Security Analysis of Deterministic Re-Keying with Masking & Shuffling: Application to ISAP

Balazs Udvarhelyi, Olivier Bronchain, and François-Xavier Standaert

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

Abstract. Single-trace side-channel attacks are important attack vectors against the security of authenticated encryption schemes relying on an internal re-keying process, such as the NIST Lightweight Cryptography finalist ISAP. In a recent work of Kannwischer et al., it was suggested to mitigate such single-trace attacks with masking and shuffling. In this work, we first show that combining masking and re-keying is conceptually useless since this combination can always be attacked with a complexity that is just the sum of the complexities to attack a masked implementation (without re-keying) and a re-keyed implementation (without masking). We then show that combining shuffling and re-keying is theoretically founded but can be practically challenging: in low-cost embedded devices (e.g., ARM Cortex-M0) that are the typical targets of single-trace attacks, the noise level of the leakages is such that multivariate attacks can be powerful enough to recover the shuffling permutation in one trace. This second result does not prevent the shuffling + re-keying combination to be effective in more noisy contexts, but it suggests that the best use cases for leakage-resilient PRFs as used by ISAP remain the ones where no additional countermeasures are needed.

1 Introduction

ISAP [10,9] is an authenticated encryption scheme submitted to the NIST Lightweight Cryptography Standardization Process.[1] It comes with claims of improved resistance against side-channel attacks thanks to leakage-resilient features. Precisely, it embeds a re-keying process that mixes a long-term key with public data (e.g., nonces) at a low rate, which can be viewed as a permutation-based variant of the tree-based leakage-resilient PRF constructions discussed in [11,24,13]. The main underlying idea of this construction is that it allows reducing the need to resist against Differential Power Analysis (DPA) to the need to resist Simple Power Analysis (SPA).[2] Since ISAP's re-keying is quite expensive, this idea is then used sparsely (i.e., for initialization and finalization only), in the spirit of a leveled implementations [20]. Concretely, the relevance of this design therefore highly depends on the difficulty to prevent SPA.

[1] https://csrc.nist.gov/projects/lightweight-cryptography.

[2] By DPA (resp., SPA), we mean side-channel attacks where the adversary can observe the leakage of many (resp., a few) different inputs of the leaking primitive.

In two recent and independent works, it has been demonstrated that performing SPA against a permutation-based re-keying is possible on low-end embedded devices [3,18]. In both cases, advanced analytical side-channel attacks like [27] are especially effective because the adversary can average her measurements in order to obtain a strong (noise-free) side-channel signal. In the CHES 2020 paper by Kannwischer et al., it is therefore suggested that SPA security on such low-end devices could be obtained by combining the re-keying of the ISAP design with algorithmic-level countermeasures like masking, which has been applied to the Keccak permutation in [14], or shuffling [16,28], for which the application to bitslice permutation-based designs remains to be investigated.

Combining countermeasures is a popular idea in the side-channel literature. In general, the hope is that the complexity to attack a combined countermeasure will be the product of the complexities to attack its components. For example, in case side-channel measurements are sufficiently noisy, it is known that such a multiplicative effect happens when combining masking and shuffling [21]. In this paper, we question whether the same multiplicative effect takes place when combining re-keying with masking or shuffling, as proposed in [18].

For masking, we answer the question negatively in a definitive manner by showing that a divide-and-conquer side-channel attack of its combination with re-keying is always possible. That is, the complexity to attack a masked leakage-resilient PRF is only the sum of the complexities to (1) extract the useful signal of its masked state and (2) exploit this useful signal in an analytical attack. The latter can be explained by the fact that the useful signal of a masked state is the same as the useful signal of an unprotected state since masking can only amplify the noise of an implementation.[3] In other words, and independent of the level of noise in the measurements, combining re-keying with masking will never lead to a multiplicative effect. At best, the masked implementation can become hard to attack. But in this case, the significant overheads of the re-keying scheme (which iterates the permutation n times to digest an n-bit value) becomes a waste, since this re-keying does not bring any significant additional security benefit.

For shuffling, the situation is different since it is known that its combination with a leakage-resilient PRF is at least theoretically founded [15]. Intuitively, the reason is that in case of sufficient noise, the shuffling countermeasure is modifying the shape of the signal since it emulates a parallel implementation that would combine (e.g., sum) the deterministic parts of multiple byte's leakage functions into a single leakage sample. So the security of this proposal boils down to the question whether the noise of a low-end embedded device such as considered in [3,18] is always sufficient for this emulation to take place. We answer the question negatively by describing a multivariate attack against a shuffled implementation of Keccak in an ARM Cortex M0. We show that we can

[3] Concretely, it could even make the situation worse since the computational overheads of some masked computations (e.g., multiplications) could even increase the signal, which we do not investigate since quite implementation-specific and leading to the same conclusion that masking and re-keying do not combine well.

recover its permutation in a single trace. It implies that a trivial side-channel dissection (in the sense of [6]) of the shuffling + re-keying combination is possible, leading to the same powerful single-trace attacks as without shuffling. So while shuffling ISAP in a more noisy device remains a good strategy in order to prevent averaging traces, it is not a sufficient one to gain high confidence in low-end embedded devices where shuffling can be the target of highly multivariate attacks that further circumvent the already low noise available on such devices.

So overall, our results mitigate the hope that countermeasures aiming to amplify the side-channel noise or to limit the side-channel signal always combine well when a leakage-resilient authenticated encryption scheme like ISAP is implemented on a low-end embedded software platform. They rather suggest that the best use cases for such schemes remain the ones where no additional countermeasures are needed, like larger parallel hardware implementations.

Besides, for designers aiming at securing low-end Cortex-like microcontrollers with a leakage-resilient primitive, relying on AES coprocessors (when available) is currently a better option. In terms of security, such coprocessors inevitably leak less than a software implementation. In terms of performances, they are faster. We refer to [25,5] for two recent examples in this direction.

Related work. To some extent, our conclusion regarding masking and re-keying could be inferred from a previous work of Belaïd et al. [2]. It concluded that the cost vs. security trade-off of a standalone leakage-resilient primitive is better than its combination with masking when the leakage is sufficiently bounded, and that masking alone is preferable otherwise. We consolidate this conclusion by exhibiting the poor (additive) combination of complexities that such a mix implies in general. As for shuffling, it is also known that multivariate attacks can be quite damaging against them and the analysis in [15] was coming with a cautionary note in this direction. Yet, our results show the sensitivity of such security evaluations to small variations of the attack methodology. In particular, the main addition that we made compared to this previous analysis is to extract more information thanks to a dimensionality reduction step [22].

We finally note that we take the ISAP scheme as a case study, but our conclusions also apply to the application of single-trace attacks in the context of post-quantum cryptographic primitives investigated in [18].

2 Background

We next describe necessary background for the rest of the paper. We start by describing the notations, follow with reminders about template attacks (in linear subspaces) and conclude with the description of the two countermeasures we study, masking and shuffling, along with attacks against them.

2.1 Notations

A random variable is expressed with a capital letter and its realisation with a lower case letter such that x is a realisation of X. When clear from the context,

we use the shortcut notation of x for $X = x$. In the context of shuffling, we denote random vectors with bold letters such that $\boldsymbol{x}$ is a realisation of the random vector $\boldsymbol{X}$. Vectors can be indexed with subscripts such that x_i is the i-th element in the vector $\boldsymbol{x}$. In the context of masking, the shares are denoted with superscripts such that x^i is the i-th share of x. The measured side-channel leakages are realisations of random vectors that we always denote $\boldsymbol{l}$. We use the subscript to distinguish the leakage source. For example, $\boldsymbol{l}_x$ is the leakage generated by the manipulation of x and $\boldsymbol{l}_{\boldsymbol{x}}$ is the leakage vector originated from the vector $\boldsymbol{x}$.

2.2 Profiled template attacks

A profiled template attack is performed in 2 steps. The first one is called profiling phase. There, the adversary constructs an estimation of the *Probability Density Function* (PDF) of the leakages $\boldsymbol{l}$ conditioned on a secret variable x. In this work, we will use Gaussian template attacks in a linear subspace [8,22]. It is similar to Gaussian template attacks with a preliminary linear projection of the leakage samples of length n to a n'-dimensional subspace with $n' \leq n$. Formally, the adversaries we consider build a PDF estimation of the form

$$\hat{\mathsf{f}}(\boldsymbol{l}|x) = \frac{1}{\sqrt{(2\pi)^{n'} \cdot |\boldsymbol{\Sigma}|}} \cdot \exp^{\frac{1}{2}(\boldsymbol{W}\boldsymbol{l}-\boldsymbol{\mu}_x)\boldsymbol{\Sigma}(\boldsymbol{W}\boldsymbol{l}-\boldsymbol{\mu}_x)'}, \tag{1}$$

where $\boldsymbol{l}$ is a leakage vector of length n, $\boldsymbol{W}$ is a linear projection matrix of size $n' \times n$, $\boldsymbol{\mu}_x$ a mean vector of length n' and $\boldsymbol{\Sigma}$ a covariance matrix of size $n' \times n'$. The profiling consists in estimating a projection matrix $\boldsymbol{W}$ with Linear Discriminant Analysis (LDA), the covariance matrix $\boldsymbol{\Sigma}$ and the means $\boldsymbol{\mu}_x$ for all x.[4]

The second step in profiled attacks is to leverage the PDF estimation to recover a secret realisation x from leakage observations $\boldsymbol{l}$. Namely, based on the estimated PDFs, the adversary applies Bayes's rule such that

$$\hat{\mathsf{p}}(x|\boldsymbol{l}) = \frac{\hat{\mathsf{f}}(\boldsymbol{l}|x)}{\sum_{x^* \in X} \hat{\mathsf{f}}(\boldsymbol{l}|x^*)} \cdot \tag{2}$$

Based on the observed leakage $\boldsymbol{l}$, the adversary will guess the value of x as

$$\hat{x} = \underset{x^*}{\operatorname{argmax}} \quad \hat{\mathsf{p}}(x^*|\boldsymbol{l}). \tag{3}$$

The adversary can optionally combine multiple leakage observations in order to recover a long-term secret such as an encryption key. To do so, she calculates the likelihoods of each possible secret by multiplying the $\hat{\mathsf{p}}(x|\boldsymbol{l})$ obtained with Equation (2) for multiple leakages.[5]

[4] If $\boldsymbol{W}$ is the identity, this is equivalent to standard Gaussian templates attacks [8].

[5] The realisation x may not always be a long term secret. For example, when targeting a block cipher, x is usually an intermediate value that is bijectively mapped to a secret key byte k with the relation $x = \mathrm{Sbox}(k \oplus p)$, with p a public plaintext.

In order to evaluate our attacks, we will use the success rate (SR) as metric [23]. It is the probability that the adversary recovers the correct value of the secret variable, which we denote as

$$\mathrm{SR}_x = \Pr[x = \hat{x}], \tag{4}$$

where the subscript notation defines the target variable (in this case, x).

2.3 Masking countermeasure

Masking is a popular countermeasure against side-channel attacks. It consists in randomizing the manipulated data by replacing x by d random shares x^i. The shares are uniformly distributed and ensure that $x = \sum_{i=1}^{d} x^i$. Thus, any combination of $d-1$ shares remains independent of x which corresponds to the so-called *d-probing security* [17]. In order to maintain this property during the entire computation, the linear operations can be applied straightforwardly in a share-by-share fashion, which has a cost that is linear in d. Non-linear operations (e.g., multiplications) are more challenging and require to mix shares, which implies heavier overheads and randomness. Since our following investigations hold already at the encoding level, we do not detail these multiplications.

In order to attack a masked (software) implementation, an adversary can first perform an attack on each of the shares individually as in the unprotected case from subsection 2.2 to obtain $\hat{\mathsf{p}}(x^i|\boldsymbol{l})$ for all the x^i. The probability of the shared secret x is then given by

$$\hat{\mathsf{p}}(x|\boldsymbol{l}) \propto \sum_{\{x^0,\ldots.x^{d-1}\} \in \mathcal{X}^{d-1}} \prod_{i=1}^{d} \hat{\mathsf{p}}(x^i|\boldsymbol{l}). \tag{5}$$

Informally, this equation shows the interest of masking that is "multiplying" the uncertainty (or noise) of the different shares. This results in the attack complexity growing exponentially in d [7]. We note that in practice, this (exponential) guarantee only holds under the assumptions that measurements are sufficiently noisy and that leakages are a linear combination of shares [12].

2.4 Shuffling countermeasure

Shuffling is another popular side-channel countermeasure. While masking randomizes the manipulated data, shuffling randomizes the execution flow. Namely, when an algorithm is composed of independent operations, these can be executed in a random order while maintaining correctness. One typical application of shuffling is an Sbox layer applied to an input vector $\boldsymbol{x}$ of size $|\boldsymbol{x}|$ (e.g., 16 for the AES). Next we detail such an application of shuffling thanks to Algorithm 1. There, the first step is to generate a random uniform permutation $\boldsymbol{\pi}$ over of the set $\{0, 1, \ldots, |\mathbf{x}| - 1\}$. This is done with gen_perm$(\cdot, \cdot)$ that takes as input the

permutation size together with randomness $\mathcal{R}$.[6] The second step is to iterate over all the indexes i with $0 \leq i < |\boldsymbol{x}|$. At every iteration, one value $j = \pi_i$ is fetched from the permutation $\boldsymbol{\pi}$. The Sbox is then applied to j-th entry of the input vector and stored at the corresponding index of the output vector.

Algorithm 1 Shuffled Sbox layer.

Input: $\boldsymbol{x}$ and randomness $\mathcal{R}$.
Output: $\boldsymbol{y}$ such that $\forall i, 0 \leq i < |\boldsymbol{x}|, y_i = \text{Sbox}(x_i)$

1: $\boldsymbol{\pi} \leftarrow \text{gen\_perm}(\mathcal{R}, |\text{x}|)$
2: **for** i in $\{0, 1, ..., |\mathbf{x}| - 1\}$ **do**
3: $\quad j \leftarrow \pi_i$
4: $\quad y_j \leftarrow \text{Sbox}(x_j)$

Informally, in order to perform a side-channel attack against a shuffled implementation, an adversary has to map the leakage of x_j to the correct x_i for every iteration. If the permutation is known by the adversary, such an implementation is equivalent to an unprotected one. If it is not, the adversary is forced to perform a so called "integration" attack which sums together every $\boldsymbol{l}_{x_j}$ (resp., $\boldsymbol{l}_{y_j}$) [28]. This has the effect of turning the leakage of a serial implementation (e.g., software) into the leakage of a parallel one (e.g., hardware) where the adversary has only access to the sum of all leakages. In practical case studies, the permutation generally leaks partially to the adversary through $\boldsymbol{l}_{\pi_i}$. These leakages can be due to the generation of the permutation π itself, to its storage and loading from memory and to the addresses used to load x_j and to store y_j. We analyse template attacks exploiting such leakages in section 4.

3 Re-keying + masking

In order to break the re-keying scheme of ISAP with an SPA, the first step is to recover a maximum of information on each of the intermediate variables. To do so, the adversary is allowed to blend measurements, meaning that she can observe multiple leakages for the same secret input. The second step is to recombine this partial information on the intermediate variable to obtain a key guess [27,18,3]. Next, we study the case where masking is combined with re-keying.

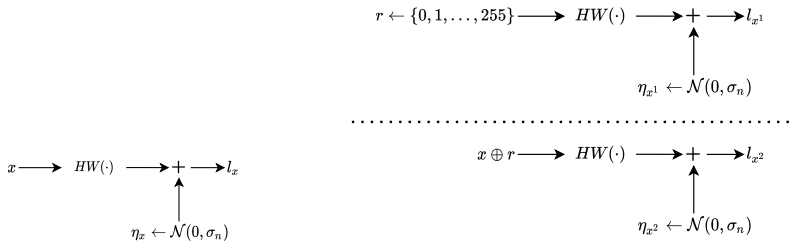
Since our goal is to show that re-keying and masking generically combine badly, we perform our investigations in a simulated setting where the noise level is tightly controlled. This allows us to illustrate our claims in an easily interpretable context and to show that they hold even in case masking is perfectly implemented (e.g., without independence flaws). We next present the parameters selected for our simulations and then discuss the obtained results.

[6] In this work, we assume that $\text{gen\_perm}(\cdot, \cdot)$ is pre-computed and the permutation is stored in memory. It can also be generated on-the-fly if needed.

3.1 Simulation settings

For our simulations, the leakage of a given variable is its Hamming weight (HW) with additional Gaussian noise (as previously considered in the simulated setting of [18]). We consider different noise levels σ_n^2, such that the Signal to Noise Ratio (SNR) of our leakages corresponds to 0.1, 1 and 10 [19]. Our simulation settings are summarized in Figure 1, where the sensitive variable x is an 8-bit word. For the unprotected setting (see Figure 1a), the leakage of each variable x is written as l_x. For the first-order masking (see Figure 1b), the leakage on both shares x^1 and x^2 are respectively denoted as l_{x^1} and l_{x^2}.

We performed our experiments 1000 times to average the results. For each experiment, we used 5000 measurements of the leakages l_x, l_{x^1} and l_{x^2}.



(a) Unmasked implementation. (b) Masked implementation.

Fig. 1: Simulation settings with Hamming weight leakages and Gaussian noise.

3.2 Security analysis

In the following, we will first detail how the adversary can blend measurements in both the masked and unmasked settings. Then, we detail what is her SR in recovering an intermediate variable x first for unmasked and then for masked implementation. We will show that their asymptotic SR are equivalent, making the effect of masking and shuffling independent for the adversary.

In order to blend independent measurements, and to exploit multiple traces that manipulate the same secret x, the adversary can use the equation

$$\hat{\mathsf{p}}'(x|\boldsymbol{l}) = \prod_{i=0}^{t-1} \hat{\mathsf{p}}(x|\boldsymbol{l}^i), \tag{6}$$

where $\boldsymbol{l}^i$ is one of the t traces to combine and $\boldsymbol{l}$ is their concatenation. In the unmasked case, the adversary can simply use Equation (2) for $\hat{\mathsf{p}}(x|\boldsymbol{l}^i)$. In the masked case, Equation (5) has to be used.

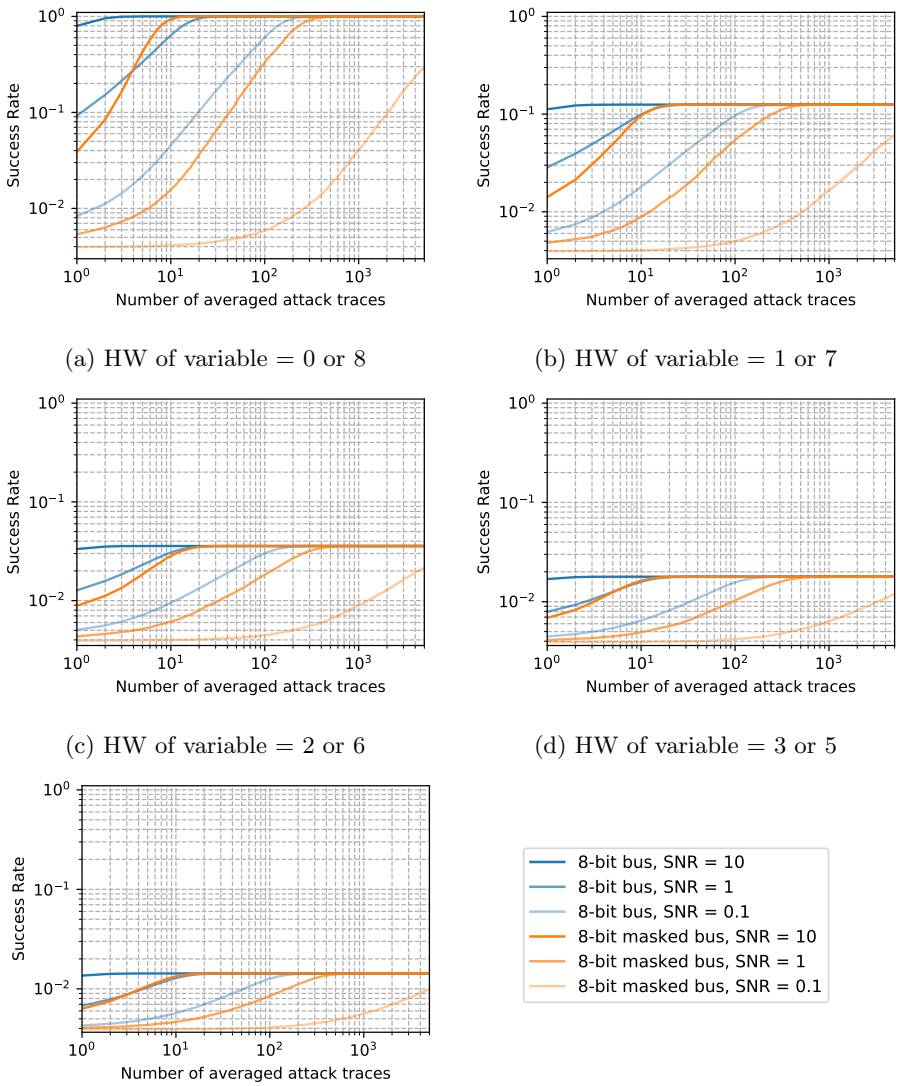
Unmasked implementation. In the unmasked setting, computing Equation (6) is equivalent to directly averaging all the $\boldsymbol{l}^i$'s. It allows recovering the mean of the leakages μ_x, which is $\mathrm{HW}(x)$ in our simulation since it averages the additive Gaussian noise η (see Figure 1a). By averaging, the adversary can increase the SNR and have a tighter approximation of $\mathrm{HW}(x)$.

The success rate of this adversary is reported on the blue curves of Figure 2 for different Hamming weights and SNRs. Since we focus on a SPA setting, the amount of information that can be extracted from the leakage (and therefore the SR) indeed depends on this Hamming weight. The x-axis corresponds the number of averaged traces t, the y-axis is the SR_x and the different plots are for different $\mathrm{HW}(x)$. We observe that: (i) the SR_x first increases with t and then saturates: this is because the noise is averaged and so the estimate of leakage mean becomes more accurate up to the point where there is no noise left; (ii) lowering the SNR (i.e., adding noise), slows down the convergence of SR_x but does not affect its asymptotic value: this is because the asymptotic value of SR_x only depends on the side-channel signal (i.e., the deterministic part of the – here Hamming weight – leakage function); (iii) this asymptotic value is larger for extreme Hamming weights (e.g., it is worth 1 for the Hamming weights 0 and 8) and lower for the intermediate Hamming weights: this is because multiple values of x then lead to the same HW, making it impossible for the adversary to recover it with probability one. More precisely, SR_x is inversely proportional to the number of values with the same HW. For example, when $\mathrm{HW}(x) = 1$ (or $\mathrm{HW}(x) = 7$), the asymptotic SR_x is equal to $\mathrm{SR}_x = 1/8 = 0.125$ as we count 8 values on an 8-bit bus that have $\mathrm{HW}(x) = 1$ (resp., $\mathrm{HW}(x) = 7$).

Masked implementation. We now observe that moving to a masked implementation leads to essentially similar observations. For this purpose, we first report the histograms of the two-dimensional leakages corresponding to a masked encoding with two shares in Figure 3. The x-axis corresponds to the leakage on the first share (l_{x^1}) and the y-axis corresponds to the leakage on the second share (l_{x^2}). Each subplot is for a different Hamming weight. Clearly, we observe that just as in the unprotected case where the adversary could distinguish 9 distributions corresponding to the 9 Hamming weights, we still have 9 different distributions in the masked case. As a result, the side-channel signal that can be obtained when masking is the same as in the case of an unprotected implementation (up to the comment of Footnote 3) and this actually holds independently of the leakage function: masking amplifies the noise but does not affect the signal.

This fact is directly reflected in the orange SR_x curves of the masked implementation that are reported in Figure 2. On the one hand, the asymptotic values are equal to the ones of the unprotected case. On the other hand, converging to this asymptotic value (i.e., extracting all the signal) will need more measurements as expected from masking security proofs [12].

As a result, attacking a combination of masking and re-keying can always proceed in two steps with additive complexities. The first one is the attack against all intermediate values x to recover partial information. The number of traces t



(a) HW of variable = 0 or 8

(b) HW of variable = 1 or 7

(c) HW of variable = 2 or 6

(d) HW of variable = 3 or 5

(e) HW of variable = 4

(f) Legend

Fig. 2: Success rate of a value-recovery attack against unprotected and masked implementations at different noise levels and for different target values.

required for each of them is impacted by masking but not the amount of signal that can be recovered asymptotically. The second one is the attack that recombines all these partial informations with SASCA, which remains unchanged. So no multiplicative effect takes place in this combination of countermeasures which can always be broken with a divide-and-conquer approach. As mentioned in in-

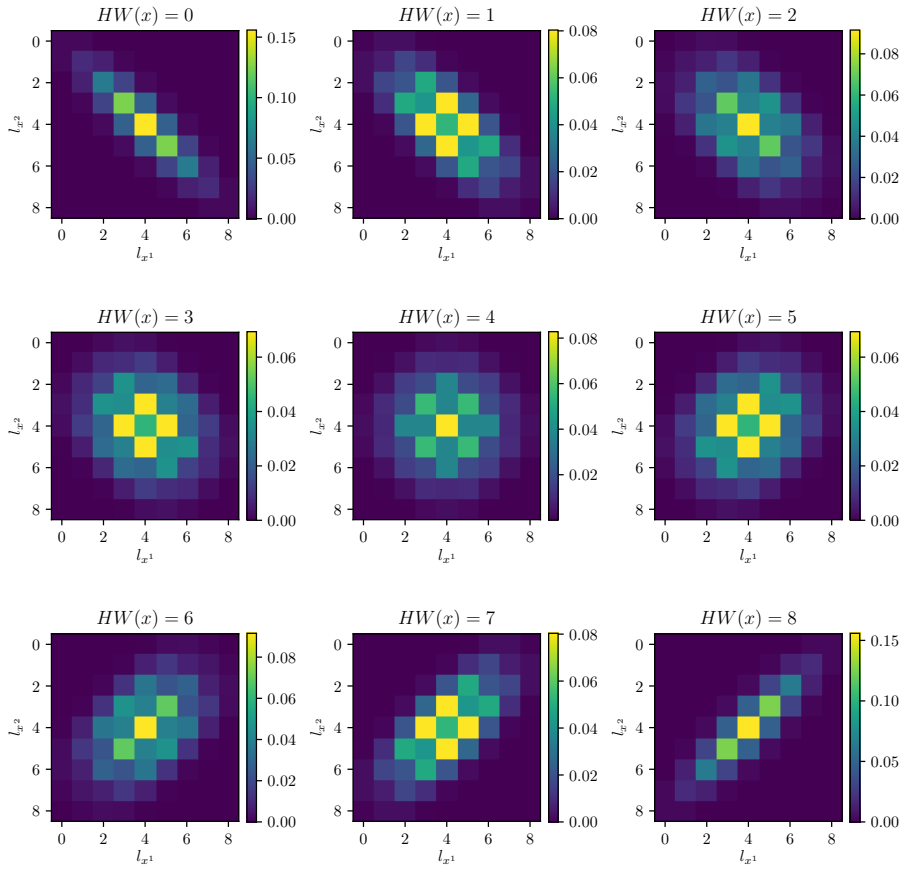


Fig. 3: Masked PDFs $\mathsf{f}_{\mathsf{x}}(\boldsymbol{l}_{\boldsymbol{x}^1}, \boldsymbol{l}_{\boldsymbol{x}^2} | \boldsymbol{x})$ for different $\mathrm{HW}(\boldsymbol{x})$ values and SNR 10.

troduction, this result is expected since the security of a PRF-based re-keying scheme depends on the amount of side-channel signal that a leaking implementation provides. Therefore, a countermeasure aiming at amplifying the noise cannot be of any help to satisfy this assumption: it can only make the signal extraction more difficult, as would be provided by masking used as a stand-alone countermeasure. This conclusion holds for any number of shares.

4 Re-keying + shuffling

Since combining re-keying with masking cannot lead to a strong (multiplicative) impact on the security of an implementation, we now consider shuffling as an alternative option. The situation is different in this case since in order to make the shuffling ineffective, the adversary has to recover the permutation $\boldsymbol{\pi}$ that is used

to randomize the execution of the underlying operations in a single measurement. There is no possibility to combine multiple leakages for this part of the attack since the permutation is an ephemeral secret. If she succeeds, the situation is similar to an unprotected setting and the leakage of x can be averaged again. If not, the shuffling affects the shape of the leakage function, ultimately emulating a parallel implementation where the shuffled operations cannot be distinguished. While this ideal situation is not expected to happen [28], the question we tackle in this section is whether shuffling always adds some complexity to the attack. We next show that in the case of a low-noise MCU (e.g., an ARM Cortex M0), the leakage may be enough to recover the whole permutation with a single trace, making the combination of shuffling and re-keying irrelevant.

4.1 Implementation and measurement setup

The previous published attacks against re-keying schemes focused on the Keccak permutation [18,4] and its usage in the ISAP authenticated encryption scheme [3]. Next, we describe both our software implementation and the measurements setup that we keep as close to the one used in [3] as possible.

Regarding the software, we modified the reference implementation of ISAP [1]. We leveraged the shuffling with double indexing from Algorithm 1 and modified the Keccak implementation to combine shuffling and re-keying. More precisely, since a Keccak state contains 25 *lanes*, we generate permutations $\boldsymbol{\pi}$ with 25 elements and shuffle 25 independent operations.[7] The permutations are generated before the re-keying process and stored in memory. Therefore during the attack, the leakages $\boldsymbol{l}_{\pi_i}$ on the permutation indices π_i is independent of the permutation generation. They only result from the memory loadings and from other bijectively related leaking variables (e.g., memory addresses).

Regarding the measurement setup, our implementation is running on an STM32F0308 Discovery board, based on an ARM Cortex M0 as in [3]. An additional crystal was added to the board to provide stable a clock source for the side-channel measurements. Decoupling capacitors were also removed. We measured our traces thanks to the Tektronix CT-1 AC current probe and a Picoscope 5244D oscilloscope. The clock frequency of our MCU was set to the maximum value of 48MHz and we sampled at 500 MSamples/s.

4.2 Leakage modeling

In order to recover the permutation indices π_i, our adversary needs to obtain probabilities $\hat{\mathsf{p}}(\pi_i|\boldsymbol{l})$. To do so, she uses the templates from subsection 2.2. More precisely, she first computes the SNR for each permutation index π_i [19]. Then, she selects the Points-Of-Interest for each of them by only considering samples

[7] It is not always possible to find 25 independent operations within the Keccak round function. Yet, we will show that even in this best case (for the designer) where there are 25 independent operations, shuffling is ineffective.

above the noise floor. On average, we used $n = 450$ samples for each permutation index. For each of them, we then estimated a PDF with Equation (1), with the number of subspace dimensions n' as a parameter.

4.3 Permutation index recovery

We first focus on an adversary attempting to recover each of the permutation indexes π_i within the permutation $\boldsymbol{\pi}$ independently, next denoted as $\mathcal{A}_1$. To do so, she uses the previously described templates. Namely, with a single leakage observation $\boldsymbol{l}$, she assumes a value $\hat{\pi}_i$ for the permutation index π_i with Equation (3). On Figure 4, we present the average SR_{π_i} of the first permutation within the shuffling implementation of ISAP described in subsection 4.1. We observe that by increasing the number of dimensions in the linear subspace n', the SR_{π_i} increases. Its maximum is of 99.18% with $n' = 15$ when all our 20,000 profiling traces are used. We can also see that the use of 5000 profiling traces and $n' = 7$ is enough to obtain the asymptotic SR values.

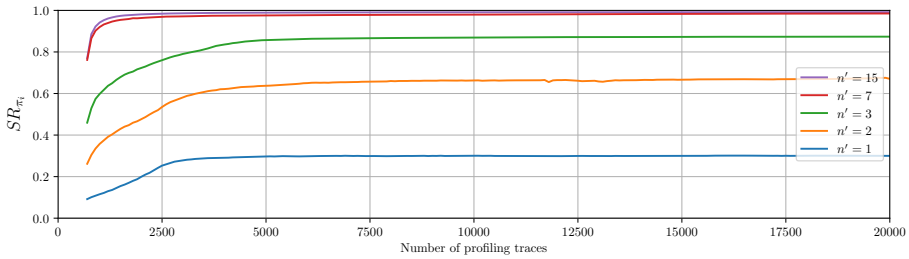


Fig. 4: Estimated SR_{π_i} for $1 \leq n' \leq 15$ and various number of profiling traces.

4.4 Full permutation recovery

We now discuss how to turn the information about the permutation indexes π_i into a full permutation recovery. To do so, we introduce the corresponding success rate $\mathrm{SR}_{\boldsymbol{\pi}}$ that is defined as

$$\mathrm{SR}_{\boldsymbol{\pi}} = \Pr[\pi_i = \hat{\pi}_i,\ \forall i]. \tag{7}$$

A first intuitive solution, denoted $\mathcal{A}_2$, is to derive an estimate for the full permutation $\hat{\boldsymbol{\pi}}$ by leveraging the $\mathcal{A}_1$ adversary and directly plugging the obtained π_i within $\hat{\boldsymbol{\pi}}$. The SR of these two adversaries are linked as $\mathrm{SR}_{\boldsymbol{\pi}} = \prod_i \mathrm{SR}_{\pi_i}$.

From this, we observe that even though the observed SR_{π_i} in Figure 4 is close to 100%, it is not sufficient to recover the full permutation with overwhelming probability. Namely we have that $\mathrm{SR}_{\boldsymbol{\pi}} \approx 0.9918^{25} \approx 0.8139$.

In order to improve these results, the adversary $\mathcal{A}_3$ that we describe in Algorithm 2 takes advantage of a characteristic of permutations which are under attack. Namely, $\forall i, j$ such that $i \neq j$ we have $\pi_i \neq \pi_j$ and this constraint is not enforced with the straightforward divide & conquer adversary $\mathcal{A}_2$. A simple option to exploit this constraint is to enumerate the $\hat{\boldsymbol{\pi}}$'s from the most to the least probable, and to keep as guess the first one being a permutation.

Algorithm 2 Permutation adversary $\mathcal{A}_3$.

Input: Lists for probabilities $\hat{\mathrm{p}}(\pi_i|\boldsymbol{l})$ $\forall i$ and $\forall \pi_i$.
Output: Permutation guess $\hat{\boldsymbol{\pi}}$ with $\hat{\pi}_i \neq \hat{\pi}_j$ $\forall i, j$ and $i \neq j$.

1: **for** $\hat{\boldsymbol{\pi}} \leftarrow \mathrm{Enumerate}(\hat{\mathrm{p}}(\pi_0|\boldsymbol{l}), \hat{\mathrm{p}}(\pi_1|\boldsymbol{l}), \ldots)$ **do**
2: **if** $\hat{\pi}_i \neq \hat{\pi}_j$ $\forall i, j$ and $i \neq j$ **then**
3: **return** $\hat{\boldsymbol{\pi}}$

Various enumeration algorithms exist in the literature: $\mathcal{A}_3$ takes the key enumeration algorithm of [26] to instantiate $\mathrm{Enumerate}(\cdot)$, which has the advantage to be optimal. The $\mathrm{SR}_{\boldsymbol{\pi}}$ of this adversary is reported in Figure 5 where the y-axis is $1 - \mathrm{SR}_{\boldsymbol{\pi}}$ in log-scale. The x-axis is the maximum number of permutations enumerated (i.e., the maximum number iterations of the loop) in Algorithm 2. The dashed horizontal lines are for a single iteration and correspond to $\mathcal{A}_2$. First, we observe that by enabling more enumeration steps, the SR increases. Second, we observe that the SR also increases with the number of dimensions n' in the template's linear subspace. Putting things together, for the best templates ($n' = 15$), we observe that $\mathrm{SR}_{\boldsymbol{\pi}}$ starts from 0.814 if no enumeration is allowed and increases up to 0.995 when setting the maximum number of steps to 20. Enumerating more was useless in our case, since with high probability, a permutation was found with at most 20 steps. Hence, the computational cost of the enumeration is negligible compared to the rest of the attack.

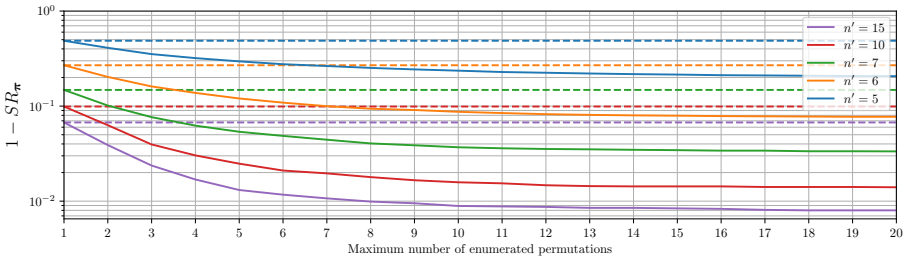


Fig. 5: $1 - \mathrm{SR}_{\boldsymbol{\pi}}$ for different subspace dimensions n' and as function of the enumeration depth. Dashed lines represent the value without enumeration.

Our experiments therefore illustrate that recovering a leaking permutation on the considered low-end platform is possible with low complexity. An adversary can then cancel the impact of the shuffling countermeasure, making analytical attacks against re-keying like [18,3] possible. We finally note that various other attacks could be applied against the shuffling and refer to [28] for a survey.

5 Conclusions

Single-trace attacks are an important threat against leakage-resilient PRFs such as used in the ISAP authenticated encryption scheme. A recent work of Kannwischer et al. suggested to avoid them by combining the re-keying scheme that leakage-resilient PRFs leverage with masking or shuffling. We first showed that combining such PRFs with masking is in general not a good idea as it just adds up the complexities to attack the two countermeasures separately. We then showed that combining these PRFs with shuffling, while theoretically appealing, can be practically challenging. For low-end embedded devices that are typical targets for such a combination (since they correspond to the targets of the single-trace attacks in [18,3]), implementing shuffling securely requires preventing multivariate side-channel attacks aiming at recovering the shuffling permutation (which are easy in low-noise settings). This second conclusion is not a general claim and shuffling could be effectively combined with re-keying on more noisy targets. But it shows that preventing single-trace attacks on low-end devices for which single-trace attacks are a concern is not trivial. Whether this can be achieved on the Cortex M0 device we analyzed is an interesting open question, which raises the challenge of hiding its leaky load/store operations. Overall, our results suggest that ISAP may not be the best option to ensure side-channel security on low-end devices and finds a more natural application to more parallel (possibly hardware) architectures. If aiming at exploiting a leakage-resilient PRF, leveraging the AES co-processors available on many Cortex-like chips seems more adequate, since they leak less and are faster [25,5].

Acknowledgments. François-Xavier Standaert is a senior research associate of the Belgian Fund for Scientific Research (F.R.S.-FNRS). This work has been funded in parts by the European Union through the ERC project SWORD.

References

1. ISAP code package. `https://github.com/isap-lwc/isap-code-package`. Accessed: 2021-03-10.
2. Sonia Belaïd, Vincent Grosso, and François-Xavier Standaert. Masking and leakage-resilient primitives: One, the other(s) or both? *Cryptogr. Commun.*, 7(1):163–184, 2015.
3. Davide Bellizia, Olivier Bronchain, Gaëtan Cassiers, Vincent Grosso, Chun Guo, Charles Momin, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Mode-level vs. implementation-level physical security in symmetric cryptography - A practical guide through the leakage-resistance jungle. In *CRYPTO (1)*, volume 12170 of *Lecture Notes in Computer Science*, pages 369–400. Springer, 2020.

4. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The KECCAK reference. `https://keccak.team/files/Keccak-reference-3.0.pdf`.
5. 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):XXX–XXX, 2021.
6. Olivier Bronchain and François-Xavier Standaert. Side-channel countermeasures' dissection and the limits of closed source security evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):1–25, 2020.
7. 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.
8. 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.
9. Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, Bart Mennink, Robert Primas, and Thomas Unterluggauer. ISAP v2.0. *IACR Trans. Symmetric Cryptol.*, 2020(S1):390–416, 2020.
10. Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, and Thomas Unterluggauer. ISAP - towards side-channel secure authenticated encryption. *IACR Trans. Symmetric Cryptol.*, 2017(1):80–105, 2017.
11. Yevgeniy Dodis and Krzysztof Pietrzak. Leakage-resilient pseudorandom functions and side-channel attacks on Feistel networks. In *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 21–40. Springer, 2010.
12. 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.
13. Sebastian Faust, Krzysztof Pietrzak, and Joachim Schipper. Practical leakage-resilient symmetric cryptography. In *CHES*, volume 7428 of *Lecture Notes in Computer Science*, pages 213–232. Springer, 2012.
14. Hannes Groß, David Schaffenrath, and Stefan Mangard. Higher-order side-channel protected implementations of KECCAK. In *DSD*, pages 205–212. IEEE Computer Society, 2017.
15. Vincent Grosso, Romain Poussier, François-Xavier Standaert, and Lubos Gaspar. Combining leakage-resilient PRFs and shuffling - towards bounded security for small embedded devices. In *CARDIS*, volume 8968 of *Lecture Notes in Computer Science*, pages 122–136. Springer, 2014.
16. 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.
17. 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.
18. Matthias J. Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on Keccak. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):243–268, 2020.
19. 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.
20. Olivier Pereira, François-Xavier Standaert, and Srinivas Vivek. Leakage-resilient authentication and encryption from symmetric cryptographic primitives. In *CCS*, pages 96–108. ACM, 2015.

21. Matthieu Rivain, Emmanuel Prouff, and Julien Doget. Higher-order masking and shuffling for software implementations of block ciphers. In *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2009.
22. 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.
23. 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.
24. François-Xavier Standaert, Olivier Pereira, Yu Yu, Jean-Jacques Quisquater, Moti Yung, and Elisabeth Oswald. Leakage resilient cryptography in practice. In *Towards Hardware-Intrinsic Security*, Information Security and Cryptography, pages 99–134. Springer, 2010.
25. Florian Unterstein, Marc Schink, Thomas Schamberger, Lars Tebelmann, Manuel Ilg, and Johann Heyszl. Retrofitting leakage resilient authenticated encryption to microcontrollers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):365–388, 2020.
26. Nicolas Veyrat-Charvillon, Benoît Gérard, Mathieu Renauld, and François-Xavier Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. In *Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 390–406. Springer, 2012.
27. 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.
28. 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.