Electromagnetic Leakage Assessment of a Proven Higher-Order Masking of AES S-Box

Nicolas Bordes, P. Maistri

Electromagnetic Leakage Assessment of a Proven Higher-Order Masking of AES S-Box

Nicolas Bordes*,Paolo Maistri†
*Univ. Grenoble Alpes, CNRS, Grenoble INP[1], LJK, 38000, Grenoble, France
†Univ. Grenoble Alpes, CNRS, Grenoble INP[1], TIMA, 38000 Grenoble, France
{first.last}@univ-grenoble-alpes.fr

***Abstract*—Many digital systems need to provide cryptographic capabilities. A large part of these devices is easily accessible by the malicious user, and may be vulnerable to side channel attacks such as power or electromagnetic analysis. From one side, the designer has to protect the architecture with proven countermeasures; on the other, the actual implementation must be validated in order to prove the absence of undesired leakages. In this paper, we present an implementation of two optimized and proven masking schemes of order 3 and 7 for an embedded software AES, and prove its robustness by showing the absence of significant leakage in the nonlinear layer.**

***Index Terms*—Masking, Side channel, Leakage Assessment.**

I. Introduction

Digital systems often support cryptographic operations and protocols in order to protect sensitive data or communications. As well-designed as a cryptographic algorithm may be, vulnerabilities may appear during its implementation and execution. Side-channel attacks aim to exploit physical variations during the program execution. These physical variations are called side-channels and can be, for example, the time it takes for the program to execute [1], or the power consumption of the circuit [2] but also the electromagnetic radiations [3] emitted during the computations. When these fluctuations depend on secret internal data, an attacker may be able to extract information that would be not available in a model where they only have a "black box" access to the cryptographic function. Even a partial knowledge of the intermediate values of a program implementing cryptographic primitives can be devastating for its security. A recent example is given in [4] where the authors published a practical side-channel attack against Titan security keys: they exploited electromagnetic leakage during the computation of an ECDSA (Elliptic Curve Digital Signature Algorithm) signature to retrieve the full private key, allowing to generate new valid signatures and thus clone the security key.

The attacker usually needs to have physical access to the device in order to do measurements: thus, embedded devices are the most targeted. Additionally, measurements in these devices are often with very little noise, which makes the attack easier on embedded devices than on any other setup.

Embedded devices are the most susceptible to be targeted by side-channel attacks, but they also have hard efficiency and production constraints. These are even stronger than for other devices and they must be taken into account when designing and choosing countermeasures.

Most countermeasures against side-channel attacks aim at limiting the exploitability of the hypothetical signal an attacker could be able to measure from the device and to increase the cost (in time, technical expertise, hardware measurement tool, . . .) of the attack. A first approach is the more obvious one: by reducing the signal strength, it is harder for an attacker to get useful information for it. To do so in practice turns out to be hard: in a context where the attacker has physical access to the device, this countermeasure can often be defeated by a direct intervention of the attacker. Most importantly, this countermeasure is specific for a given side channel: protecting against a wide range of side-channels can be very costly without any guarantee that the attacker will not find a new side-channel to exploit.

Another approach is based on lowering measure reproducibility. In practice, the attacker often must gather several traces in order to reduce measurement noise and improve signal quality. However, these traces must be synchronized: the attacker must be able to know which measurement points in the traces correspond to the same computation. Adding dummy code at random location during runtime makes this synchronization difficult [5], [6], [7]. This approach has its limits: if only one trace is enough to attack the device [8], desynchronization does not help; also, using signal processing and pattern recognition techniques, the points of leakages can be nonetheless identified and one may be able to counteract desynchronization attempts [9], [10].

For this reason, there is a strong need to provide countermeasures that are at the same time efficient, cost-effective, and formally proven to guarantee the absence of vulnerabilities in the actual implementation. The current trend is to use masking schemes of order (i.e., the degrees of random variability) larger than one, as simple masking can still be easily broken. Our goal is therefore to improve the efficiency of masked implementations (in our case, of the Advanced Encryption Standard – AES), especially at higher orders, in the hope that it will help moving from ad hoc and specialized countermeasures against side-channel attacks toward formally secure and generic ones. In this paper, we present an implementation of two optimized and proven masking schemes of order 3 and 7 (in order to compare with similar state of the art) for an embedded software AES, and prove its robustness by showing

[1]OMITTED FOR BLIND REVIEW

the absence of significant leakage in the nonlinear layer.

The paper is organised as follows. In the next section, some theoretical concepts about masking are recalled. Section III deals with the way masking can be actually implemented in embedded software. Section IV describes our implementations choices and the resulting raw performance, while the experimental security assessment against side channel attacks are discussed in Section V. Finally, Section VI concludes the paper.

II. High-order Masking

The concept at the root of masking is to use a secret sharing algorithm to split the sensitive data into multiple shares that are individually statistically independent from the original data. All computations are done on those shares such that what an attacker is observing looks like noise and does not directly depend on the sensitive data. This approach has been introduced simultaneously by Goubin et al. [11] and Chari et al. [12] in 1999.

Masking is said to be at order d when the number of shares is equal to $d + 1$. When using an additive secret sharing scheme, an order d masking is achieved by drawing uniformly at random the first d shares $x_0; ...; x_{d-1}$ and computing the last share x_d such that $x = \oplus_{i=0}^{d} x_i$. This way, every subset made by less than d shares is uniformly distributed.

There is a strong need for formal security models, which are needed to adopt a generic and modular approach to the design of masked implementations based on the composition of smaller secure masked circuits. In particular, these models apply to the implementation, as well as to the capabilities of an attacker. The d-probing model is an attack model where the attacker is given access to up to d probes on the target. An attacker is often not strictly limited by the number of probes, especially on a software implementation where the sequential execution can lead an attacker to observe the leakage of each operations over a whole period of time. On the other hand, an attacker observation always embeds measurement noise.

Additionally, the masking order d plays an important role in the security because, for equivalent noise level, the number of measurements needed for a successful attack increases exponentially in d. Unfortunately, generic high-order schemes also come with a significant overhead as shown in Section V. A detailed and formal analysis of secure compositional models goes beyond the scope of this paper. The interested reader can find more details in [13], [14].

III. Masking in Practice

As explained in Section I, the devices that are the most vulnerable against side-channel attacks are embedded ones. In this context, the processor used often implements the ARM architecture. It is thus natural to consider implementing a masking scheme specifically for this architecture. Following Schwabe and Stoffelen's work [15], we will further focus on one of the most popular modern microprocessor family in this context: the ARM Cortex-M4[1]. This family of microprocessors implements the ARMv7E-M architecture which provides all the Thumb-1, Thumb-2 and Digital Signal Processing (DSP) instructions.

In the 2000's, many attempts have been made to design ad hoc masked implementations and "provably secure" ones. However, many were found to be vulnerable to side-channel attacks a few years after their publication. In the light of those attempts, we will use the more recent compositional security models introduced by Barthe *et al.* [13]. As shown before, these models's goal is specifically turned toward the design of complex masked circuits given only elementary gadgets (i.e., operations).

Since the ultimate goal of masking is to amplify noise during the critical computations by introducing random values, the effect of masking in practice will be demonstrated by comparing the leakages when random gates in the masked circuit are leveraging an embedded TRNG, or rather when they use deterministic values.

A. That's too much to (m)ask: the masking order

The choice of the masking order at which these gates are instantiated is crucial for both security and performance since it has a quadratic impact on both. As recalled in the introduction, however, simple low-order masking is not enough against a skilled attacker.

The most recent and optimized implementations using the same generic approach as ours [16], [17] chose an order such that the number of shares is a power of two. Their choice was made because they are based on a specific implementation approach that provides the best performance when the size of the registers (expressed in bits) is a multiple of the number of shares. For a fair comparison, we chose to also use an order where the number of shares is a power of two. Additionally, we chose to use gadgets that give better performance while still being proved secure [18]. With these constraints in mind, we provide two masked implementations: one at order 3 (four shares) and one at order 7 (eight shares).

Even if recent symmetric block cipher proposals were made that were specifically designed to be masked, they are not yet widely used in concrete applications as of today. We therefore focus on AES, which is used in a wide range of applications, most commonly as a building block of symmetric encryption schemes. Implementations of the AES has been the target of heavy optimization efforts, which are often made to decrease its running time or its memory footprint. This is perfectly acceptable where the physical access to the device is near impossible; for embedded devices, however, the threat model is different and side-channel attacks must not be overlooked. In this section, we present our proposal for a secured embedded implementation of AES against high-order attacks.

B. Masking the AES S-Box

Every step other than SubBytes in the AES being linear and easily masked, the biggest differences between masked implementations are often found in the computation of the S-Box. There are mainly four popular approaches.

[1] https://developer.arm.com/ip-products/processors/cortex-m/cortex-m4

Table lookups is a popular way to implement the AES S-box, even in implementations that are not masked. This method consists in pre-computing tables to be used during the actual computation. During the actual computation of the S-Box, the implementation just has to fetch the value at the offset corresponding to the byte for which we want to compute the S-Box. However, it may lead to cache-timing attacks when the implementation is running on a processor where a data cache is available [19]. Thus, the precomputed tables must be built depending on the value of the first $d-1$ masks of the masked byte. However, doing so is very memory consuming when trying to protect against high-order attacks, as it requires to pre-compute and store tables for every possible different value for each mask.

The second approach is to consider the original description using an inversion in $\mathbb{F}_{2^8}$ and designing a masked implementation of this operation using multiplication and squaring.

The third approach is based on bitslicing. Instead of applying a (potentially complex) operation on one of the 16 bytes at a time, the idea is to slice each byte and see them as eight 16-bit words instead of sixteen 8-bit words, by gathering all the i-th bits in the i-th 16-bit word. For a masked implementation of the AES, instead of using a masked circuit for the inversion in $\mathbb{F}_{2^8}$ and applying it to the 16 bytes of the state, masking gadgets for the binary AND are used to build the full circuit of the S-Box from Boyar and Peralta [20] which will be executed on the 16 bytes in parallel thanks to bitslicing.

Another approach, called shareslicing, consists in regrouping the shares of the same value inside the same register. As in the case of bitslicing, this allows to parallelize the computation of the masked gadget by using only the most common instructions.

IV. Making Choices: Our Implementation

We first considered both shareslicing and bitslicing to implement the nonlinear layer of the AES. However, the security of shareslicing heavily relies on the assumption that there is no interaction between the bits of a register during the execution. In light of the work of Gao et al. [21], since our target uses a Cortex-M4 processor and to avoid relying on this assumption, we choose to not use this techniques for our implementation. Additionally, to avoid unwanted effects due to bit interactions inside registers, we make sure that the shares of a given value are always stored in different registers throughout the execution.

Hence, our implementation uses the bitsliced approach to implement all 16 S-boxes in parallel as in [22]. In non-masked implementations, bitslicing techniques are often used together with n-bit registers (with $n \geq 16$) to compute the AES encryption on $\frac{n}{16}$ blocks in parallel. In our case, $n = 32$ on Cortex-M4. However, during the computation of a masked implementation, each one of the eight 16-bit words of the bitsliced state is masked, which means that each word is in fact a masked state of $d+1$ 16-bit words. Without going too much into the details, our implementation is largely based on each of the 32 AND gates of the circuit must be replaced by a masked AND gadget working on 16 masked bits in parallel. Nonetheless, since our target is a microprocessor with 32-bit registers, we still want to maximize the use of our registers. To do so, we exploit the same technique shown in [22], where the authors proposed to group the 32 AND gates by pairs allowing to replace each pair by a masked AND gadget working on 32 masked bits in parallel such that each share is 32-bit wide.

The transformations of the linear layer can be applied share-wise which introduces much less overhead than for the nonlinear layers. Since we are using a bitsliced approach to the computation of the AES S-Box, the inputs and outputs of the nonlinear layers are in a bitsliced representation. To avoid having to pay costly bit manipulations that are needed to go from and into a more standard representation where each byte of the block state is stored in a single byte in memory, the linear components (*MixColumns*, *ShiftRows* and *AddRoundKey*) are implemented using the same bitsliced representation. Our implementation reuses already existing bitsliced version of *MixColumns* [15], [23] and *Shiftrows*.

During the design of the masked nonlinear layer, TightPROVE [24] is used to ensure that the circuit for the S-Box is secure at order d. However, this is not sufficient for the composition of rounds to secure as well at the same order. For this reason, in this paper only the security of the nonlinear layer circuit is formally proven. We conjecture that the composition is nonetheless secure and it can be done, for example, by using TightPROVE on the circuit that includes every linear operations after the output of the last AND gates of the previous nonlinear layer and the circuit of the S-box itself. The validation of the full implementation is thus left for future work.

A. Randomness Generation

Each gadget for the masked AND gate at order 3 (respectively 7) requires the generation of 5 (respectively 20) random masks. In our implementation, the masked functions for text encryption and key scheduler take as parameter a function pointer to `rng_fill`, a function that is used to fill a buffer of arbitrary size with random values. For our specific environment, `rng_fill` is implemented using the Random Number Generator (RNG) embedded in the STM32L432. This RNG is based on ring oscillators and allows to generate 32-bit words. A specific control register in memory is updated by the RNG module to tell whether or not the random value is ready to be read for the data register in memory. It takes approximately 64 cycles to generate each 32-bit word. The performance impact of the RNG on the overall implementation is discussed in Section IV-B.

The embedded RNG is working asynchronously with the rest of the chip and thus it may induce unexpected and non-deterministic delays while waiting for new random values to be drawn. Thus, during the leakage assessment presented in Section V-B, all needed random values are drawn beforehand and stored in a buffer. This buffer contains as many random values as needed by the function on which the assessment is conducted. This impacts negatively on the memory footprint of

TABLE I: Number of cycles to compute AES-128 keyschedule, encryption and S-Box, using either the embedded RNG or deterministic values.

Order d	3		7	
	With RNG	Without RNG	With RNG	Without RNG
AND gate	491	145	1771	497
S-Box	9353	3929	31025	10721
Keyschedule	100644	46393	323758	120716
Encryption	102442	48203	327383	124343

the implementation, but has the major advantage that two consecutive runs of the same function are taking exactly the same number of cycle to execute, allowing the analysis of multiple traces without risking desynchronization between them. This approach to random values generation is thus advantageous for an attacker and detrimental to its performance (since the RNG is a device running parallel to the processor). It is done only for leakage assessment and not in the final implementation.

The experimental leakage assessment exploits this RNG (through `rng_fill`), but also uses a fake alternative function that fills the buffer with deterministic values. This allows to observe almost directly the impact of masking by comparing the effect of the RNG on the detection of leakages.

B. Performance

The performance of our implementation in the number of cycles by block encryption is presented in Table I. As a comparison with an unprotected AES optimized for the same platform, Schwabe and Stoffelen report that their implementation is computing a single block encryption in 661.7 cycles [15]. In the same paper, they proposed an implementation masked at order 1 that takes 7422 cycles by using an RNG that outputs a 32-bit every 40 cycles, while the RNG we are using for our implementation is slower (64 cycles for each 32-bit words).

For masked implementations at higher orders, we do not directly compare to the one by Journault and Standaert [16] since they use masking schemes only at order 32. Also, we do not compare to the implementation [25] because it is targeted at ARM Cortex-A, a higher end family of processors with vectorized instructions. Instead, we compare our implementation to [22] which has the same target (ARM Cortex-M) and the same range of masking order. They report that their implementation uses $3280d^2+14075d+12192$ cycles to compute an AES encryption where d is the masking order, including the cost of randomness generation. However, they use a RNG that is even faster than the one used by Schwabe and Stoffelen: it is able to produce a 32-bit random value every 10 cycles.

C. Taking RNG performance into account

As seen in Table I and also reported in almost every implementations [15], [16], [17], [22], the performance of the RNG itself is crucial to be taken into account. In fact, in our implementation the cost of randomness generation is taking around 53% (respectively 62%) of the total number of cycles needed for a block encryption at order 3 (respectively order 7). Indeed, this impact strongly depends on the efficiency of the RNG used. Thus, to achieve a fair comparison, we approximate the cost of generating the needed random values as a function of the parameter n_{RNG}, which represents the number of clock cycles taken by the RNG to output a 32-bit random word.

At order 3, each S-Box is using 32 AND gates and each bitsliced AND gate on 16-bit operands uses 16×5 random bits; at order 7, each bitsliced AND gate on 16-bit operands uses 16×20 random bits. For each nonlinear layer of the AES, 32 AND gates on 16-bit operands are computed. Thus, the overall randomness usage of an S-Box computation is $32\times16\times5$ bits, that is **80** 32-bit words at order 3 and $32\times16\times20$ bits, that is **320** 32-bit words at order 7. In AES-128, there are 10 nonlinear layers to apply to the state, which means that at order 3 (respectively 7) our implementation needs $80\times10=800$ (respectively $320\times10=3200$) 32-bit words of randomness. Then, the cost of randomness generation for a full block encryption takes approximately $800n_{RNG}$ at order 3 and $3200n_{RNG}$ at order 7.

The implementation of Schwabe and Stoffelen [15] at order 1 is using $n_{RNG}=40$. With the same value for n_{RNG}, our implementation would take approximately $48203+800n_{RNG}=80203$ cycles to run at order 3 and $124343+3200n_{RNG}=252343$ at order 7.

The implementation of Goudarzi and Rivain [22] is using $n_{RNG}=10$ and takes 83937 cycles to run at order 3 and 271437 cycles at order 7. With the same value for n_{RNG}, our implementation would take approximately $48203+800n_{RNG}=56203$ cycles (vs 83937) at order 3 and $124343+3200n_{RNG}=156343$ (vs 271437) at order 7. This is an improvement of around 33% at order 3 and around 42% at order 7.

V. Secure Evaluation

A. Experimental Setup

Our implementation's test target is an STM32L432 Nucleo development board[2] that embeds an ARM Cortex-M4 processor, 256 kilobytes of flash memory and 64 kilobytes of SRAM. The processor has an embedded Random Number Generator (RNG) using ring oscillators to generate 32-bit random output.

We monitor the target board and acquire traces using a ChipWhisperer Lite Capture board[3]. To do so, we configure and connect six of the 30 pins available on the STM32L432 to establish a synchronization protocol between the target board and the capture board.

Since masking is a generic counter-measure that is side-channel agnostic, we chose to measure leakages from the processor using an electromagnetic directional probe from Langer EMV-Technik[4]. The probe is then connected to a pre-

[2]https://www.st.com/en/evaluation-tools/nucleo-l432kc.html
[3]https://store.newae.com/chipwhisperer-lite-cw1173-two-part-version/
[4]https://www.langer-emv.de/en/product/lf-passive-100-khz-up-to-50-mhz/36/lf-b-3-h-field-probe-100-khz-up-to-50-mhz/3

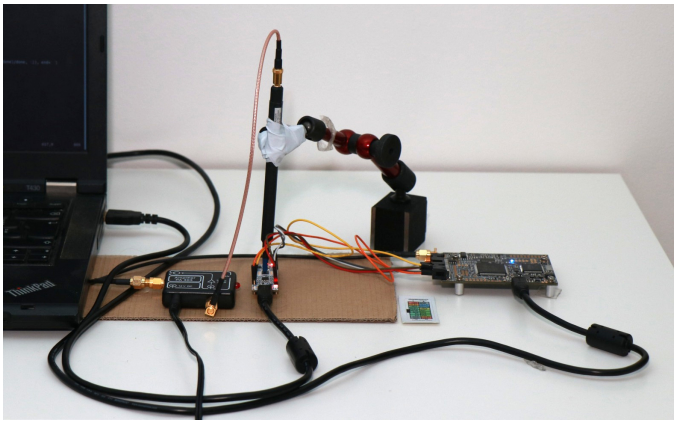


Fig. 1: The hardware bench.

amplifier in order to amplify the signal before it reaches the capture board. This setup is illustrated in Figure 1.

Every experiment will be done twice: once using deterministic values in lieu of random masks (RNG off); and a second time using the integrated RNG of the STM32L432 (RNG on) to generate these random masks. Doing so allows to directly compare the effect of masking on the leakages. It also helps distinguishing between points in time that are not depending on the processed data (e.g., loop counter increment or unconditional jump) with points in time actually leaking information to the attacker. Thus, this can be used to get rid of many points before doing a multivariate analysis by keeping only the most information-carrying points, reducing the overall cost of the assessment.

The capture board used in our setup is limited to 24 000 time samples at once. This means that we are not able to make a measurement for each cycle during the full masked encryption for neither order 3 nor order 7, which takes respectively 48 203 and 124 343 cycles to execute. Instead, we assess the leakages that occurs during the computation of the S-boxes. We generate the required random values beforehand. This allows to run the implementation at both order 3 (in less than 4000 cycles) and order 7 (in less than 11000 cycles) and assess the leakage under the same conditions. A full assessment of the complete implementation and with an improved setup is planned for the near future.

B. Leakage Assessment

The Test Vector Leakage Assessment (TVLA) is a procedure that consists in trying to detect statistically meaningful differences between two datasets containing traces measured during the computation on the device under test and differing with respect to a specific parameter: for instance, one containing the traces measured during the encryption of a constant plaintext with a constant key, whereas the second dataset contains the traces measured during the encryption of a random plaintext with the same key. Being able to statistically distinguish the two sets of traces would mean that what is leaked during the execution depends on the plaintext on which the encryption is performed.

This can be seen as a necessary condition to potential attacks against the implementation, but the number of total traces analysed is crucial in experimental evaluation. It is not uncommon to start observing significant differences between the two datasets only after several hundred thousands traces measured if the environmental measurement noise is significant or when some counter-measures are implemented.

The underlying statistical test commonly used is a Welch's t-test done for each point in time. The common approach to aggregate every trace inside each dataset is to compute their mean and to try to distinguish this mean from the mean of the other dataset. The Welch's t-test is producing a statistic t:

$$t = \frac{\mu_1 - \mu_2}{\sqrt{\frac{{s_1}^2}{n_1} + \frac{{s_2}^2}{n_2}}},$$

with μ_i the mean of the i-th dataset, s_i its variance and n_i its size. The greater the absolute value of t, the more statistically significant the difference.

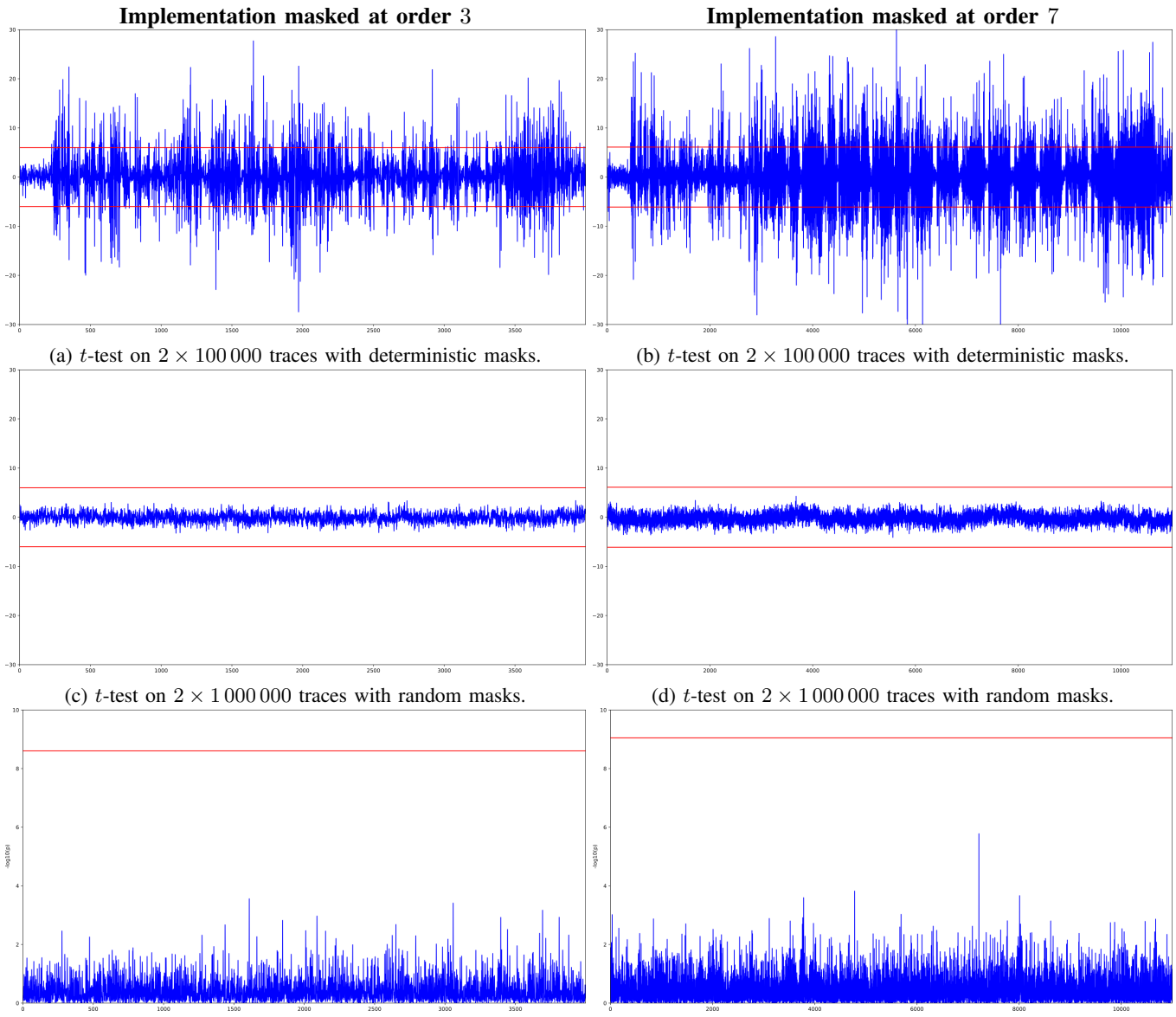
In the original proposal of the TVLA, the two datasets are compared using a common threshold directly on the value of t. This threshold is fixed to ± 4.5 for every point in time. This means that the device under test is said to successfully pass the test (i.e., no leakage is found) if the maximum absolute value of t over all points in time is lower than 4.5. In [26], the authors proposed to try to distinguish these datasets by using moments of higher order (variance, skewness, kurtosis, ...) instead of the variances.

Replacing the Welch's t-test by Pearson's χ^2-test has been proposed in 2015 as a complementary approach [27]. As for the t-test, this test aims at trying to distinguish the two datasets. However, it does not consists in comparing the means (or higher-order moments) of the observations but instead it works directly on the whole distribution. Nonetheless, this test also work at the granularity level of a single point in time.

In the χ^2-test setting, there is no concept of moment of higher order as for the t-test since it does not use moments. In some case the χ^2-test can directly detect leakages that otherwise would need a higher-order analysis in the case of the t-test. We use this test as it was intended and described in the work of Moradi *et al.* [27], that is as a complementary approach to the t-test. The results are visible in Figure 2: on the left for order 3, and on the right for order 7. The latter looks more leaking than the former, which is a likely artifact created by more complex code. From top to bottom, the reader can se the TVLA analysis for the deterministic (i.e., unprotected) and random masks using Welch's t-test, and the χ^2-test using random masks. The latter has also been performed on deterministic masks: it shows obvious leakages and the result is thus not shown for brevity.

C. Multivariate analysis

In the TVLA methodology, both the t-test and the χ^2-test are conducted on each point in time independently. This is sometimes called a univariate (or vertical) analysis, in opposition with a multivariate (or horizontal) analysis where $n > 1$ points in time are considered simultaneously. In a



(a) t-test on $2 \times 100\,000$ traces with deterministic masks.

(b) t-test on $2 \times 100\,000$ traces with deterministic masks.

(c) t-test on $2 \times 1\,000\,000$ traces with random masks.

(d) t-test on $2 \times 1\,000\,000$ traces with random masks.

(e) χ^2-test on $2 \times 1\,000\,000$ traces with random masks.

(f) χ^2-test on $2 \times 1\,000\,000$ traces with random masks.

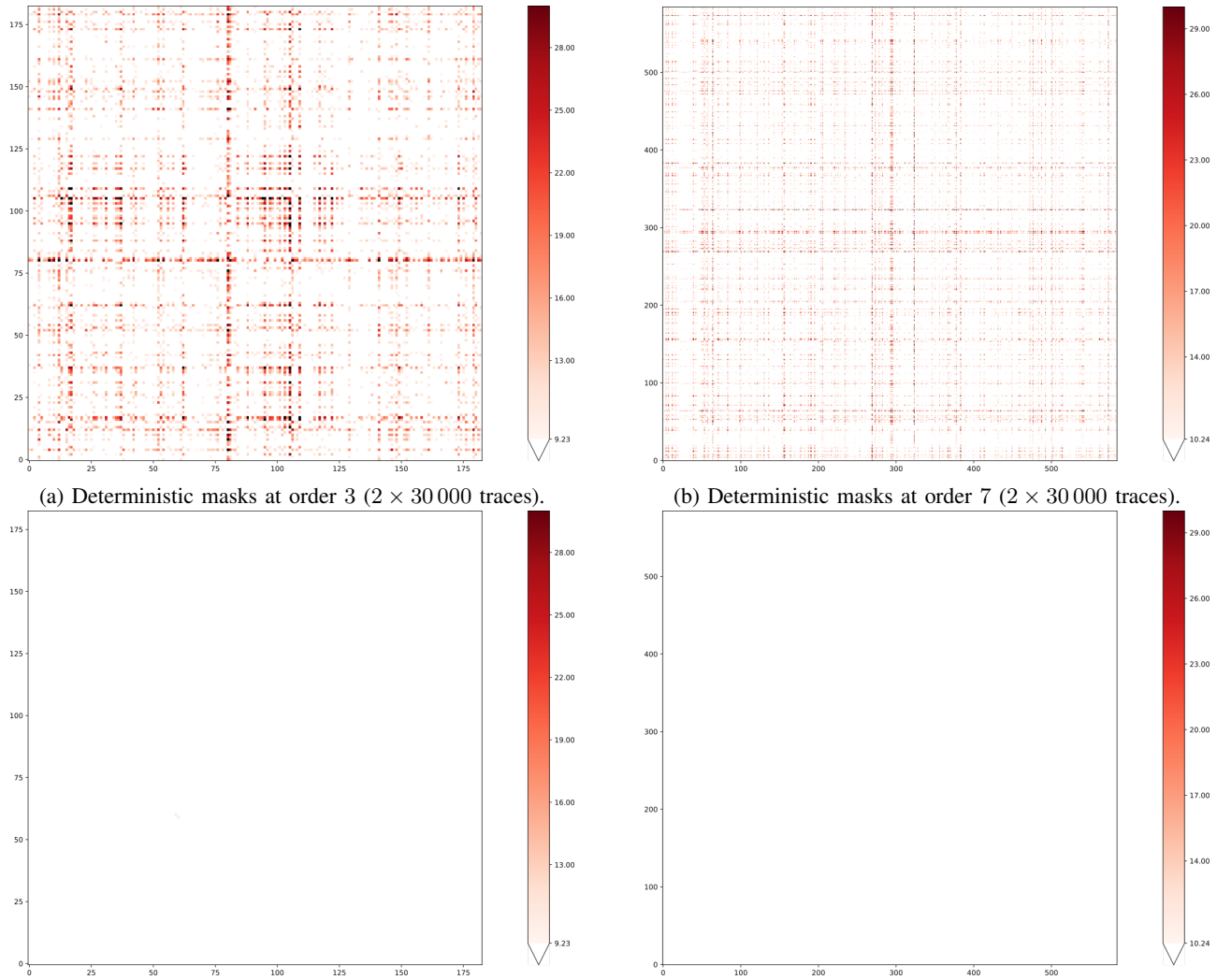
Fig. 2: TVLA (fixed vs. random) on the S-box layer masked at order 3 (left) and 7 (right). Top figures (a,b) use deterministic masks (i.e., no masking).

multivariate setting, the distribution of subsets of n points are compared in order to distinguish between the fixed and the random dataset.

An univariate leakage assessment is more adapted to an implementation where multiple shares are manipulated in parallel as in hardware implementations of masked circuits. However, in software implementations the computation are done sequentially and different shares of the same value are manipulated at different clock cycles, except when the value in memory or in a register is overwritten with another value. Thus, a multivariate analysis is often needed to successfully achieve a side-channel attack on masked software implementation.

The major drawback of a multivariate analysis is its efficiency: when the number of datapoint of a single trace increases, the number of different subsets of n points grows exponentially. The goal of an attacker is then to find the so-called *Points of Interest* (PoI), which are a reduced set of points in time that are carrying the most information and are thus more susceptible to lead to an attack. Detecting those points is often the most challenging part, while the Welch's t-test or the χ^2-test can be adapted to the multivariate case by carefully choosing an aggregation function that is applied on the subsets of n points.

We conduct a bivariate analysis using the χ^2-test on our implementation by following the methodology of Moradi *et*



(a) Deterministic masks at order 3 ($2 \times 30\,000$ traces).

(b) Deterministic masks at order 7 ($2 \times 30\,000$ traces).

(c) Random masks at order 3 ($2 \times 1\,000\,000$ traces).

(d) Random masks at order 7 ($2 \times 1\,000\,000$ traces).

Fig. 3: Bivariate χ^2-test (fixed vs. random) on the S-box layer masked at order 3 (183 Points of Interest) and 7 (585 POI).

al. [27]. In order to limit the time and memory complexity of a bivariate analysis, we keep only the points in time that actually carry information. We do so by filtering out the ones that do not go above the threshold during the univariate χ^2 analysis on the implementation using deterministic masks. Thanks to this step, we are able to reduce the number of points from 4000 to 183 at order 3, and from 11000 to 585 at order 7, corresponding to an improvement by a factor of about 20.

The result of the bivariate analysis is shown in Figure 3. The colour of each pixel represents the p-value of the combination of the two points in time at the corresponding coordinate. The diagonal thus corresponds to the univariate analysis. Since our bivariate analysis relies on a symmetric combination function of each point, the resulting graph is also symmetric. p-values above the threshold are shown in levels of red and the ones that are below the threshold are in plain white. When deterministic masks are used, only $30\,000$ traces for each dataset are sufficient to detect clear differences between the two datasets whereas after $1\,000\,000$ traces for each dataset, no bivariate leakage has been found for the implementation at both order 3 and 7 when the embedded RNG is used to generate the masks.

VI. Conclusion and Perspectives

In this paper, we presented a secure implementation of the AES and conducted an experimental assessment of the leakage that occurs when executing it giving more confidence in its security. We have shown that performance is largely affected by the RNG production rate, and that our implementation [28] improves comparable implementations by 30 to 40%. We have shown the effectiveness of the masking scheme, formally proven, against univariate and bivariate statistical analysis to prove the actual absence of leakages. It must be pointed out, however, that only the nonlinear layer of this implementation is formally proven to be d-order secure, and that only this step has been assessed against leakage due to limitations of our experimental platform. In the future, the analysis will

be extended to the whole AES implementation, by formally proving the complete code, assessing the leakage through higher-order statistical tests, or by more extensive experiments on the EM leakage, and porting to RISC-V architectures.

ACKNOWLEDGMENT

This work has been supported by the French National Research Agency in the framework of the "Investissements d'avenir" program (ANR-15-IDEX-02).

REFERENCES

[1] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Proceedings*, ser. Lecture Notes in Computer Science, N. Koblitz, Ed., vol. 1109. Springer, 1996, pp. 104–113.

[2] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Proceedings*, ser. Lecture Notes in Computer Science, M. J. Wiener, Ed., vol. 1666. Springer, 1999, pp. 388–397.

[3] J. Quisquater and D. Samyde, "Electromagnetic analysis (EMA): measures and counter-measures for smart cards," in *Smart Card Programming and Security, International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Proceedings*, ser. Lecture Notes in Computer Science, I. Attali and T. P. Jensen, Eds., vol. 2140. Springer, 2001, pp. 200–210.

[4] V. Lomné and T. Roche, "A side journey to titan," *IACR Cryptol. ePrint Arch.*, p. 28, 2021. [Online]. Available: https://eprint.iacr.org/2021/028

[5] C. Clavier, J. Coron, and N. Dabbous, "Differential power analysis in the presence of hardware countermeasures," in *Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Proceedings*, ser. Lecture Notes in Computer Science, Ç. K. Koç and C. Paar, Eds., vol. 1965. Springer, 2000, pp. 252–263.

[6] J. Coron and I. Kizhvatov, "Analysis of the split mask countermeasure for embedded systems," in *Proceedings of the 4th Workshop on Embedded Systems Security, WESS 2009*, D. N. Serpanos and W. H. Wolf, Eds. ACM, 2009.

[7] ——, "Analysis and improvement of the random delay countermeasure of CHES 2009," in *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Proceedings*, ser. Lecture Notes in Computer Science, S. Mangard and F. Standaert, Eds., vol. 6225. Springer, 2010, pp. 95–109.

[8] M. J. Kannwischer, P. Pessl, and R. Primas, "Single-trace attacks on keccak," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 3, pp. 243–268, 2020.

[9] F. Durvaux, M. Renauld, F. Standaert, L. van Oldeneel tot Oldenzeel, and N. Veyrat-Charvillon, "Efficient removal of random delays from embedded software implementations using hidden markov models," in *Smart Card Research and Advanced Applications - 11th International Conference, CARDIS 2012, Revised Selected Papers*, ser. Lecture Notes in Computer Science, S. Mangard, Ed., vol. 7771. Springer, 2012, pp. 123–140.

[10] E. Cagli, C. Dumas, and E. Prouff, "Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing," in *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Proceedings*, ser. Lecture Notes in Computer Science, W. Fischer and N. Homma, Eds., vol. 10529. Springer, 2017, pp. 45–68.

[11] L. Goubin and J. Patarin, "DES and differential power analysis (the "duplication" method)," in *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Proceedings*, ser. Lecture Notes in Computer Science, Ç. K. Koç and C. Paar, Eds., vol. 1717. Springer, 1999, pp. 158–172.

[12] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Proceedings*, ser. Lecture Notes in Computer Science, M. J. Wiener, Ed., vol. 1666. Springer, 1999, pp. 398–412.

[13] G. Barthe, S. Belaïd, F. Dupressoir, P. Fouque, B. Grégoire, P. Strub, and R. Zucchini, "Strong non-interference and type-directed higher-order masking," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 116–129.

[14] G. Cassiers and F.-X. Standaert, "Trivially and efficiently composing masked gadgets with probe isolating non-interference," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2542–2555, 2020.

[15] P. Schwabe and K. Stoffelen, "All the AES you need on cortex-m3 and M4," in *Selected Areas in Cryptography - SAC 2016 - 23rd International Conference, Revised Selected Papers*, ser. Lecture Notes in Computer Science, R. Avanzi and H. M. Heys, Eds., vol. 10532. Springer, 2016, pp. 180–194.

[16] A. Journault and F. Standaert, "Very high order masking: Efficient implementation and security evaluation," in *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Proceedings*, ser. Lecture Notes in Computer Science, W. Fischer and N. Homma, Eds., vol. 10529. Springer, 2017, pp. 623–643.

[17] D. Goudarzi, A. Journault, M. Rivain, and F. Standaert, "Secure multiplication for bitslice higher-order masking: Optimisation and comparison," in *Constructive Side-Channel Analysis and Secure Design - 9th International Workshop, COSADE 2018, Proceedings*, ser. Lecture Notes in Computer Science, J. Fan and B. Gierlichs, Eds., vol. 10815. Springer, 2018, pp. 3–22.

[18] N. Bordes and P. Karpman, "Fast verification of masking schemes in characteristic two," in *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part II*, ser. Lecture Notes in Computer Science, A. Canteaut and F. Standaert, Eds., vol. 12697. Springer, 2021, pp. 283–312.

[19] E. Tromer, D. A. Osvik, and A. Shamir, "Efficient cache attacks on AES, and countermeasures," *J. Cryptol.*, vol. 23, no. 1, pp. 37–71, 2010.

[20] J. Boyar and R. Peralta, "A new combinational logic minimization technique with applications to cryptology," in *Experimental Algorithms, 9th International Symposium, SEA 2010, Proceedings*, ser. Lecture Notes in Computer Science, vol. 6049. Springer, 2010, pp. 178–189.

[21] S. Gao, B. Marshall, D. Page, and E. Oswald, "Share-slicing: Friend or foe?" *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 1, pp. 152–174, 2020.

[22] D. Goudarzi and M. Rivain, "How fast can higher-order masking be in software?" in *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part I*, ser. Lecture Notes in Computer Science, J. Coron and J. B. Nielsen, Eds., vol. 10210, 2017, pp. 567–597.

[23] E. Käsper and P. Schwabe, "Faster and timing-attack resistant AES-GCM," in *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Proceedings*, ser. Lecture Notes in Computer Science, C. Clavier and K. Gaj, Eds., vol. 5747. Springer, 2009, pp. 1–17.

[24] S. Belaïd, D. Goudarzi, and M. Rivain, "Tight private circuits: Achieving probing security with the least refreshing," in *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Proceedings, Part II*, ser. Lecture Notes in Computer Science, T. Peyrin and S. D. Galbraith, Eds., vol. 11273. Springer, 2018, pp. 343–372.

[25] B. Grégoire, K. Papagiannopoulos, P. Schwabe, and K. Stoffelen, "Vectorizing higher-order masking," in *Constructive Side-Channel Analysis and Secure Design - 9th International Workshop, COSADE 2018, Proceedings*, ser. Lecture Notes in Computer Science, J. Fan and B. Gierlichs, Eds., vol. 10815. Springer, 2018, pp. 23–43.

[26] A. Moradi, "Statistical tools flavor side-channel collision attacks," in *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings*, ser. Lecture Notes in Computer Science, D. Pointcheval and T. Johansson, Eds., vol. 7237. Springer, 2012, pp. 428–445.

[27] A. Moradi, B. Richter, T. Schneider, and F. Standaert, "Leakage detection with the x2-test," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, no. 1, pp. 209–237, 2018.

[28] N. Bordes, "Masked AES-128 for Cortex M - Implementation and evaluation," https://gricad-gitlab.univ-grenoble-alpes.fr/bordesn/masked-aes128-cortexm-implem-eval, 2021.