary. In this case, the data complexity N of the attack is inversely proportional to the product of the MI between each share and the leakage [BCG+23], so that:

$$N \geq \frac{c}{\prod_{i=1}^{n} \mathrm{MI}(X_i, L)}, \tag{7}$$

increases exponentially with the number of shares given that the MI per share is small enough. As mentioned in introduction, this guarantee only holds as long as the leakage function ensures some independence between the shares' leakages (i.e., as long as it does not recombine the shares). When replacing the MI by the PI in this equation, we no longer have a lower bound but an estimate that measures the particular model used to estimate the PI.

Attacking a masked implementation requires the adversary to model each share independently to obtain probability densities on the shares $\hat{\mathsf{f}}[x_i|\boldsymbol{l}]$, and then to recombine them to obtain probability densities on the target variable:

$$\hat{\mathsf{f}}[x|\boldsymbol{l}] \propto \sum_{\{x_0,\ldots,x_{n-1}|\sum x_i=x\}\in\mathcal{X}^n} \prod_{i=1}^{n} \hat{\mathsf{f}}[x_i|\boldsymbol{l}] \tag{8}$$

2.6 Soft Analytical Side-Channel Attacks

The previous sections showed how we can profile the leakage of an intermediate variable X. However, during the computation of a masked encryption algorithm, several intermediate states exist which can all leak useful information. Directly profiling many variables in the same template is rapidly impractical as the number of classes grows exponentially. SASCA has been introduced as an efficient way to profile several variables independently and exploit them jointly [VGS14]. For this purpose, it models the relations between the variables with a factor graph and leverages the Belief Propagation (BP) algorithm.

Concretely, the factor graph is a bipartite graph containing the variables (circles) on one side and the relations (squares) on the other. The edges represent the relations between the variables. An example is given in Figure 1.

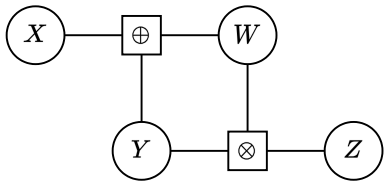


Fig. 1: Exemplary factor graph for $X \oplus Y = W$ and $Y \otimes Z = W$.

Internally, the BP algorithm iterates over 3 steps. At iteration t, we have:

1. The belief of a variable X is computed as the product of its likelihood and the beliefs coming from its neighbors ∂X.

$$P_X^{t+1}(x) = \hat{\mathsf{p}}(X = x) \prod_{R\in\partial X} m_{R\to X}^t(x),$$

2. From variable X to a relation R, the belief is the product of the beliefs on the variable divided by the belief from the destination relation.

$$m_{X\to R}^{t+1}(x) = P_X^t / m_{R\to X}^t(x).$$

3. From relation R to variable X, the belief is the sum over all compatible values for the relation of the products of the beliefs from all neighbors of R except X, where we denote the compatibility function with ψ_R, which has a value of 1 if the values are compatible with the relation and 0 otherwise:

$$m_{R\to X}^{t+1}(x) = \sum_{x_i\in\mathcal{X}_i \text{ s.t. } \{X_1,\ldots,X_k\}=\partial R\setminus X} \psi_R(x, x_1, \ldots, x_k) \prod_{i=1}^{k} m_{X\to R}^{t}(x_i),$$

This algorithm is proven to converge on a tree-like structure with a number of iterations corresponding to twice the diameter of the graph. With graphs containing cycles like Figure 1, the algorithm becomes a heuristic.

3 Noise Generation

In this section, we detail the implementation of our coprocessor-based noise engine. In order to be characterized as noise, the coprocessor should process hard-to-predict data, with ideally random states at each cycle. This way, an adversary is not able to predict the data and its leakage, which would make the noise deterministic and easy to filter. It should ideally be slow (i.e., take many cycles of computation) but quick to configure such that interrupts of the main task are sparse and short. The coprocessor should also rely on a large (parallel) architecture in order to generate more algorithmic noise and to better hide the leakage of the CPU that runs a masked implementation.

In practice, we configured a Direct Memory Access (DMA) peripheral which loads the buffer to process into to coprocessor and unloads it without the CPU being interrupted. The buffer can contain multiple blocks of data to process. When the buffer is fully processed, the CPU gets interrupted, the DMA and the coprocessor are reset and the cycle can restart. The buffer size therefore gives a trade-off parameter between the memory used and the overhead in execution time due to the interrupts. We note that the suitable coprocessors of the MCU we used did not allow circular DMA transfers.[1] Hence we needed to reset after processing the buffer. A representation of this process is shown in Figure 2.

The coprocessor we choose for our experiments was an AES-128 implementation with a 128-bit architecture, performing the encryption of one block of data in 16 cycles (i.e., one cycle per round). In order for its states to remain unpredictable, we put this coprocessor in a leakage-resilient mode of operation for which it is expected that a large parallel implementation provides sufficient security. Concretely, we used a construction similar to the one in [BMPS21] where it is shown that the 128-bit coprocessor maintains 64 bits of security as long as we re-key it every 100 encryptions. We used the True Random Number Generator (TRNG) of the MCU to generate the random buffer and initializing

[1] Which are DMA transfers where the DMA automatically resets its pointer to the start address at the end of the buffer, without input from the CPU.

the coprocessor with a random key. The TRNG generated a 32-bit random word every 40 cycles. After each completed buffer, we generate a new random key with the TRNG and encrypt the buffer with the new key. Overall, this strategy keeps performance overheads due to resets quite small (a few percents).

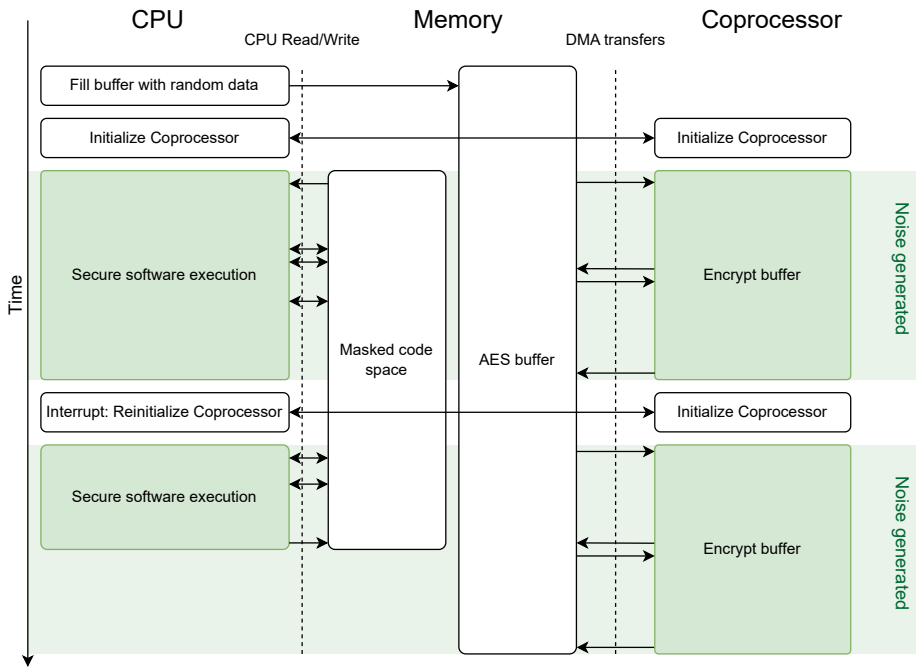


Fig. 2: Execution scheme of the noise engine.

We note that reducing the buffer size would increase the computation time of our masked implementation, but improve the security of the leakage-resilient mode of operation running in parallel. It could also improve the security of the masked implementation due to the desynchronization of the traces that it would imply (since interrupts could then act as somewhat random delays).

4 Evaluation Setup

We now describe the implementation we analyze and our measurement setup.

4.1 Description of the Target

The implementation under test is the bitsliced AES implementation of Goudarzi and Rivain [GR17] with state-of-the-art PINI gadgets from [CS20]. We precisely

used the assembly code provided in [BC22]. Bitslicing allows efficient masked software implementations. It works by splitting each word and placing each bit into a different register, so that bitwise operations can be applied at the register level. Given the size of the CPU, we are not limited to one bit per register and can easily parallelize all the 16 S-boxes of the AES in 32-bit registers. In the case of masked implementations, each register is shared into n registers such that in one register, only bits of one share are present. This avoids recombining shares in the barrel shifter, also known as the shareslicing issue [GMPO20].

4.2 Measurement Setup

Our measurements were performed on a Chipwhisperer CW308 board with the `STM32F415RG` daughterboard. This board includes an ARM Cortex-M4 CPU with the required peripherals. We used a Tektronix CT-1 AC current probe on the dedicated measuring pins.[2] For sampling, we used the Picoscope 5244D 12-bit oscilloscope at its maximum speed of 500MS/S. The clock of our DUT was set at 40MHz and derived with internal PLLs from an 8MHz crystal on the CW308board. We note that we also performed measurements to make sure our measurement setup produces results close to Bronchain and Standaert's measurement setup [BS21]. For this, we kept our setup identical with the exception of the daughterboard being a `STM32F051R4` with a Cortex-M0 CPU.

5 Side-Channel Metrics

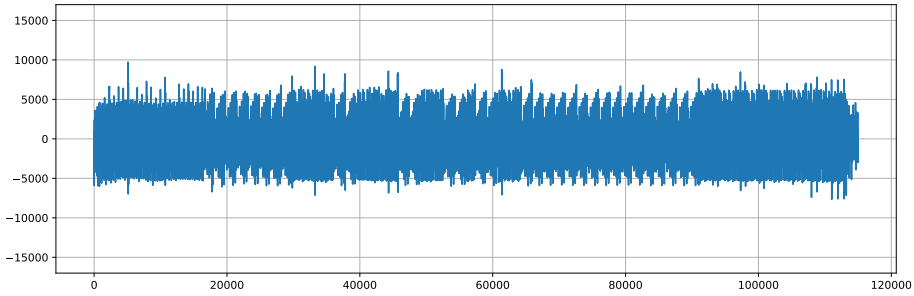
In this section, we compare two side-channel metrics, namely the SNR and the PI. Beforehand, we show in Figure 3 the effect of the coprocessor on the measurements by showing mean leakage traces with and without the noise generation. The mean leakage traces were computed over 100k non-averaged traces and represent the execution of the first S-box layer of the AES. We clearly see the impact of the co-processor: without it, groups of operations are distinguishable; when activated, they are hidden within the generated noise.

5.1 SNR Comparison

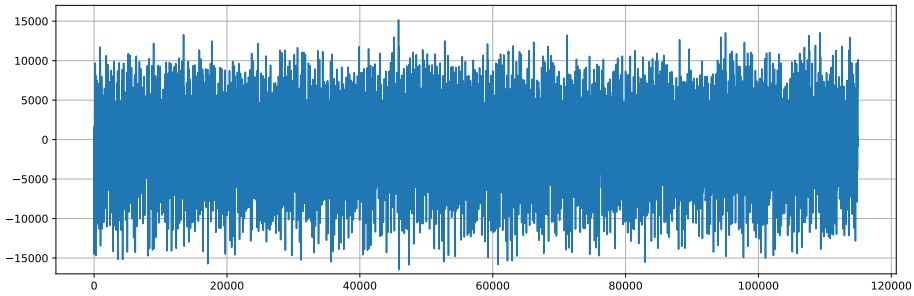
First, we compare the SNR with and without the noise engine for each state in the computation of the first S-box Layer. The target states have a 16-bit width, corresponding to the 16 S-Boxes computed in parallel. In each word, 16 bits correspond to the 16 S-boxes and the 16 remaining ones are set to zero. Thus, each bit of the bus is modeled and no algorithmic noise comes from the bus. Every state corresponds to a share of an intermediate variable and is processed independently. We used 500k traces with random inputs to compute the SNR.

In Figure 4, we show the (123×2) SNR curves calculated for each intermediate state of the AES bitslice S-box for $n = 2$ shares, with (top) and without (bottom)

[2] This effectively shorts the onboard shunt resistor.



(a) Without noise generation.



(b) With noise generation.

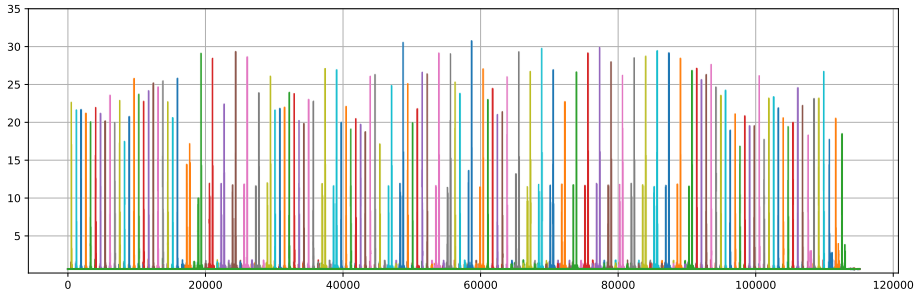
Fig. 3: Comparison of mean leakage traces.

noise generation. We observe two types of SNR peaks in the upper figure: the tighter ones are the XOR (and NXOR) gadgets, the more spread out, and slightly higher ones correspond to the AND gadgets. The impact of the noise generation appears on the bottom figure, where all time samples exhibit a reduced SNR. There is no significant alteration of the shape of the SNR curves. We observed a similar reduction for implementations with more shares.

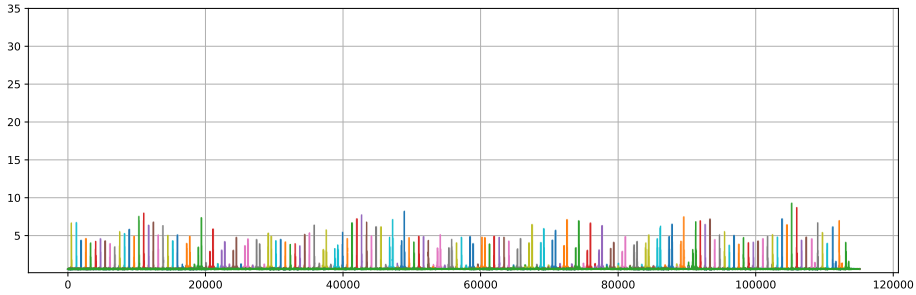
5.2 PI Comparison

In order to evaluate the reduction of the PI due to the coprocessor, we first detail the steps followed to generate our leakage models. First, for each intermediate state, we find our Points-Of-Interest (POIs) which we define as the time samples that are higher than the noise floor of the SNR. It leads to around 2000 samples for each state share. We then profile these intermediate state shares with 500k traces using the RLDA leakage model on 16-bits, with 10 dimensions in the subspace. Finally, 5000 fresh traces are used to evaluate the PI.

To justify these parameters, Figure 5 shows the extracted information for an exemplary state share as a function of the number of profiling traces used, with and without the noise generation. It can be seen that all models converge with



(a) Without noise generation



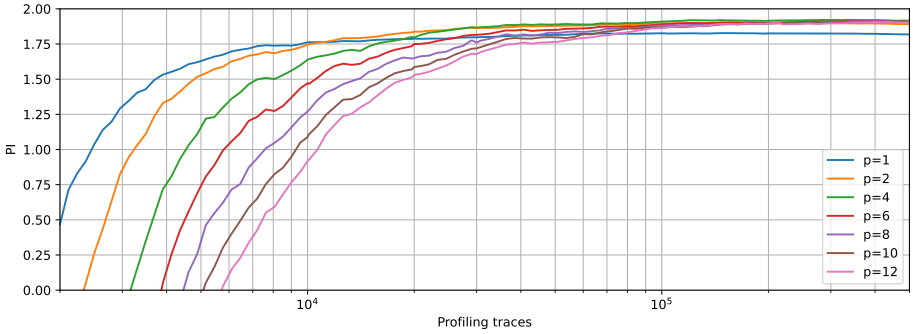
(b) With noise generation

Fig. 4: SNR for each share of the first S-box layer masked with two shares.

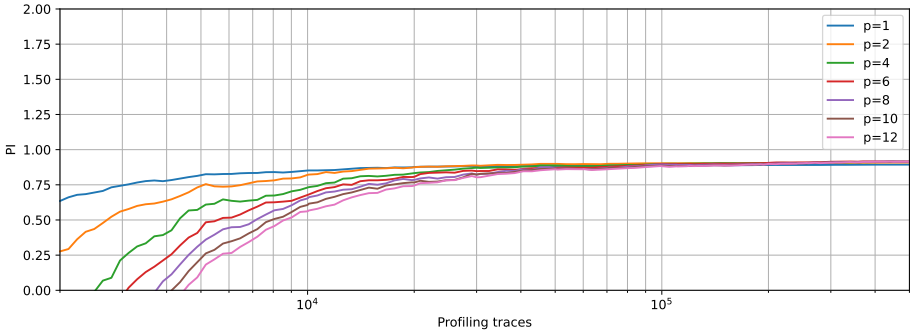
$\approx$ 100k traces. Furthermore, increasing the number of dimensions (e.g., beyond p=4) has limited impact, especially in the noisy case (bottom figure).

We next show the PI per share of the input of the S-box layer in Table 1, computed from a 2-share implementation. We observe that the 8 words of the AES state have slightly different levels of information, but the noise produced by the coprocessor consistently reduces it. This is independent of the share that is being modeled. We also note that the PI per share of the linear operations is stable with the number of shares (since these operations are applied share-wise). Since our following attacks will primarily exploit this information, we do not detail the evaluation of the PI per share for the non-linear operations that takes place for larger numbers of shares (due to multiple shares manipulations). On average, we observed a reduction by an approximate factor 2 for the PI per share when activating the noise generation. Based on this value, we can extrapolate the security gains that the noise generation brings thanks to Equation 7: it should increase the data complexity by an approximate factor 2^n with n shares.[3]

[3] This is assuming that the independence condition holds to a sufficient degree. Since the gadgets we use were tested against such defaults in [BC22] and the possible reduction of the security order due to transitions is orthogonal to the noise issue we discuss, we did not reproduce this part of the experiments in the paper.



(a) Without noise generation



(b) With noise generation

Fig. 5: PI per share obtained for an exemplary variable.

For completeness, we also computed the results obtained with two 8-bit LDA-based models, in order to compare them with the 16-bit RLDA-based model. As shown in Table 2, this leads to significant reduction of the perceived information. This last table is interesting since it uses a similar model as [BS21] and shows significantly lower PI values that this previous work obtained with a Cortex-M0 `STM32F0` MCUs. We repeated the measurements of [BS21] with our measurement setup (just plugging/unplugging the devices) and obtained similar results. So we posit that the changes are due to the higher complexity of the Cortex-M4 and a change of technology node, from 180nm for the F0 line to 90nm for the more recent (and lower power) F4 line we evaluate in this paper.[4]

6 Attack results

In this section, we finally present concrete attacks against implementations with different masking orders, and discuss the effectiveness of the proposed noise gen-

[4] https://blog.st.com/stm32g0-mainstream-90-nm-mcu/
https://www.st.com/en/microcontrollers-microprocessors/stm32f405-415.html

	Share #	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7
No noise	0	1.90	1.53	1.21	1.89	0.81	2.00	2.47	3.12
	1	1.97	1.64	1.29	1.97	0.67	2.12	2.60	3.91
With noise	0	0.95	0.71	0.42	0.72	0.37	0.92	1.06	1.69
	1	1.06	0.67	0.46	0.76	0.30	1.10	1.12	1.78

Table 1: PI per share: input of the S-box layer, 2 shares, 16-bit RLDA model.

	Share #	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7
No noise	0	0.89	0.75	0.76	0.98	0.57	0.87	1.25	1.56
	1	0.98	0.96	0.86	1.05	0.50	1.18	1.56	2.39
With noise	0	0.62	0.49	0.35	0.52	0.30	0.59	0.75	1.01
	1	0.70	0.49	0.40	0.55	0.26	0.79	0.77	1.21

Table 2: PI per share: input of the S-box layer, 2 shares, two 8-bit LDA models.

eration. First, a baseline template attack is shown against the key addition. Next, we compare our results to the attack presented in [BS21]: a SASCA exploiting the leakages of all the intermediate states of the S-box layer.

This baseline attack is a textbook template attack on each of the 8 words of the key addition. We profile each share of each word as explained in subsection 5.2, using the 16-bit RLDA models. Then, we recombine the likelihoods obtained on our shares to obtain the likelihood of the unmasked variable as shown in Equation 8. Figure 6 shows the results of attacks against 3 masked implementations, with 2, 3 and 4 shares. For each number of shares, we measured a dataset with and without noise generation. As a reference, we also ran the attack against an implementation without masking. The figure shows the mean rank (in bits) of the key as a function of the number of traces used in the attacks [SMY09]. We used the rank estimation algorithm presented in [PSG16]. The horizontal black line represents a rank of 32 bits, set as a limit for the success of the attack (below that rank, enumeration is almost instantaneous).

First, looking at the dashed lines, the impact of masking is clearly visible. For each additional share, roughly 10 times more traces are needed. Next, the impact of the noise generation can be seen in the rightwards shift when moving from the dashed curves to the plain curves. As expected from our previous extrapolation, the security gain brought by the additional algorithmic noise is amplified as the number of shares increases. This is reflected by the gap between curves of the same color, that grows roughly with 2^n. For the 4-share design, we were not able to reduce the key rank below 2^{32} with one million attack traces.

For completeness, we repeated the SASCA of [BS21], where the adversary computes probabilities on the intermediate variables using Equation 8 and combines them thanks to BP. We performed this attack using the same datasets as

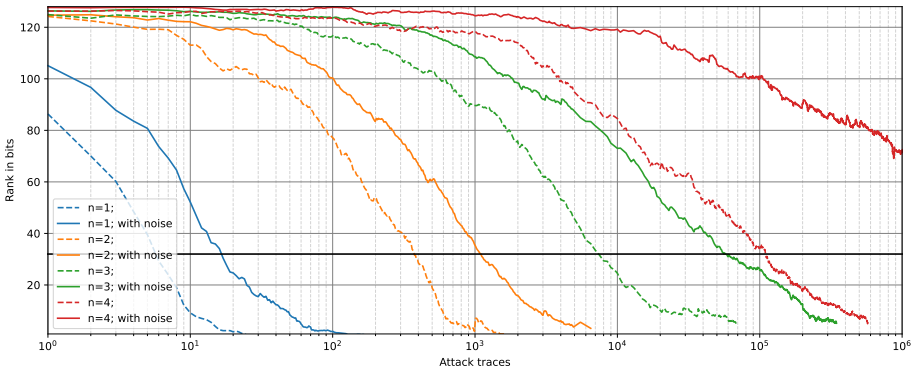


Fig. 6: Key rank for $\neq$ number of shares, with and without noise generation.

the baseline attack. All variables were profiled using the same RLDA models and parameters. The factor graph of the AES being cyclic, we studied the number of iterations and observed the best results after 5 or 6 iterations, depending on the cases. Iterating the BP algorithm further lead to worse ranks on the key.

The results of the SASCA and baseline attacks are compared in Table 3, which gives the number of traces required to reach the rank of 2^{32} (mostly because the improvements are hardly distinguishable from Figure 6). The improvement of the SASCA over the baseline attack is limited and decreases with the masking order. We posit that this limited effectiveness is due to the lower information obtained on the target intermediate variables combined with the information loss when propagating beliefs through XOR operations.

	$n=2$		$n=4$		$n=4$	
Noise generation	Off	On	Off	On	Off	On
TA	373	1101	7500	36500	110500	NA.
SASCA	295	1024	7000	36000	109000	NA.
Gain in %	21	7	7	1.3	1.3	NA.

Table 3: Data complexity to obtain a 2^{32} rank with SASCA (vs. template attack).

7 Conclusions

The versatility of low-end MCUs makes them appealing solutions for deployment in various applications. Yet, their simplicity also makes them good targets for physical attacks. Previous works showed that their low-noise makes them vulnerable to horizontal side-channel analysis. In this work, we show that the move

to more advanced manufacturing technologies combined with the generation of algorithmic noise thanks to coprocessors running in parallel to masked software implementations can improve this situation with limited performance overheads.

Activating co-processors admittedly has a cost in terms of power consumption. However, we expect that this higher power consumption can be compensated by the lower number of shares required to reach a given security level – the concrete confirmation of this tradeoff being an interesting scope for further research. It would also be interesting to further improve the side-channel security of software implementations by combining the noise engines we propose with other heuristic means to reduce the leakage (e.g., via time randomizations).

Acknowledgments. François-Xavier Standaert is a senior research associate of the Belgian Fund for Scientific Research (FNRS-F.R.S.). This work has been funded in parts by the ERC Consolidator grant 724725 (acronym SWORD), the ERC Advanced Grant 101096871 (acronym BRIDGE) and by the EU-Walloon FEDER Project USERMedia (convention 501907-379156). Views and opinions expressed are those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

BC22. Olivier Bronchain and Gaëtan Cassiers. Bitslicing arithmetic/boolean masking conversions for fun and profit with application to lattice-based kems. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):553–588, 2022.

BCG+23. Julien Béguinot, Wei Cheng, Sylvain Guilley, Yi Liu, Loïc Masure, Olivier Rioul, and François-Xavier Standaert. Removing the field size loss from duc et al.'s conjectured bound for masked encodings. In *COSADE*, volume 13979 of *Lecture Notes in Computer Science*, pages 86–104. Springer, 2023.

BCPZ16. Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In *CHES*, volume 9813 of *Lecture Notes in Computer Science*, pages 23–39. Springer, 2016.

BGG+14. Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In *CARDIS*, volume 8968 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014.

BHM+19. Olivier Bronchain, Julien M. Hendrickx, Clément Massart, Alex Olshevsky, and François-Xavier Standaert. Leakage certification revisited: Bounding model errors in side-channel security evaluations. In *CRYPTO (1)*, volume 11692 of *Lecture Notes in Computer Science*, pages 713–737. Springer, 2019.

BMPS21. Olivier Bronchain, Charles Momin, Thomas Peters, and François-Xavier Standaert. Improved leakage-resistant authenticated encryption based on hardware AES coprocessors. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):641–676, 2021.

BS21. Olivier Bronchain and François-Xavier Standaert. Breaking masked implementations with many shares on 32-bit software platforms or when the

security order does not matter. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):202–234, 2021.

CDSU23. Gaëtan Cassiers, Henri Devillez, François-Xavier Standaert, and Balazs Udvarhelyi. Efficient regression-based linear discriminant analysis for side-channel security evaluations: Towards analytical attacks against 32-bit implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(3):270–293, 2023.

CGLS21. Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Trans. Computers*, 70(10):1677–1690, 2021.

CGP+12. Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. Conversion of security proofs from one leakage model to another: A new issue. In *COSADE*, volume 7275 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2012.

CJRR99. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

CK14. Marios O. Choudary and Markus G. Kuhn. Efficient stochastic methods: Profiled attacks beyond 8 bits. In *CARDIS*, volume 8968 of *Lecture Notes in Computer Science*, pages 85–103. Springer, 2014.

CRR02. Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

CS20. Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020.

CS21. Gaëtan Cassiers and François-Xavier Standaert. Provably secure hardware masking in the transition- and glitch-robust probing model: Better safe than sorry. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):136–158, 2021.

DDF14. Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In *EUROCRYPT*, volume 8441 of *Lecture Notes in Computer Science*, pages 423–440. Springer, 2014.

DFS15. Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete - or how to evaluate the security of any leaking device. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 401–429. Springer, 2015.

DFS19. Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete (or how to evaluate the security of any leaking device), extended version. *J. Cryptol.*, 32(4):1263–1297, 2019.

DHS01. Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern classification, 2nd Edition.* Wiley, 2001.

FGP+18. Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.

GMPO20. Si Gao, Ben Marshall, Dan Page, and Elisabeth Oswald. Share-slicing: Friend or foe? *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):152–174, 2020.

GP99. Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In *CHES*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.

GR17. Dahmun Goudarzi and Matthieu Rivain. How fast can higher-order masking be in software? In *EUROCRYPT (1)*, volume 10210 of *Lecture Notes in Computer Science*, pages 567–597, 2017.

HOM06. Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An AES smart card implementation resistant to power analysis attacks. In *ACNS*, volume 3989 of *Lecture Notes in Computer Science*, pages 239–252, 2006.

ISW03. Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

Man04. Stefan Mangard. Hardware countermeasures against DPA ? A statistical analysis of their effectiveness. In *CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 222–235. Springer, 2004.

MOS11. Stefan Mangard, Elisabeth Oswald, and François-Xavier Standaert. One for all - all for one: unifying standard differential power analysis attacks. *IET Information Security*, 5(2):100–110, 2011.

NRS11. Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptol.*, 24(2):292–321, 2011.

PGMP19. Thomas Prest, Dahmun Goudarzi, Ange Martinelli, and Alain Passelègue. Unifying leakage models on a rényi day. In *CRYPTO (1)*, volume 11692 of *Lecture Notes in Computer Science*, pages 683–712. Springer, 2019.

PR13. Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159. Springer, 2013.

PSG16. Romain Poussier, François-Xavier Standaert, and Vincent Grosso. Simple key enumeration (and rank estimation) using histograms: An integrated approach. In *CHES*, volume 9813 of *Lecture Notes in Computer Science*, pages 61–81. Springer, 2016.

SA08. François-Xavier Standaert and Cédric Archambeau. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2008.

SLP05. Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In *CHES*, volume 3659 of *Lecture Notes in Computer Science*, pages 30–46. Springer, 2005.

SMY09. François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2009.

UHS+18. Florian Unterstein, Johann Heyszl, Fabrizio De Santis, Robert Specht, and Georg Sigl. High-resolution EM attacks against leakage-resilient prfs explained - and an improved construction. In *CT-RSA*, volume 10808 of *Lecture Notes in Computer Science*, pages 413–434. Springer, 2018.

UHSS17. Florian Unterstein, Johann Heyszl, Fabrizio De Santis, and Robert Specht. Dissecting leakage resilient prfs with multivariate localized EM attacks - A practical security evaluation on FPGA. In *COSADE*, volume 10348 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2017.

VGS14. Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In *ASIACRYPT (1)*, volume 8873 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2014.

VMKS12. Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 740–757. Springer, 2012.