Second-Order Low-Randomness $d + 1$ Hardware Sharing of the AES

Siemen Dhooghe*
imec-COSIC, ESAT, KU Leuven
Leuven, Belgium
siemen.dhooghe@esat.kuleuven.be

Aein Rezaei Shahmirzadi*
Ruhr University Bochum
Horst Görtz Institute for IT Security
Bochum, Germany
aein.rezaeishahmirzadi@rub.de

Amir Moradi
Ruhr University Bochum
Horst Görtz Institute for IT Security
Bochum, Germany
amir.moradi@rub.de

ABSTRACT

In this paper, we introduce a second-order masking of the AES using the minimal number of shares and a total of 1268 bits of randomness including the sharing of the plaintext and key. The masking of the S-box is based on the tower field decomposition of the inversion over bytes where the changing of the guards technique is used in order to re-mask the middle branch of the decomposition. The sharing of the S-box is carefully crafted such that it achieves first-order probing security without the use of randomness and such that the sharing of its output is uniform. Multi-round security is achieved by re-masking the state where we use a theoretical analysis based on the propagation of probed information to reduce the demand for fresh randomness per round. The result is a second-order masked AES which competes with the state-of-the-art in terms of latency and area, but reduces the randomness complexity over eight times over the previous known works. In addition to the corresponding theoretical analysis and proofs for the security of our masked design, it has been implemented on FPGA and evaluated via lab analysis.

KEYWORDS

AES, Hardware, Low Randomness, Masking, Side-Channel Analysis

1 INTRODUCTION

In 1999, Kocher *et al.* [31] published an article on side-channel attacks where a device's physical properties such as its power consumption are used in order to retrieve the key from a block cipher. The attack, known as Differential Power Analysis (DPA), consists of taking several measurements of the power consumption of the device and correlating it with a key guess under a power model such as the Hamming weight function. In order to combat side-channel attacks, Chari *et al.* [12] and Goubin and Patarin [23] independently introduced Boolean masking. With masking, each secret value $x \in \mathbb{F}_2$ (for example values depending on the plaintext or key of a block cipher) is split in multiple parts $(x^0, ..., x^n) \in \mathbb{F}_2^n$ such that $\sum_{i=0}^{n} x^i = x$. The essential idea of masking is based on *noise amplification*. Observing multiple parts of the computation at the same time exponentially increases the noise (in terms of the number of parts) which was present due to measuring errors or noise inherent to the device. This allows the industry to create countermeasures where attackers would need several millions if not billions of power samples in order to break it.

Threshold Implementation (TI) is the first strategy that provides security against side-channel attacks in hardware platforms [38]. The approach has been extended in higher orders in [7], but fails to provide multivariate security due to the lack of fresh masks as discussed in [39]. While TI defines the number of input shares based on the algebraic degree of the target function as well as the desired security order, it has been shown that the same level of security can be achieved using the minimum number of shares, i.e., $d + 1$ input shares [24, 40]. However, security flaws with higher-order attacks have been reported in [34] due to insufficient refreshing, highlighting the need for providing precise security claims.

Around the same time power analysis attacks were made public, the National Institute of Standards and Technology's (NIST) competition had ended with the Rijndael cipher as its winner to become the Advanced Encryption Standard (AES) [17]. Ever since, the AES has become a key component in modern day's industrial cryptographic solutions. This includes solutions on embedded devices which means they are vulnerable to side-channel attacks. As a result, there is a practical need for maskings of the AES and, in turn, this need requires the solutions to be efficient and industrially viable. The academic community responded to this need causing several innovations on tackling AES's masking (which turned out to be non-trivial due to the eight-bit high-degree S-box). In 2016, De Cnudde *et al.* [18] proposed a first and second-order masking of the AES using a tower field decomposition. This was improved in 2017 by Groß *et al.* [25] using domain-oriented masking. In 2018, De Meyer *et al.* [19] further improved the area cost by considering a multiplicative masking approach.

The solutions for masking the AES have significantly advanced over the years. However, as time progressed, researchers noted a hidden cost in the solutions. Namely, each masking required fresh random bits in order to secure its computation. The cost for the generation of this randomness was not reported due to its dependence on the generator present in the platform or due to the dependence on the used pseudo-random number generator. However, it was clear this generational cost was not trivial. In the work by De Cnudde *et al.* [18], they reported using several unrolled implementations of the PRINCE cipher [8]. For second-order security, this caused the randomness generation (the PRINCE implementations) to take up a larger area than the masking itself. Considering the masking was made with minimal area as the efficiency goal, this was a significant additional overhead. As a result, the authors of the work posed the open question to reduce the randomness cost of their masking strategy. The result that randomness generation would take up such a large part of the cost was not unique among the masked solutions. In the work by De Meyer *et al.* [19], the Trivium stream cipher [9] was used to generate the randomness which would result in a 50% overhead on the area of the second-order masking. Later, the authors of [28] constructed and synthesized

*These authors contributed equally to this work.

different variants of Pseudo Random Number Generators (PRNGs), including the Linear-Feedback Shift Register (LFSR) and randomness generation based on Keccak-f. They demonstrated that the implementation costs required for the generation of mask bits is high. Recently, several works addressed this issue and presented some techniques to reduce the randomness complexity. In [22], a new techniques has been proposed for masking algorithms to share randomness over multiple gadgets, reducing the total number of required fresh masks. Later, a technique provided in [46] which amortizes the randomness costs of several parallel masked multiplications in higher orders. In an independent work, the randomness cost was reduced further for masked multiplications while the running time of pseudo-random number generation was improved in software platforms [15]. All mentioned works mainly focused on software platforms and their application to hardware platforms is quite limited. In this paper, we focus on hardware platforms to reduce randomness complexity while maintaining higher order security which has gained only very little attention in the literature so far.

Over the last few years, it has become clear that in order to make efficient maskings their randomness cost must be reduced. As a result, in 2018 Wegener and Moradi [48] made a first-order secure AES that required no fresh randomness. But as a trade-off, the masking needed four shares in order to operate, which significantly increased the area of the masking itself. Moreover, the latency of design in terms of the number of clock cycles was considerably high. Later on, in 2019 Sugawara [45] made a three-shared masked AES without the need for fresh randomness. Their solution still has an increased area overhead compared to solutions using fresh randomness as they require three shares, but the trade-off already improved. In 2021, Beyne *et al.* [5] provided the first second-order AES with low-randomness. Their result was based on the sharing by Wegener and Moradi and, again, required four shares and a significant increase in area. To conclude, no solution has thus far provided a masking of the AES with a reduced randomness complexity using the minimal number of shares and with a fair area trade-off.

Related Works. One of the first attempts to make a second-order secure AES S-box is presented in 2015 [13]. Based on the Threshold Implementation (TI) strategy, the authors provided a masked variant of AES S-box using six input shares and 126 bits of fresh masks per S-box. Later in 2016, based on "Consolidated Masking Schemes" (presented in [40]), the authors of [18] provided a second-order secure AES design using the minimum number of input shares, i.e., three shares. Although their design is more cost-effective in terms of area overhead, the demand for fresh masks is higher as each S-box needs 162 bits of fresh masks. A more compact implementation was introduced in [25] with three shares using the Domain Oriented Masking (DOM) scheme which reduced the randomness cost to 54 bits per S-box. While all aforementioned designs used Boolean masking as the underlying scheme, in 2018 the authors of [19] used multiplicative masking and demonstrated that it can be more efficient in hardware. Their S-box construction uses 53 bits of fresh masks while providing the same level of security with a lower area overhead. Note that all the aforementioned fresh masks should be updated every clock cycle to provide security. Generation of such a high number of fresh masks can be even more costly than a masked S-box or even more than the entire masked cipher. As a result, there is a need to find a secure design with lower randomnesses cost with the same area overhead and latency to make its applicability more feasible in small embedded devices.

Contributions. Inspired by the open question posed by De Cnudde *et al.* [18] and by the significant overheads on area cost of AES implementations without fresh randomness [5, 45, 48], we investigated masking schemes to protect the AES. We achieved to make a second-order masking of the AES using the minimal number of shares that has an over eight times reduction of the randomness cost over the current state-of-the-art solutions.

To achieve this reduction, we needed a first-order probing secure S-box with no randomness using three input shares. This was created using the following components.

- A uniform three-share multiplier made by generalizing the algorithm from Shahmirzadi and Moradi [44].
- A first-order probing secure $GF(16)$ inverter using three shares made via duplication of the square-scale-and-multiply function.
- A uniform sharing for two parallel multipliers made by the use of the changing of the guards technique paired with a "two-stage shared Feistel" to allow the invertibility of the construction.

The second-order security of a single round is achieved by adding fresh masks to the S-box (with the upside that now we can use the same randomness for every S-box). The probing security of multiple rounds is then achieved by the following techniques.

- The chaining of the S-boxes over the columns using the changing of the guards.
- The refreshing of the columns per round to ensure probed information does not diffuse outside each column.

As the randomness generation in hardware platforms, e.g., ASIC, can be very costly, we were able to reduce the fresh randomness complexity of our design with the above mentioned carefully crafted masked functions while maintaining the area overhead and latency, in terms of the number of clock cycles, in the same range compared to the state-of-the-art. We support our claims with theory analyses, confirmed the security of our S-box construction with a formal verification tool, and conducted an experimental analysis on an FPGA-based evaluation board. The full HDL codes are available in GitHub.

2 PRELIMINARIES

In this section, we introduce the AES, the probing model, threshold implementations, the changing of the guards construction, and $d+1$ sharing.

2.1 Description of AES

We quickly introduce the AES-128 cipher [17]. AES-128 consists of a 128-bit state and a 128-bit key divided into bytes. The cipher is composed of 10 rounds each applying an addition of a subkey, a bricklayer of S-boxes, a `ShiftRows` operation, and a `MixColumns` operation. This is visually represented in Figure 1. The S-box consists of an inversion over $GF(256)$ (extended so zero is mapped

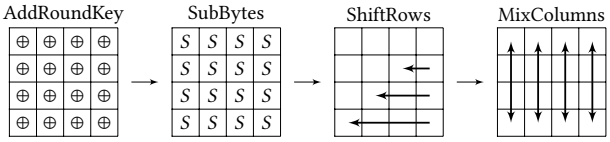


Figure 1: Representation of the AES.

to zero) and an affine map. The key schedule for AES-128, which operates on 4 columns of 32 bits each, is depicted in Figure 2.

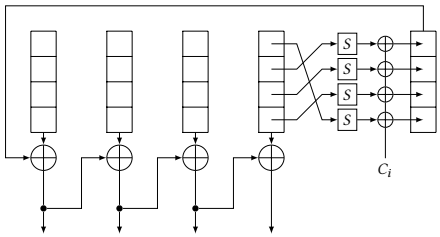


Figure 2: The AES-128 key schedule. The i^{th} round constants are denoted by C_i.

2.2 Glitch-Extended Probing Security

The security model considered in this work is "glitch-extended probing model" [40], which is a security model based on "probing model" introduced by Ishai *et al.* [27]. This abstract but effective model has been used in considerable scientific papers to prove the security of their constructions.

Consider that a masked algorithm is represented as a circuit, a directed acyclic graph where each wire represents a digital value (typically a bit) and each vertex is a binary operation. In the dth-order probing model, an adversary is able to observe d intermediate wires of the circuit during the execution of the algorithm. These d wires should be simulatable by a simulator who is not given any of the secret values (such as the key or plaintext of a cipher). As a result, it suffices to show the probed values are random values following some fixed distribution independent of the secret values.

While this security model is a good first step in the provable security of side-channel countermeasures, it does not reflect all realistic properties of hardware implementations. Specifically, the probing model provides security under the assumption that there is no data-dependent activation timing, which fits best on software platforms. This is due to a common phenomenon in CMOS technologies called glitches. The inaccurate assumption of the probing model not capturing the effect of glitches leads to insecure designs as shown in [33, 36]. More precisely, the physical characteristics such as transitions, coupling effects, or glitches are not considered in the d-probing model [21]. Hence, an extended model is needed for these unwanted effects.

One of the most promising models to capture the above mentioned effects is presented by Faust *et al.* [21] called the *robust probing model*, this model covers the physical properties of hardware platforms. For example, to cover the effect of glitches, a *glitch-extended* probe is made where a probe on a combinatorial circuit is extended to all signals (up to registers or primary inputs) that involve in the computation of the probed wire. The introduction of such a simple model enabled the relevant scientific communities to develop formal verification tools to evaluate a small circuit [1, 29] (small due to the limitations of the complexity of the method). Furthermore, it helped to reduce the implementation cost of several schemes while maintaining the same level of security [3, 43, 44].

2.3 Boolean Masking and Threshold Implementations

Boolean masking is a technique based on splitting each secret variable $x \in \mathbb{F}_2$ in the circuit into shares $\bar{x} = (x^0, x^1, \ldots, x^{s_x-1})$ such that $x = \sum_{i=0}^{s_x-1} x^i$ over $\mathbb{F}_2$. A random Boolean masking of a fixed secret is uniform if all sharings of that secret are equally likely. There are several approaches to masking a circuit. In this work, we make use of threshold implementations proposed by Nikova *et al.* [38].

Let $\bar{F}$ be a shared function in the threshold implementation corresponding to the unshared function $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$. Then $\bar{F}$ can have the following properties.

Definition 2.1 (Threshold Implementations [7, 38]). Let $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a function and $\bar{F} : \mathbb{F}_2^{ns_x} \to \mathbb{F}_2^{ms_y}$ be a sharing of F. The sharing $\bar{F}$ is said to be

(1) *correct* if $\sum_{i=0}^{s_y-1} F^i(x^0, \ldots, x^{s_x-1}) = F(x)$ for all $x \in \mathbb{F}_2^n$ and for all shares $x^0, \ldots, x^{s_x-1} \in \mathbb{F}_2^n$ such that $\sum_{i=0}^{s_x-1} x^i = x$,
(2) d^{th}*-order non-complete* if any function in d or fewer shares F^i depends on at most $s_x - 1$ input shares,
(3) *uniform* if $\bar{F}$ maps a uniform random sharing of any $x \in \mathbb{F}_2^n$ to a uniform random sharing of $F(x) \in \mathbb{F}_2^m$.

The property of uniformity is important to this work as it allows for the design of first-order and higher-order secure designs using minimal fresh random bits since the entropy from the input of each shared function is preserved. Following the work by Dhooghe *et al.* [20], in a second-order setting where the shared functions are uniform, multivariate (multi-round) security is ensured by having fresh masks be present after each possible probe position in that round (which in [20] translates to the round function being resilient-uniform). The security is guaranteed by the first probe's observed values being re-masked causing the second probe to view independent information from the first one. Since the shared functions are uniform, both probes view uniform random information (which means both probes view information independent on the secret). As a result, we can reduce the randomness required to re-mask the round function. For example, instead of re-masking the full state after the `MixColumns` layer, it suffices to re-mask the columns where the same randomness is used for each column in case the shared functions are uniform. Such added randomness is depicted in Figure 8.

2.4 Changing of the Guards

The changing of the guards proposed by Daemen [16] is a technique that transforms a non-complete sharing into a uniform and non-complete sharing. The technique works by embedding the sharing into a Feistel-like structure. We use the extension by Beyne *et al.* [5]

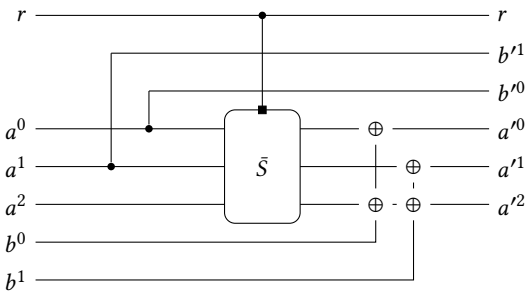


Figure 3: The changing of the guards with additional randomness as input for three shares. We note that in our masked AES, we will change the way the guards are added to the output of the shared function.

where a first-order probing secure sharing is considered instead of a non-complete one. That implies that it is possible to use the changing of the guards method on sharings using multiple register stages and extra randomness to guarantee their security. The changing of the guards method then allows for the uniformity of the larger circuit, in turn, allowing for the re-use of the randomness in the circuit.

We give the changing of the guards method formally and depict the structure for three shares in Figure 3.

Definition 2.2. The changing of the guards method applied to a shared map $\bar{S}$ given inputs $(a^0, ..., a^s)$, $(b^0, ..., b^s)$, and randomness r is calculated as follows

$$
\begin{aligned}
&r' = r,\\
&a'^0 = S^0(a^0, ..., a^s, r) \oplus b^0, \ldots, a'^{s-1} = S^{s-2}(a^0, ..., a^s, r) \oplus b^{s-1},\\
&a'^s = S^s(a^0, ..., a^s, r) \oplus b^0 \oplus \ldots \oplus b^{s-1},\\
&b'^0 = a^0, \ldots, b'^{s-1} = a^{s-1}.
\end{aligned}
$$

We refer to the $(b^0, ..., b^{s-1})$ as the *guards* of the shared S-box $\bar{S}$.

2.5 $d+1$ Sharing

Threshold Implementations (TI) have been proven to be a reliable solution as they are immune against glitches. However, the number of input shares depends on the algebraic degree of the target function as well as on the desired security order which can make the implementation cost of masked circuits unaffordable. It becomes even more challenging when higher-order security is desired as fresh masks should be introduced to achieve security against multivariate adversaries [13, 39].

By removing the dependency of the number of shares on the algebraic degree, it has been shown that the minimum number of $d+1$ shares can be employed to realize a dth-order secure design. The authors of [40] demonstrated that first-order security can be achieved with no fresh mask using the minimum number of shares for a certain group of quadratic functions. While it is limited to first-order security and only quadratic functions, a methodology called Domain Oriented Masking (DOM) has been introduced later in [24] offering side-channel security in hardware platforms with arbitrary protection order. As a side note, it has been shown that higher-order variants using this scheme are affected by security flaws as reported in [34]. The scheme uses fresh randomness to achieve glitch-extended probing security and splits quadratic operations into two parts by registers to avoid the propagation of glitches. The fresh masks should be added right before register stage to assure that the extended probe on the output of the register is independent of the secret. For example, a two-share masked variant of a simple two-input AND gate $f(a, b) = x$, with input shares a^0, a^1, b^0, b^1 and output shares x_0, x_1, can be implemented as follows

$$
\begin{array}{llll}
f^0(a^0, b^0) & = a^0 b^0 & \rightarrow x^0 & \\
f^1(a^0, b^1, r) & = a^0 b^1 + r & \rightarrow x^1 & x^0 + x^1 = y^0 \\
\hline
f^2(a^1, b^0, r) & = a^1 b^0 + r & \rightarrow x^2 & x^2 + x^3 = y^1 \\
f^3(a^1, b^1) & = a^1 b^1 & \rightarrow x^3 &
\end{array},
$$

where r is a bit of fresh randomness. The coordinate functions' f^i output must be stored in registers. The phase that is dedicated to the coordinate functions' computation is known as the *expansion layer*. These registered results are XORed to generate the output shares and it is referred to as the *compression layer*.

The demand for fresh masks in $d+1$ sharing schemes has been relaxed in [43]. They presented an algorithm enabling them to find two-share first-order secure masked realizations of several lightweight block ciphers with no fresh masks. In this paper, we are dealing with second-order security meaning we use at least three shares. Hence, we adjusted the algorithm to find first-order secure solution of the target function where no fresh masks are used. We then add fresh randomness before the compression layer to make the previous found solution second-order secure. This enables us to reuse the fresh masks in the expansion layer in other nonlinear operations. As we use this algorithm for quadratic functions, we explain our strategy to find a first-order glitch-extended probing secure and uniform sharing of a constant-free arbitrary quadratic function with two inputs $y = f(a, b)$ following the instructions in [43]. Due to using three input shares, its shared variant has nine quadratic terms, i.e., $a^i b^j$ where $i, j \in \{0, 1, 2\}$. We have to use nine coordinate functions $f^{3i+j}(a^i, b^j)$ each of which contains the corresponding quadratic term.

(1) As the first step, for each $i, j \in \{0, 1, 2\}$, we make the set F_{3i+j}, containing all possible 2-input constant-free coordinate functions for $f^{3i+j}(a^i, b^j)$ whose Algebraic Normal Form (ANF) must include $a^i b^j$. As a result, we have nine different sets, each of which has cardinality four.
(2) In the second step, without loss of generality, let us assume that $f^0(.)$, $f^1(.)$, and $f^2(.)$ are XORed to make an output share y^0. We search for tuples in $F_0 \times F_1 \times F_2$ whose outputs have an identical joint probability distribution which makes them independent of the input a and b, and consequently first-order glitch-extended probing secure. Moreover, the result of the XOR, i.e., y^0, must be a balanced function meaning that it yields as many zeros as ones over its input set. This is a necessary condition to achieve uniformity, as discussed in [30]. The tuples that fulfill both conditions make the set $F^{0,1,2}$. The same procedure should be followed to make two other sets $F^{3,4,5}$ and $F^{6,7,8}$.

(3) The last step is dedicated to find a correct sharing, i.e., $y^0 + y^1 + y^2 = y$. In other words, we search for tuples in $F^{0,1,2} \times F^{3,4,5} \times F^{6,7,8}$ whose XOR yields the unshared output value y. If such a tuple exists, the found solution is a correct, uniform, and glitch-extended probing secure sharing of $x = f(a, b)$.

It is noteworthy to mention that while we considered the case in which $f^0(.)$, $f^1(.)$, and $f^2(.)$ are compressed as given above, there are other configurations that can be taken into account in the search process.

3 AES SHARING DETAILS

In this section, we detail the second-order sharing of the AES S-box, the application of the changing of the guards method, the fresh randomness injected in every round, and the masking of the key schedule.

3.1 S-Box Sharing

The sharing of the S-box uses the tower field decomposition given by Canright [10] and is shown in Figure 4. It is important to note that the randomness which is used in the S-box can be re-used over all the different shared S-boxes. The masking of the S-box follows from the masking of each separate part in the design, namely the input/output isomorphisms, the $GF(16)$ scale-and-multiply function, the $GF(16)$ inverter, the addition of the changing of the guards, and the final multipliers. We go over these components.

Input/Output Isomorphisms. We have used the same functions that are used in the Canright's design [10]. Since these functions are linear, we instantiate them three times and apply them to each shares individually. Note that, we should place a register layer right before the output affine layer to achieve security under the glitch-extended probing model.

$GF(16)$ *Scale-and-Multiply.* The first nonlinear operation in the tower field decomposition consists of a linear scale function and a $GF(16)$ multiplier between the most and least significant nibbles of the input of the S-box. As the scale function is linear, its sharing is done share-wise. For the sharing of the multiplier, we use the technique presented by Shahmirzadi and Moradi [44] to find first-order secure solutions. While the original algorithm in the paper uses two shares, we extended it to three shares. We found several uniform and glitch-extended probing secure constructions for the $GF(16)$ multiplier. We present the one used in this work where the addition and multiplication are the field operations of $GF(16)$ (see Equation (1)). Each coordinate function $f^i(.)$ is second-order non-complete. Under the glitch-extended probing security, each probe in the compression layer is extended to at most three coordinate functions. For example, the probe on y^i is extended to $\{x^{3i}, x^{3i+1}, x^{3i+2}\}$. This set of probed values has an identical joint distribution for every choice of the input secrets. In addition, the sharing is also uniform.

$$
\begin{array}{lll}
f^0(a^0,b^0) = a^0b^0 + b^0 & \rightarrow x^0 & \\
f^1(a^0,b^1) = a^0b^1 + b^1 & \rightarrow x^1 & x^0 + x^1 + x^2 = y^0 \\
f^2(a^0,b^2) = a^0b^2 & \rightarrow x^2 & \\
\hline
f^3(a^1,b^0) = a^1b^0 & \rightarrow x^3 & \\
f^4(a^1,b^1) = a^1b^1 + b^1 & \rightarrow x^4 & x^3 + x^4 + x^5 = y^1 \\
f^5(a^1,b^2) = a^1b^2 + b^2 & \rightarrow x^5 & \\
\hline
f^6(a^2,b^0) = a^2b^0 + b^0 & \rightarrow x^6 & \\
f^7(a^2,b^1) = a^2b^1 & \rightarrow x^7 & x^6 + x^7 + x^8 = y^2 \\
f^8(a^2,b^2) = a^2b^2 + b^2 & \rightarrow x^8 &
\end{array}
\tag{1}
$$

To achieve second-order security, we add fresh masks to our first-order secure construction in a ring-refreshing construction (*i.e.* the shared values $(x^0, x^1, ..., x^8)$ are re-masked to $(x^0 + r^0 + r^1, x^1 + r^1 + r^2, ..., x^8 + r^8 + r^0)$). We should highlight that these fresh masks can be reused in all other S-boxes in all rounds. Note that all component functions are second-order non-complete and fresh masks are added to make the subsequent compression layer second-order glitch-extended probing secure.

$GF(16)$ *Inverter.* We provide a sharing of the $GF(16)$ inverter that is given by the lookup table `0132ED8AF67C495B`. We make a first-order secure construction with three shares using no fresh masks and then add fresh randomness to achieve second-order security. To this end, we decompose the $GF(16)$ inverter into smaller quadratic functions as introduced in [10](see Figure 5a). In this decomposition, the only nonlinear part is the $GF(4)$ multiplier which is a quadratic function. Since it is a quadratic function, we followed the instructions stated in Section 2.5 to find a uniform and probing secure sharing for the $GF(4)$ multiplier. The search led to several solutions, one of which is given in Appendix A. This sharing is used in the masked variant of the last two $GF(4)$ multipliers.

Looking at Figure 5a, the output of the $GF(4)$ inverter should be joint-uniform with the input to achieve a first-order secure construction of the $GF(16)$ inverter. Namely, the output of the $GF(4)$ inverter must be joint-uniform with the upper branch (which are two bits of the input) as they are given to the subsequent upper $GF(4)$ multiplier. The same holds true for the lower $GF(4)$ multiplier, where the output of the $GF(4)$ inverter and the lower branch (which are the two other bits of the input) are its input and must be joint-uniform. We combine all functions in the gray dashed block in Figure 5a including the first $GF(4)$ multiplier, the square-scale function, and the $GF(4)$ inverter to make a 4-bit to 2-bit function denoted by "Square-Scale-Multiplier" in the figure. As a side note, the $GF(4)$ inverter is just a bit permutation, and the square-scale function is linear, making the Square-Scale-Multiplier a quadratic function. We examined all solutions for sharings of the $GF(4)$ multiplier to make a first-order secure inverter. While most of them were first-order secure, none of them led to a joint-uniform solution with its input. As a result, we searched for uniform sharings of the Square-Scale-Multiplier. We found several solutions which were either joint-uniform with the upper branch or joint-uniform with the lower branch but found no solutions which were joint-uniform with both. Hence, we used two different solutions in our construction as shown in Figure 5b where one solution is used to feed the upper $GF(4)$ multiplier and the other feeds the lower $GF(4)$ multiplier. Note that we placed a register after the compression layer of the

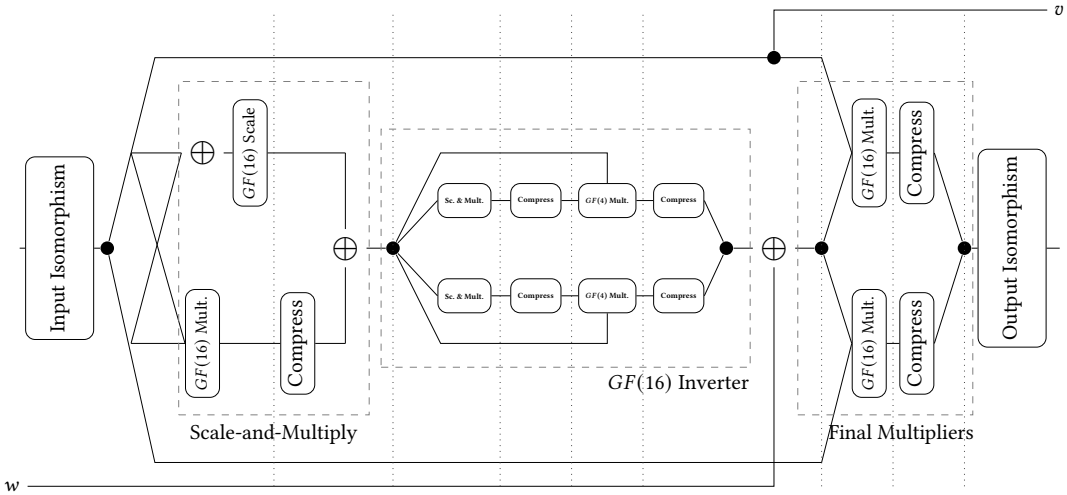
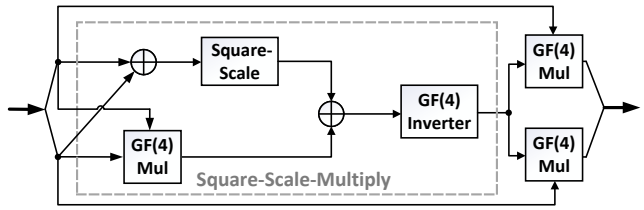
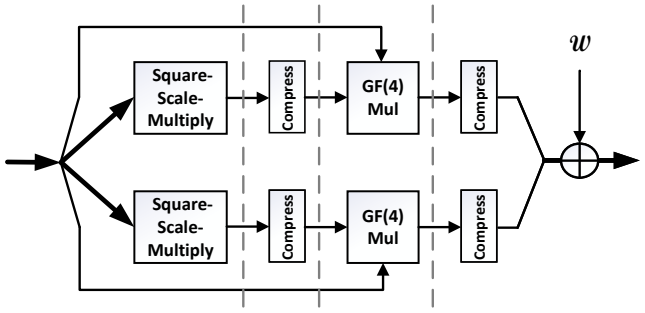


Figure 4: Representation of the second-order shared AES S-box. Register stages are denoted by dotted vertical lines and the guards are denoted by w and v.



(a) unprotected



(b) 1st-order secure

Figure 5: Inversion in $GF(16)$

multipliers. In this way, we constructed a first-order probing secure $GF(16)$ inverter with three shares. The two solutions are given in detail in Appendix B and Appendix C.

Nevertheless, the outputs of the last multipliers are not joint-uniform even though each output is individually uniform. We checked all solutions for the $GF(4)$ multiplier in the construction and none of them led to a joint-uniform solution. So, we used the changing of the guards technique to achieve uniformity at the output of the masked $GF(16)$ inverter. This is explained in detail in Section 3.2.

Similar to the previous stages, we add fresh randomness before the compression of the shares (using a domain-oriented masking style of refreshing) to achieve second-order glitch-extended probing security. Namely, the cross-domain terms are refreshed while other terms with the same domain are left untouched.

Adding and Extracting the Guards. Unlike regular applications of the changing of the guards method, in this sharing the guards are added in the middle of the S-box. The guards consist of two nibbles (w^0, w^1) which are added to the output of the inversion over $GF(16)$. Denoting the bits of the guards as $w^0 = (w_0^0, w_1^0, w_2^0, w_3^0)$ and $w^1 = (w_0^1, w_1^1, w_2^1, w_3^1)$, they are transformed to the vectors $(w_0^i, w_1^i, w_0^i + w_1^i)$ and $(w_2^i, w_3^i, w_2^i + w_3^i)$ for $i \in \{0, 1\}$. This transformation allows for a second-order probing secure conversation to a zero-sharing.

The guards for the next S-box are extracted from the upper limb of the tower field decomposition. We note that the guards should not be taken from the lower limb as this would cause the S-box to be non-uniform as explained in Section 5.3. The reason for this difference is that the sharing for the upper multiplier is different from the sharing of the lower multiplier in the next stage.

The Final Multipliers. Consider the two parallel $GF(16)$ multipliers before the output isomorphism. We follow the notation of the different branches given in Figure 6. Namely, denote the shares of the three input branches of the multipliers by a^i for the upper branch, b^i for the lower branch, and c^i for the middle branch (for $i \in \{0, 1, 2\}$). After the addition of the guards, we denote the shares of the middle branch by d^i. Note that the three shared branches (a^i, b^i, d^i) are joint uniform.

The two parallel $GF(16)$ multipliers follow the same structure as the multiplier in the $GF(16)$ scale-and-multiply function (including the ring refreshing before the compression phase). The multipliers are used such that, after compression, the upper multiplier returns the shares $e^i = a^i d + d^i + d^{i+1}$ (where for ease we neglect the added randomness before the compression). Similarly, after compression,

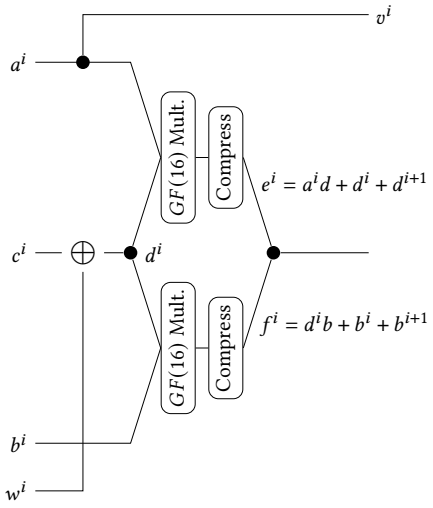


Figure 6: View of the output shares of the final multipliers in the shared AES S-box.

the lower multiplier calculates the shares $f^i = d^i b + b^i + b^{i+1}$. The joint uniformity of the two last multipliers' output is discussed in Section 5.3.

3.2 Changing of the Guards Implemented

In order to guarantee the uniformity of the shared round function and to allow for a first-order probing secure shared S-box without the use of fresh randomness, the changing of the guards technique is applied.

A straightforward application of the technique can cause an overhead in randomness in order to achieve second-order security over multiple rounds as noted by Beyne *et al.* [5]. This is due to the method of chaining the S-boxes causing unwanted diffusion in the shared cipher. For the AES sharing in this work, we chain the S-boxes over the columns as shown in Figure 7. Meaning that we instantiate four cells of guards in order to secure the round function. In the computation of the final S-box of a chain, the guards from the last S-box can be used in the chain of S-boxes in the next round. The motivation for chaining the S-boxes in a column-wise fashion is explained in the security analysis in Section 5.5. In the key schedule, we use an additional cell of guards to chain the four S-boxes per round of the schedule.

3.3 Fresh Randomness per Round

In the AES masking, we inject fresh randomness per round of the state function and the key schedule. More specifically, the randomness is injected before the calculation of the `MixColumns`. To reduce the overhead on the number of random bits used in the sharing, we re-use the random bits over the columns of the state as depicted in Figure 8. Every round, four half-words (r_0, r_1, r_2, r_3) are generated to refresh the S-boxes. The reason for this choice of re-masking is explained in the security analysis of Section 5.5. Essentially, we re-mask the diffusion caused by the changing of the guards and the MixColumns operation.

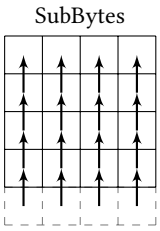


Figure 7: The changing of the guards method over the state. We denote the extra cells needed to store the guards for the next round by gray dashed boxes.

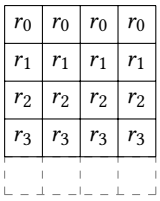


Figure 8: The randomness which is injected at the input of the `MixColumns`.

3.4 Key Schedule

Since the key schedule consists of linear functions and the same S-box as the one used in the state function, its sharing can be copied from the previous sections. In order to avoid leakage over many rounds, the S-boxes in the key schedule are refreshed. However, to reduce the randomness cost, the same randomness which was used to refresh the state can be used in the key schedule of the same round.

4 IMPLEMENTATION AND ARCHITECTURE

Our serialized AES encryption function requires 256 clock cycles to perform a full encryption with the side note that the design can be fed with a new key and plaintext at the same time the ciphertext is produced byte-wise. As a result, the average needed clock cycles for each encryption decreases if the inputs are fed back-to-back. An overview of the data path of the design is depicted in Figure 9. The state registers as well as the key registers are viewed as a 4×4 square array of bytes, and the byte located at row i and column j is denoted as byte $4j + i$. In the first 16 clock cycles, the key and plaintext are loaded byte-wise to the module. Meanwhile, the AddRoundKey is performed and the result is fed into the S-box. Afterwards, four bytes of the key are fed to the S-box in the subsequent four cycles. At this point, the key registers are disabled for four clock cycles waiting for the result of the first key-byte S-box. Once the result of the S-box is ready for the key schedule, the next round's key byte is generated and is added to the corresponding state byte. The corresponding key byte is stored in the key registers to be used in the next round. The ShiftRows is applied when the last state byte comes out of the S-box (this is not shown in Figure 9 to ease the diagram). In the next clock cycle, the MixColumns operation is performed in parallel to the refreshing. Namely, the input of the MixColumns is refreshed by a total of 64-bits of fresh masks. The

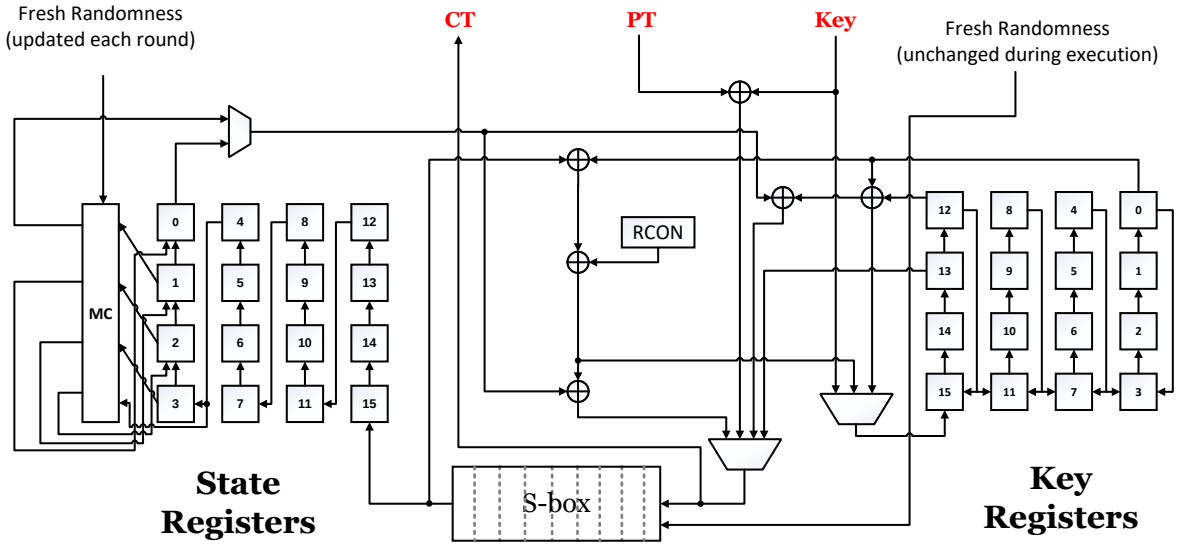


Figure 9: Our design of the serialized AES encryption.

same fresh masks can be used for the other columns in the round (including the column of the key schedule). So, these 64-bit fresh masks should be updated each round apart from the last round, where the MixColumns operation is missing.

As stated in Section 3.2, each column is chained to apply the changing of the guard technique. Since we have four columns for the state registers and one column for the key registers, the AES module needs $5 \times 8 = 40$ bits of fresh randomness for the guards. They should be given to the module at the start of the encryption and no extra costs are required to instantiate multiple rounds as the required guard bits of the last byte of each column are stored in registers to be used in the next round. So, in the corresponding clock cycle, the guards are given to the S-box construction followed by chaining the other three bytes in the column. Note that, apart from 40 bits of randomness for the guards, 132 fresh masks should be provided at the start of the encryption to be used in the S-box calculations in all rounds.

As a result, our design requires a total of 1260 random bits per encryption including the sharing of the plaintext and key. More specifically, we require 256 bits for the sharing of the plaintext, 256 bits for the key, 40 bits for the instantiation of the guards, 132 to make the S-box second-order secure, and a total of 576 bits to refresh the columns.

We refer to Table 1 for the comparison of our design to the state-of-the-art. We synthesized our design using the Nangate 45nm Library with the Synopsis Design Compiler tool to have a fair comparison. We also used a compile option avoiding any optimization across modules for security reasons. Note that, the fresh masks needed for sharing the plaintext and key are included in the table. A second-order secure AES implementation is introduced in [13], where six input shares are used with a high randomness cost. As a result, the area overhead of the S-box is significant. Using Boolean masking, the authors of [18] provided a second-order secure AES with a high randomness complexity. The implementation costs are reduced in [25] while using the same masking scheme. Later, the area overhead has been reduced using multiplicative masking at the cost of a slightly higher latency with roughly the same randomness complexity [19]. Notably, our design requires significantly fewer fresh masks per encryption (more than nine times less compared to the best published design) with the same latency (# of clock cycles) and roughly the same range of area overhead. Similar to other works in the field, the area overhead for fresh mask generation is excluded from the table.

Table 1: Implementation results for AES-128.

(using Synopsis Design Compiler, and Nangate 45nm Library, excluding RNGs)

Design	Security Order	No. of Shares	Rand/ Enc. [bits]	Area [GE]	Latency [cycles]
[13] [a]	2	6	25 712	11 174	-
[18]	2	3	32 912	12 640	276
[25]	2	3	11 312	12 024	246
[19]	2	3	11 112	10 931	256
This work	2	3	1 268	12 880	256

[a] Just a single S-box

5 GLITCH-EXTENDED PROBING SECURITY

In this section, we show that the AES masking detailed in Section 3 is second-order glitch-extended probing secure as introduced in Section 2.2. This is proven in several steps. First, by showing that the S-box is itself second-order secure in Section 5.1. Then, by showing that the round function is uniform in Section 5.3 and that a single round is second-order secure (recall that if the round function is both probing secure and uniform, the entire AES masking is already first-order secure since it follows the threshold implementation properties explained in Section 2.3). Finally, in Section 5.5, we show

that the entire AES masking is second-order secure by including a multi-round probing analysis.

5.1 S-Box Verification via SILVER

SILVER [30] is a formal verification tool, designed to evaluate a given circuit under different security notions. Namely, it assesses the security of a given design based on the models to avoid writing proofs for every design. It receives the netlist of the hardware design and verifies it without any simplification. Consequently, its results are reliable with no false negative or false positive considering the model. It performs an exhaustive analysis of the probability distributions on each gate and wire of the netlist and proves the statistical independence of subsets of the wires on the input secrets of this list. We evaluated our S-box construction under the glitch-extended probing model dedicated to hardware platforms. We synthesized our construction by Synopsis Design Compiler using NanGate 45 ASIC standard cell library and generated the netlist. We disabled any optimization across different module to fulfill non-completeness property.

First-Order Probing Security Without Randomness. As the first step, we removed all fresh masks from our S-box construction and evaluated the design with SILVER. We confirmed the first-order glitch-extended probing security of the design as well as the uniformity of the output. It took around 20 minutes on a machine with 16 CPU cores with 128 GB of RAM.

The reason behind this step is the fact that any fresh masks required for first-order security of an intermediate variable cannot be reused in a second-order secure design. In a second-order attack, an adversary can probe two intermediate values. Therefore, in case of reusing a fresh mask needed for first-order security, the adversary can observe the fresh mask with one probe and with the other probe observes the intermediate value whose security depends on the fresh masks. In this step, we demonstrated that our design needs no fresh masks for first-order security and all fresh masks are needed to achieve second-order security under glitch-extended probing model. This enable us to reuse these randomness, reducing the total number of required fresh masks.

Second-Order Probing Security With Randomness. SILVER cannot handle a large design due to high number of individual statistical test leading to unrealistic computational complexities. As a result, we could not evaluate the entire S-box construction as the number of inputs increases with fresh masks. We first confirmed the second-order security of the $GF(16)$ multiplier as well as the $GF(16)$ inverter under glitch-extended probing model. Then, we split the design in two parts. The first part encompasses all operations from the S-box's input to the $GF(16)$ inverter. The second part encompasses the two last multipliers. We confirmed the second-order security of both parts. Since both parts are second-order secure and we placed a register layer after the $GF(16)$ inverter which is refreshed with fresh masks, we can conclude that the entire design is second-order secure. In other words, we introduced the fresh masks r_1 to make the first part second-order secure and evaluated its second-order security with SILVER. Then, we added completely different fresh masks r_2 to the second part and confirmed its second-order security with SILVER as well. We refreshed the output shares of the first part with fresh masks r_3 and stored the result in registers. These registered values are given to the second-part. This refreshing imposes the independence of the composed elements as discussed in [40], which can be seen as a naive method to make a composable gadget. Hence, we claim the second-order security of our S-box construction.

It took around 100 hours to verify the security of first part on the same machine that was used for the first-order security evaluation. The second part was done in around 60 hours. In the next subsection, we evaluate the entire S-box construction using a recently-published simulation-based verification tool as well to support our claim of second-order security of our masked S-box. Since no formal verification tool can currently analyze the entire encryption function, we include theoretical arguments for probing security across several rounds as described in Section 5.5 and we conducted experimental analyses on FPGA as described in Section 6.

5.2 S-Box Verification via PROLEAD

Since SILVER can at most evaluate subcircuits (for example small S-boxes), particularly in higher orders, we also evaluated our S-box construction with PROLEAD [37]. PROLEAD is an automated simulation-based tool to analyze the statistical independence of simulated intermediates probes under the robust probing model for the given masked implementation. It can handle larger circuits and similar to SILVER, it works directly on the given netlist performing logic simulation and not on simulating power traces. As a result, the security evaluation is not based on hypothetical power model.

Similar to SILVER verification, we used Synopsis Design Compiler and NanGate 45 ASIC standard cell library to generate the netlist. We ran the evaluation under glitch-extended probing model on a machine with an AMD EPYC 7352 with 48 hyper-threading cores and 448 GB of memory. The tool performed fixed versus random G-test, where the fixed input is zero vector. The required fresh masks can be set to refreshed each clock cycle or do not change during the execution. We used the latter option as the required fresh masks for the S-box remain unchanged during the execution of the cipher in our design. No leakage has been reported with 100 million simulations neither in univariate nor in multivariate evaluation including all probing sets, confirming second-order security of our masked AES S-box. The evaluation took around 28 hours on the aforementioned machine. The evaluation results reported by the tool is also included in the supplementary materials attached to this submission.

We should highlight that PROLEAD is based on simulations and statistical hypothesis tests and it cannot prove the security of a given design. As the number of simulations increases the evaluation results would be more reliable. In other words, the probability that existing leakage is not detected decreases when a higher number of simulations are considered in the evaluations. We believe that 100 million simulations is large enough to provide high level of confidence.

5.3 Uniformity of the Round Function

In Section 5.1, we have verified the uniformity of the masked S-box using SILVER. However, to add to the motivation of the masked design, we also prove it by pen-and-paper methods. More specifically,

we show that the function shown in Figure 6 is uniform (recall that c^i is a function of a^i and b^i). Proving the uniformity of a construction with the same number of inputs and outputs is equivalent to showing the shared construction is invertible for every choice of input secrets. Thus, we need to prove that the final multipliers with the changing of the guards are invertible for every choice of the input secrets. In other words, for every input secret (a, b), the shares of (a, b, w) are a function of the shares of (v, e, f) (recall that from the tower-field construction c is a function of a and b).

In a nutshell, the final multipliers form a "two-staged Feistel" construction. The middle branch (c) is multiplied by a secret and added to the lower branch (b). In turn, the upper branch (a) is multiplied by a secret and added to the middle branch. Since the construction is an iterated Feistel construction, it is invertible making it uniform.

Pick arbitrary input secrets (a, b) of the masked S-box. From the secret a and the output guards v, we can reconstruct the shares of a, namely

$$a^0 = v^0, a^1 = v^1, a^2 = v^0 + v^1 + a\,.$$

Given the shares of a, the shares of the output e, and the secret d, we can reconstruct the shares of d, namely

$$d^i = e^{i+2} + a^{i+2}d + d\,.$$

From the shares of d, the shares of the output f, and the secret b, we can reconstruct the shares of b, namely

$$b^i = f^{i+2} + d^{i+2}b + b\,.$$

Recall that the shares of c are a function of the shares of a, b (because of the tower-field construction). By summing the shares of c and d, we reconstruct the input guards w^i. Thus, the construction is invertible for any choice of input secrets (a, b) which implies the uniformity of the construction. As a result, the round function is uniform.

5.4 Single Round Probing Analysis

We consider the second-order glitch-extended probing security of a single round of the masked AES. From Section 5.3, we know that the round function is uniform. As a result, the input state of each round is a joint uniform masking.

We split the proof up in four cases.

- Case 1: both probes are placed in the same masked S-box. In this case the security comes from the second-order probing security of the S-box itself which was verified by SILVER in Section 5.1.
- Case 2: the probes are placed in different masked S-boxes. First, from Section 5.3, we know that both S-boxes have a joint-uniform input. From Section 5.1, we know that each S-box is first-order secure without any added randomness. Thus, by removing the additional randomness, the two S-boxes do not share joint information and, as a result, the two probes provide independent information. Since each S-box is first-order probing secure, the probe does not provide information on a secret value. As a result, this case is also secure.
- Case 3: the probes are placed in the linear layer and a masked S-box. The probe on the linear layer provides one share of the output of the S-box and views no added randomness. As a result, the security comes from the first case. In addition, the outputs of the other S-boxes viewed in the linear layer act as uniform randomness since each S-box is itself uniform (and thus provide a joint uniform output).
- Case 4: both probes are placed in the linear layer. Since this layer works share-wise, this is evidently secure.

5.5 Multiple Round Probing Analysis

We now consider that two probes are placed in two different rounds of the shared AES. Recall that since Section 5.4 already handled the case where the two probes are placed in the same round, proving this multi-round case proves the second-order probing security of the entire AES masking as it is the last remaining case of placement for the probes.

We argue that the fresh randomness, which is injected every round (as discussed in Section 3.3), offers the multi-round protection. The proof is made by evaluating what an adversary can observe by placing a probe in a masked S-box or the linear layer. We mark (visually) what the adversary has observed via a probe in the state. We call this a *pattern*. The observed values (or pattern) are then propagated through the diffusion layers (including the diffusion caused by the changing of the guards method). This causes the pattern which the adversary has observed to change. We propagate this until we reach the end of the ShiftRows operation where we can overlap the resulting pattern with the randomness which is added before the `MixColumns` operation as shown in Figure 8. From there, we can show that the added randomness completely masks all values which were observed by the probe, meaning the values observed by the adversary are added with independent randomness before the values are re-observed by the second probe.

Lemma 5.1. *The activity patterns caused by a glitch-extended probe (up to symmetry, the ones shown in Figure 10) are masked by the randomness of the masked AES design (the randomness shown in Figure 8).*

Proof. We show that when one probe is placed in the round function of the masked AES, the randomness before the `MixColumns` refreshes what is observed. We split the reasoning in two parts, a probe is placed in the linear layer and a probe is placed in the masked S-box.

- We consider the case where a probe is placed in the linear layer. From the design of AES's linear layer, this probe can view at most one column of the state. The observed values then pass the masked S-box layer which, due to the changing of the guards method, spreads these values out to five cells in a column of the state (an example is shown in Figure 10). These cells then pass the ShiftRows operation. However, only one cell per row remains active. As a result, the added random bits (r_0, r_1, r_2, r_3) refresh the activity pattern in the state. But the refreshing leaves an unmasked cell containing the guards for the next round. Note that these guards have not passed a nonlinear layer, thus probing this cell in the next round does not provide any additional information. The remaining active cell is again diffused to a column of the state where it is fully re-masked by that round's fresh randomness.

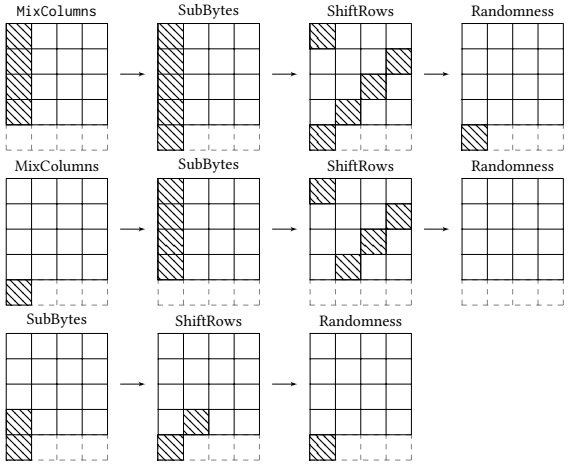


Figure 10: Three examples of the diffusion of observed values from probing. The "randomness" step denotes the addition of randomness following Figure 8.

- We consider the case where a probe is placed in the masked S-box which results in the adversary viewing the input of two S-boxes due to the changing of the guards method (an example is shown in Figure 10). Again, this reverts to the previous case where the randomness (r_0, r_1, r_2, r_3) re-masks the pattern over the following two rounds.

□

5.6 Second-Order Robust Probing Security

The previous subsections allow for the proof of second-order probing security.

Theorem 5.2. *The masked AES design from Section 3 is second-order glitch-extended probing secure.*

Proof. Consider a second-order probing adversary. We split the proof of security in cases based on where the probes are placed.

For the first case, we consider that both probes are placed in a single round of the masked AES state function. The probing security of this case is given by the analysis in Section 5.4.

For the second case, we consider that both probes are placed in different rounds of the masked AES state function. From Lemma 5.1, we find that the values observed by both probes are independently distributed. From Section 5.4 and from the round function being uniform (as shown in Section 5.3), we find that each separate probe only observes uniform random distributed values or public values (such as round constants). As a result, both probes jointly observe uniform randomness or public values which proves the probing security of this case.

Finally, the masked key schedule is also second-order probing secure as all the non-linear operations are refreshed and the masked S-box design has the same design as in the state function (shown secure in Section 5.1). Similarly, if one probe is placed in the state function and one in the key schedule. Since the values observed after the S-box are refreshed, the probes view independent distributed values and each separate probe only views uniform randomness or public values.

Since all cases are proven secure, we conclude that the masked AES design is second-order glitch-extended probing secure as given in Section 2.2. □

6 EXPERIMENTAL VERIFICATION

As mentioned in Section 5.1, we evaluated the security of our S-box construction using the formal verification tool SILVER [30] under the glitch-extended probing model. As currently no tool can verify the security of an entire encryption function, we have conducted experimental analyses in addition to the theoretical analyses. We use a byte-serial AES-128 implementation where 132 bits of fresh masks for the S-box and 40 bits for the guards are required. These fresh masks stay constant during an execution of the cipher. The relevant part of the circuit for the generation of these fresh randomness is activated only one clock cycle at the start of each encryption. Moreover, 64 bits of fresh masks are needed to re-mask the columns, which are updated only per round.

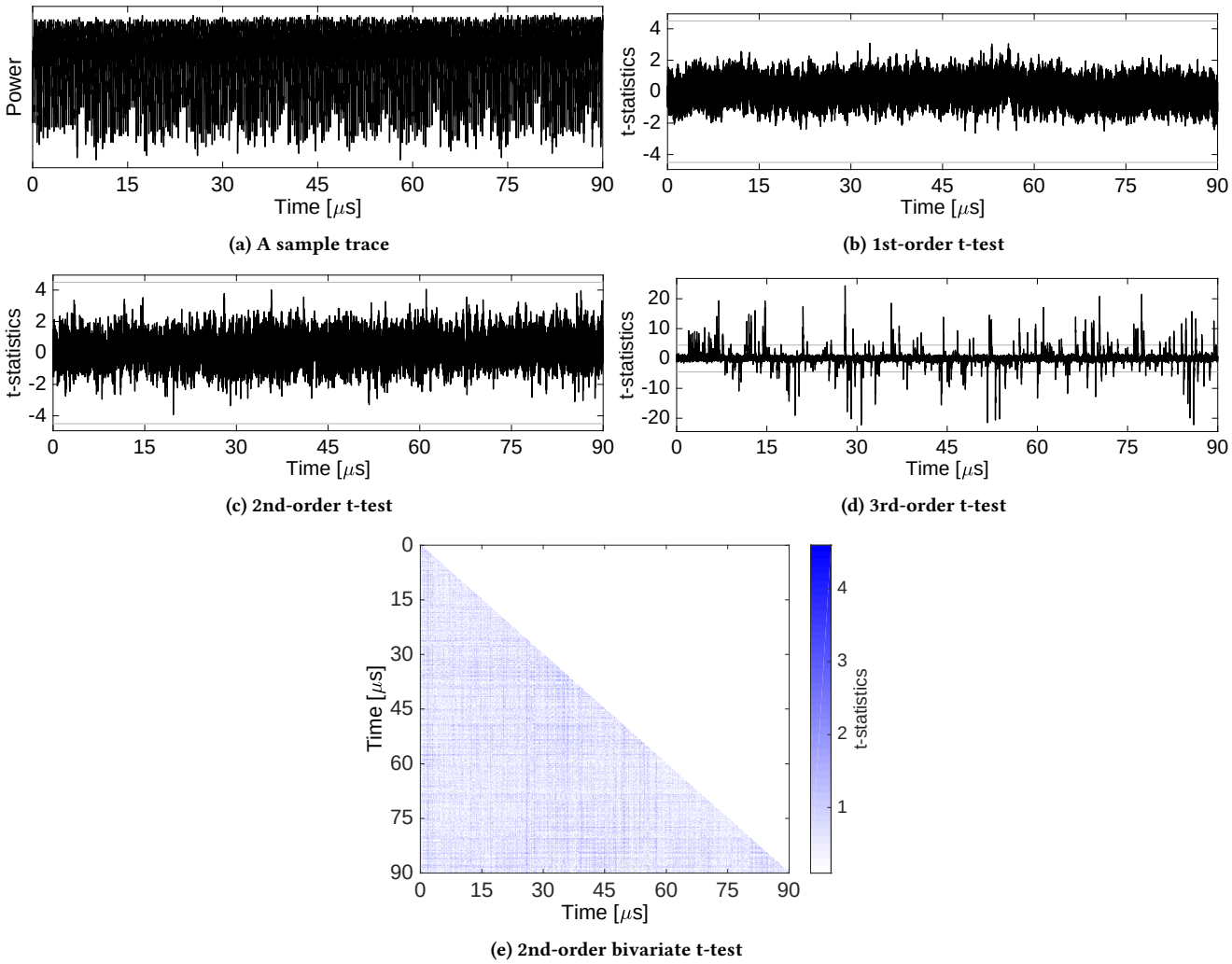
We implemented this design on a Xilinx Spartan-6 FPGA of the SAKURA-G evaluation board [**?**]. As the source of the clock, a stable 6 MHz oscillator was used. A Linear Feedback Shift Register (LFSR) with the feedback polynomial $x^{31} + x^{28} + 1$ is instantiated for each required fresh mask bit. It has been shown that the implementation of the LFSR can be efficient in Xilinx FPGAs as only three 6-to-1 Look-Up Tables (LUTs) are needed [47]. In addition, our design provides a single bit control signal indicating the end of each round which we used to update those 64-bit fresh masks. Our design uses no fresh masks at the last round of the cipher as the addition of these fresh masks are integrated into the MixColumns operation.

The control FPGA provides a three-share masked plaintext and a masked key for the target FPGA and observes the ciphertext in a three-share masked form. Note that, it is a byte-serial implementation, and the control FPGA feeds the target FPGA byte-wise and receives the ciphertext byte-wise as well.

The power consumption traces were collected by measuring the voltage drop over a 1 Ω shunt resistor placed on the VDD path of the target FPGA. A digital oscilloscope at a sampling rate of 500 MS/s has been used to measure the voltage drop. Figure 11a shows a sample power trace covering an entire encryption, where a certain pattern can be observed indicating the cipher rounds.

Following the measurement strategy explained by Schneider and Moradi in [42], we have conducted a fixed-versus-random t-test using 100 million traces. Such an analysis is supposed to detect SCA leakages without performing any key-recovery attack. In this test, which is also known as TVLA [14], the key is fixed to a certain value during the measurements while based on a single bit random value, either a fixed or a random plaintext is given to the design. The plaintext and the key are given to the design in a three-share uniform masked form regardless of the decision, i.e., whether a fixed plaintext or a random one should be given to the circuit.

We performed four different analyses to truly evaluate our design in practice. The first analysis is a *first-order univariate test* in which a t-test is applied to each power consumption sample point individually. To perform a *second-order univariate test*, we made the power traces mean-free for each group of fixed and random



(a) A sample trace

(b) 1st-order t-test

(c) 2nd-order t-test

(d) 3rd-order t-test

(e) 2nd-order bivariate t-test

Figure 11: Experimental analysis of our masked AES using 100 million traces.

individually. Afterwards, we squared each mean-free sample point and ran the ordinary t-test on the pre-processed traces. The same procedure was followed for a *third-order univariate test*, where each mean-free sample was cubed instead of squared.

For a *bivariate second-order t-test*, each combination of two mean-free sample points should be multiplied followed by an individual t-test. However, performing such a high number of individual tests is not feasible in practice, particularly in our case where every power trace has many sample points. Therefore, we also utilized the downsampling trick that has been used in several other publications including [4, 18, 44]. For each clock cycle, we took four samples that are carefully selected such that they are spread over the start, middle, and end of the cycle. This enables us to perform the analysis on all clock cycles involved in the power traces covering full encryptions. Note that, power consumption traces are low-pass filtered by the Printed Circuit Board (PCB), shunt resistor, measurement facilities, etc., as discussed in [32]. As a result, several neighboring samples points at each clock cycle contain the same information about the corresponding leakage. Considering only a few sample points per clock cycle should be adequate for such a bivariate analysis. More information and discussion can be found in [35].

The results of our experimental analyses are shown in Figure 11, where we detected no univariate or bivariate first- and second-order leakage. As presented in Figure 11d, the design exhibits third-order leakage, as expected. This further confirms the validity of our setup. Note that, we omitted the third-order multivariate analysis as the third-order univariate t-test already showed detectable leakage.

7 CONCLUSIONS AND DISCUSSIONS

In this paper, we presented a second-order masking of the AES which was implemented on FPGA and tested in practice. From the lab analysis we find no first- or second-order leakage. The security of the masking was also proven in the probing security model, where the tool SILVER reported the second-order probing security of the S-box design. Additionally, the tool and a pen-and-paper

proof verified the uniformity of the construction revealing our motivation for the sharing. Lastly, a pen-and-paper proof was given to argue why multi-round leakage is not possible as all probed information are re-masked between the rounds. The result is a masking which solves the open problem posed by De Cnudde *et al.* [18] of reducing the randomness cost of a masked tower-field decomposed AES S-box. It has been repeatedly demonstrated that the cost of fresh masks generation is high in hardware platforms [9, 28]. In this work, we have an over eight-fold reduction of the total randomness cost including the cost of sharing the plaintext and key. This reduction comes at a cost of a slight increase of the area footprint. More precisely, this increase is less than 20% when compared to the best result reported in the state-of-the-art and less than 10% compared to other masked designs based on the tower-field decomposition.

In the work, we encountered several challenges which seem highly interesting for future works. The bulk increase in area overhead comes from a "duplication method" used in the $GF(16)$ inverter. We thus pose the open problem to find a more efficient first-order probing secure sharing of the inverter without using randomness. As a second line of future works to improve the sharing, we found from the works by Beyne *et al.* [4, 6] that it is possible to achieve second-order secure sharing without adding fresh randomness per round of the cipher. However, for this technique to work, the S-box sharing requires a high nonlinearity and the diffusion caused by the changing of the guards method must be taken into account. Our sharing of the AES S-box has a trivial shared nonlinearity which was necessary to ensure its uniformity. As a result, a nonlinear uniform sharing of the tower-field decomposed S-box would be an interesting future work further decreasing the cost of randomness.

Finally, it is noteworthy to mention that reducing the randomness complexity of a design may decrease the level of noise on the actual leakage. This may increase the risk of horizontal attacks [2, 26], where the information of multiple target intermediate values are combined to reduce the noise of the actual leakages measured by an adversary. However, it has been shown that it is quite challenging to apply horizontal attacks in a close-source setting, where an adversary has no information about implementation details [11]. Further, horizontal attacks are relevant as soon as the implementation is secure at a high order (typically for sixth-order secure designs and higher) as discussed in [11]. Nevertheless, it would interesting to analyze the security of the construction in this paper against horizontal attacks as future work.

ACKNOWLEDGEMENTS

The work described in this paper has been supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972 and through the project 406956718 SuCCESS. Siemen Dhooghe is supported by a PhD Fellowship from the Research Foundation – Flanders (FWO).

REFERENCES

[1] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. 2019. maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults. In *ESORICS 2019 (Lecture Notes in Computer Science, Vol. 11735)*. Springer, Berlin, 300–318.

[2] Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. 2016. Horizontal Side-Channel Attacks and Countermeasures on the ISW Masking Scheme. In *CHES 2016 (Lecture Notes in Computer Science, Vol. 9813)*. Springer, 23–39.

[3] Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. 2018. Tight Private Circuits: Achieving Probing Security with the Least Refreshing. In *ASIACRYPT 2018 (Lecture Notes in Computer Science, Vol. 11273)*. Springer, Berlin, 343–372.

[4] Tim Beyne, Siemen Dhooghe, Amir Moradi, and Aein Rezaei Shahmirzadi. 2022. Cryptanalysis of Efficient Masked Ciphers: Applications to Low Latency. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022, 1 (2022), 679–721.

[5] Tim Beyne, Siemen Dhooghe, Adrián Ranea, and Danilo Sijacic. 2021. A Low-Randomness Second-Order Masked AES. *IACR Cryptol. ePrint Arch.* (2021), 1111. https://eprint.iacr.org/2021/1111

[6] Tim Beyne, Siemen Dhooghe, and Zhenda Zhang. 2020. Cryptanalysis of Masked Ciphers: A Not So Random Idea. In *ASIACRYPT 2020, Part I (LNCS)*. Springer, Heidelberg, 817–850. https://doi.org/10.1007/978-3-030-64837-4_27

[7] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. 2014. Higher-Order Threshold Implementations. In *ASIACRYPT 2014, Part II (LNCS, Vol. 8874)*, Palash Sarkar and Tetsu Iwata (Eds.). Springer, Heidelberg, 326–343. https://doi.org/10.1007/978-3-662-45608-8_18

[8] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. 2012. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In *ASIACRYPT 2012 (Lecture Notes in Computer Science, Vol. 7658)*. Springer, Berlin, 208–225.

[9] Christophe De Cannière. 2006. Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In *ISC (Lecture Notes in Computer Science, Vol. 4176)*. Springer, Berlin, 171–186.

[10] D. Canright. 2005. A Very Compact S-Box for AES. In *CHES 2005 (LNCS, Vol. 3659)*, Josyula R. Rao and Berk Sunar (Eds.). Springer, Heidelberg, 441–455. https://doi.org/10.1007/11545262_32

[11] Gaëtan Cassiers and François-Xavier Standaert. 2019. Towards Globally Optimized Masking: From Low Randomness to Low Noise Rate or Probe Isolating Multiplications with Reduced Randomness and Security against Horizontal Attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019, 2 (2019), 162–198.

[12] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. 1999. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings (Lecture Notes in Computer Science, Vol. 1666)*, Michael J. Wiener (Ed.). Springer, Berlin, 398–412. https://doi.org/10.1007/3-540-48405-1_26

[13] Thomas De Cnudde, Begül Bilgin, Oscar Reparaz, Ventzislav Nikov, and Svetla Nikova. 2015. Higher-Order Threshold Implementation of the AES S-Box. In *CARDIS 2015 (Lecture Notes in Computer Science, Vol. 9514)*. Springer, Berlin, 259–272.

[14] Jeremy Cooper, Elke DeMulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, Pankaj Rohatgi, et al. 2013. Test vector leakage assessment (TVLA) methodology in practice. In *International Cryptographic Module Conference*, Vol. 20.

[15] Jean-Sébastien Coron, Aurélien Greuet, and Rina Zeitoun. 2020. Side-Channel Masking with Pseudo-Random Generator. In *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 12107)*. Springer, 342–375.

[16] Joan Daemen. 2017. Changing of the Guards: A Simple and Efficient Method for Achieving Uniformity in Threshold Sharing. In *CHES 2017 (LNCS, Vol. 10529)*, Wieland Fischer and Naofumi Homma (Eds.). Springer, Heidelberg, 137–153. https://doi.org/10.1007/978-3-319-66787-4_7

[17] Joan Daemen and Vincent Rijmen. Nov. 2001. Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce.

[18] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. 2016. Masking AES with d+1 Shares in Hardware. In *CHES 2016 (LNCS, Vol. 9813)*, Benedikt Gierlichs and Axel Y. Poschmann (Eds.). Springer, Heidelberg, 194–212. https://doi.org/10.1007/978-3-662-53140-2_10

[19] Lauren De Meyer, Oscar Reparaz, and Begül Bilgin. 2018. Multiplicative Masking for AES in Hardware. *IACR TCHES* 2018, 3 (2018), 431–468. https://doi.org/10.13154/tches.v2018.i3.431-468 https://tches.iacr.org/index.php/TCHES/article/view/7282.

[20] Siemen Dhooghe and Svetla Nikova. 2022. Resilient uniformity: applying resiliency in masking. *Cryptogr. Commun.* 14, 1 (2022), 41–58. https://doi.org/10.1007/s12095-021-00515-w

[21] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. 2018. Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, 3 (2018), 89–120.

[22] Sebastian Faust, Clara Paglialonga, and Tobias Schneider. 2017. Amortizing Randomness Complexity in Private Circuits. In *Advances in Cryptology - ASIACRYPT*

2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10624). Springer, 781–810.

[23] Louis Goubin and Jacques Patarin. 1999. DES and Differential Power Analysis (The "Duplication" Method). In *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings (Lecture Notes in Computer Science, Vol. 1717)*, Çetin Kaya Koç and Christof Paar (Eds.). Springer, Berlin, 158–172. https://doi.org/10.1007/3-540-48059-5_15

[24] Hannes Groß, Stefan Mangard, and Thomas Korak. 2016. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In *Theory of Implementation Security - TIS@CCS 2016*. ACM, 3.

[25] Hannes Groß, Stefan Mangard, and Thomas Korak. 2017. An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order. In *CT-RSA 2017 (LNCS, Vol. 10159)*, Helena Handschuh (Ed.). Springer, Heidelberg, 95–112. https://doi.org/10.1007/978-3-319-52153-4_6

[26] Vincent Grosso and François-Xavier Standaert. 2018. Masking Proofs Are Tight and How to Exploit it in Security Evaluations. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II (Lecture Notes in Computer Science, Vol. 10821)*. Springer, 385–412.

[27] Yuval Ishai, Amit Sahai, and David A. Wagner. 2003. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO 2003 (Lecture Notes in Computer Science, Vol. 2729)*. Springer, Berlin, 463–481.

[28] David Knichel and Amir Moradi. 2022. Composable Gadgets with Reused Fresh Masks First-Order Probing-Secure Hardware Circuits with only 6 Fresh Masks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022, 3 (2022), 114–140.

[29] David Knichel, Pascal Sasdrich, and Amir Moradi. 2020. SILVER - Statistical Independence and Leakage Verification. In *ASIACRYPT 2020 (Lecture Notes in Computer Science, Vol. 12491)*. Springer, Berlin, 787–816.

[30] David Knichel, Pascal Sasdrich, and Amir Moradi. 2020. SILVER - Statistical Independence and Leakage Verification. In *ASIACRYPT 2020, Part I (LNCS)*. Springer, Heidelberg, 787–816. https://doi.org/10.1007/978-3-030-64837-4_26

[31] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential Power Analysis. In *CRYPTO 1999 (Lecture Notes in Computer Science, Vol. 1666)*. Springer, Berlin, 388–397.

[32] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. 2007. *Power analysis attacks - revealing the secrets of smart cards*. Springer, Berlin.

[33] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. 2005. Successfully Attacking Masked AES Hardware Implementations. In *CHES 2005 (Lecture Notes in Computer Science, Vol. 3659)*. Springer, Berlin, 157–171.

[34] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. 2019. Glitch-Resistant Masking Revisited or Why Proofs in the Robust Probing Model are Needed. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019, 2 (2019), 256–292.

[35] Amir Moradi and Oliver Mischke. 2013. On the Simplicity of Converting Leakages from Multivariate to Univariate - (Case Study of a Glitch-Resistant Masking Scheme). In *CHES 2013 (LNCS, Vol. 8086)*, Guido Bertoni and Jean-Sébastien Coron (Eds.). Springer, Heidelberg, 1–20. https://doi.org/10.1007/978-3-642-40349-1_1

[36] Amir Moradi, Oliver Mischke, and Thomas Eisenbarth. 2010. Correlation-Enhanced Power Analysis Collision Attack. In *CHES 2010 (Lecture Notes in Computer Science, Vol. 6225)*. Springer, Berlin, 125–139.

[37] Nicolai Müller and Amir Moradi. 2022. Glitch-Resistant Masking Revisited or Why Proofs in the Robust Probing Model are Needed. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022, 4 (2022).

[38] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. 2006. Threshold Implementations Against Side-Channel Attacks and Glitches. In *ICICS 06 (LNCS, Vol. 4307)*, Peng Ning, Sihan Qing, and Ninghui Li (Eds.). Springer, Heidelberg, 529–545.

[39] Oscar Reparaz. 2015. A note on the security of Higher-Order Threshold Implementations. *IACR Cryptol. ePrint Arch.* 2015 (2015), 1.

[40] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. 2015. Consolidating Masking Schemes. In *CRYPTO 2015, Part I (LNCS, Vol. 9215)*, Rosario Gennaro and Matthew J. B. Robshaw (Eds.). Springer, Heidelberg, 764–783. https://doi.org/10.1007/978-3-662-47989-6_37

[41]]sakura SAKURA. [n. d.]. Side-channel Attack User Reference Architecture. http://satoh.cs.uec.ac.jp/SAKURA/index.html.

[42] Tobias Schneider and Amir Moradi. 2015. Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations. In *CHES 2015 (LNCS, Vol. 9293)*, Tim Güneysu and Helena Handschuh (Eds.). Springer, Heidelberg, 495–513. https://doi.org/10.1007/978-3-662-48324-4_25

[43] Aein Rezaei Shahmirzadi and Amir Moradi. 2020. Re-Consolidating First-Order Masking Schemes - Nullifying Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 1 (2020), 305–342.

[44] Aein Rezaei Shahmirzadi and Amir Moradi. 2021. Second-Order SCA Security with almost no Fresh Randomness. *IACR Cryptol. ePrint Arch.* 2021 (2021), 461.

[45] Takeshi Sugawara. 2019. 3-Share Threshold Implementation of AES S-box without Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019, 1 (2019), 123–145. https://doi.org/10.13154/tches.v2019.i1.123-145

[46] Weijia Wang, Chun Guo, François-Xavier Standaert, Yu Yu, and Gaëtan Cassiers. 2020. Packed Multiplication: How to Amortize the Cost of Side-Channel Masking?. In *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12491)*. Springer, 851–880.

[47] Felix Wegener, Lauren De Meyer, and Amir Moradi. 2020. Spin Me Right Round Rotational Symmetry for FPGA-Specific AES: Extended Version. *Journal of Cryptology* 33, 3 (July 2020), 1114–1155. https://doi.org/10.1007/s00145-019-09342-y

[48] Felix Wegener and Amir Moradi. 2018. A First-Order SCA Resistant AES Without Fresh Randomness. In *Constructive Side-Channel Analysis and Secure Design - 9th International Workshop, COSADE 2018, Singapore, April 23-24, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10815)*, Junfeng Fan and Benedikt Gierlichs (Eds.). Springer, Berlin, 245–262. https://doi.org/10.1007/978-3-319-89641-0_14

A 3-SHARE $GF(4)$ MULTIPLIER USING 6 BITS OF FRESH MASKS

$F(a,\ b,\ c,\ d) = (x,\ y)$

$x = f(a,b,c,d) = bc + ad + bd$

$y = g(a,b,c,d) = ac + bc + ad$

The construction is first-order secure without fresh masks.

$f^0(a^0,b^0,c^0,d^0) = a^0 + d^0 + b^0c^0 + a^0d^0 + b^0d^0$	$\rightarrow x'^0$	
$f^1(a^1,b^1,c^0,d^0) = b^1 + c^0 + b^1c^0 + a^1d^0 + b^1d^0 + c^0d^0 + r^0$	$\rightarrow x'^1$	$x'^0 + x'^1 + x'^2 = x^0$
$f^2(a^2,b^2,c^0,d^0) = a^2 + b^2 + c^0 + d^0 + b^2c^0 + a^1d^0 + b^2d^0 + c^0d^0 + r^1$	$\rightarrow x'^2$	
$f^3(a^0,b^0,c^1,d^1) = b^0 + b^0c^1 + a^0d^1 + b^0d^1 + r^0$	$\rightarrow x'^3$	
$f^4(a^1,b^1,c^1,d^1) = b^1c^1 + a^1d^1 + b^1d^1$	$\rightarrow x'^4$	$x'^3 + x'^4 + x'^5 = x^1$
$f^5(a^2,b^2,c^1,d^1) = a^2 + b^2 + b^2c^1 + a^2d^1 + b^2d^1 + r^2$	$\rightarrow x'^5$	
$f^6(a^0,b^0,c^2,d^2) = a^0 + b^0 + b^0c^1 + a^0d^2 + b^0d^2 + r^1$	$\rightarrow x'^6$	
$f^7(a^1,b^1,c^2,d^2) = b^1 + b^1c^2 + a^1d^2 + b^1d^2 + r^2$	$\rightarrow x'^7$	$x'^6 + x'^7 + x'^8 = x^2$
$f^8(a^2,b^2,c^2,d^2) = b^2c^2 + a^2d^2 + b^2d^2$	$\rightarrow x'^8$	

$g^0(a^0,b^0,c^0,d^0) = a^0 + c^0 + d^0 + a^0c^0 + b^0c^0 + a^0d^0 + c^0d^0$	$\rightarrow y'^0$	
$g^1(a^1,b^1,c^0,d^0) = a^1 + b^1 + a^1c^0 + b^0c^0 + a^1d^0 + r^3$	$\rightarrow y'^1$	$y'^0 + y'^1 + y'^2 = y^0$
$g^2(a^2,b^2,c^0,d^0) = b^2 + c^0 + d^0 + a^2c^0 + b^2c^0 + a^2d^0 + c^0d^0 + r^4$	$\rightarrow y'^2$	
$g^3(a^0,b^0,c^1,d^1) = a^0 + a^0c^1 + b^0c^1 + a^0d^1 + r^3$	$\rightarrow y'^3$	
$g^4(a^1,b^1,c^1,d^1) = a^1c^1 + b^1c^1 + a^1d^1$	$\rightarrow y'^4$	$y'^3 + y'^4 + y'^5 = y^1$
$g^5(a^2,b^2,c^1,d^1) = a^2 + b^2 + a^2c^1 + b^2c^1 + a^2d^1 + r^5$	$\rightarrow y'^5$	
$g^6(a^0,b^0,c^2,d^2) = a^0c^2 + b^0c^2 + a^0d^2 + c^2d^2 + r^4$	$\rightarrow x'^6$	
$g^7(a^1,b^1,c^2,d^2) = a^1 + b^1 + a^1c^2 + b^1c^2 + a^1d^2 + r^5$	$\rightarrow y'^7$	$y'^6 + y'^7 + y'^8 = y^2$
$g^8(a^2,b^2,c^2,d^2) = a^2 + a^2c^2 + b^2c^2 + a^2d^2 + c^2d^2$	$\rightarrow y'^8$	

B 3-SHARE SQUARE-SCALE-MULTIPLIER USING 6 BITS OF FRESH MASKS

$F(a,\ b,\ c,\ d) = (x,\ y)$

$x = f(a,b,c,d) = b + ac + bc + d + ad$

$y = g(a,b,c,d) = a + b + c + bc + d + ad + bd$

The construction is first-order secure without fresh masks.

The outputs x and y are uniform with the input variables a and b without fresh masks.

$f^0(a^0,b^0,c^0,d^0) = a^0 + c^0 + a^0c^0 + b^0c^0 + a^0d^0$	$\rightarrow x'^0$	
$f^1(a^0,b^0,c^1,d^1) = b^0 + c^1 + d^1 + a^0c^1 + b^0c^1 + a^0d^1 + r^0$	$\rightarrow x'^1$	$x'^0 + x'^1 + x'^2 = x^0$
$f^2(a^0,b^0,c^2,d^2) = a^0 + d^2 + a^0c^2 + b^0c^2 + a^0d^2 + r^1$	$\rightarrow x'^2$	
$f^3(a^1,b^1,c^0,d^0) = a^1 + c^0 + a^1c^0 + b^1c^0 + a^1d^0 + r^0$	$\rightarrow x'^3$	
$f^4(a^1,b^1,c^1,d^1) = a^1c^1 + b^1c^1 + a^1d^1$	$\rightarrow x'^4$	$x'^3 + x'^4 + x'^5 = x^1$
$f^5(a^1,b^1,c^2,d^2) = a^1 + b^1 + c^2 + d^2 + a^1c^2 + b^1c^2 + a^1d^2 + r^2$	$\rightarrow x'^5$	
$f^6(a^2,b^2,c^0,d^0) = a^2 + d^0 + a^2c^0 + b^2c^0 + a^2d^0 + r^1$	$\rightarrow x'^6$	
$f^7(a^2,b^2,c^1,d^1) = c^1 + a^2c^1 + b^2c^1 + a^2d^1 + r^2$	$\rightarrow x'^7$	$x'^6 + x'^7 + x'^8 = x^2$
$f^8(a^2,b^2,c^2,d^2) = a^2 + b^2 + c^2 + d^2 + a^2c^2 + b^2c^2 + a^2d^2$	$\rightarrow x'^8$	

$g^0(a^0,b^0,c^0,d^0) = c^0 + b^0c^0 + a^0d^0 + b^0d^0$	$\rightarrow y'^0$	
$g^1(a^0,b^0,c^1,d^1) = a^0 + b^0 + d^1 + b^0c^1 + a^0d^1 + b^0d^1 + r^3$	$\rightarrow y'^1$	$y'^0 + y'^1 + y'^2 = y^0$
$g^2(a^0,b^0,c^2,d^2) = c^2 + d^2 + b^0c^2 + a^0d^2 + b^0d^2 + r^4$	$\rightarrow y'^2$	
$g^3(a^1,b^1,c^0,d^0) = d^0 + b^1c^0 + a^1d^0 + b^1d^0 + r^3$	$\rightarrow y'^3$	
$g^4(a^1,b^1,c^1,d^1) = b^1c^1 + a^1d^1 + b^1d^1$	$\rightarrow y'^4$	$y'^3 + y'^4 + y'^5 = y^1$
$g^5(a^1,b^1,c^2,d^2) = a^1 + b^1 + c^2 + d^2 + b^1c^2 + a^1d^2 + b^1d^2 + r^5$	$\rightarrow y'^5$	
$g^6(a^2,b^2,c^0,d^0) = b^2 + a^2b^2 + b^2c^0 + a^2d^0 + b^2d^0 + r^4$	$\rightarrow x'^6$	
$g^7(a^2,b^2,c^1,d^1) = c^1 + b^2c^1 + a^2d^1 + b^2d^1 + r^5$	$\rightarrow y'^7$	$y'^6 + y'^7 + y'^8 = y^2$
$g^8(a^2,b^2,c^2,d^2) = a^2 + c^2 + d^2 + a^2b^2 + b^2c^2 + a^2d^2 + b^2d^2$	$\rightarrow y'^8$	

C 3-SHARE SQUARE-SCALE-MULTIPLIER USING 6 BITS OF FRESH MASKS

$F(a,\ b,\ c,\ d) = (x,\ y)$

$x = f(a,b,c,d) = b + ac + bc + d + ad$

$y = g(a,b,c,d) = a + b + c + bc + d + ad + bd$

The construction is first-order secure without fresh masks.

The outputs x and y are uniform with the input variables c and d without fresh masks.

$f^0(a^0,b^0,c^0,d^0) = a^0 + a^0c^0 + b^0c^0 + a^0d^0 + c^0d^0$	$\rightarrow x'^0$	
$f^1(a^0,b^0,c^1,d^1) = c^0 + a^1c^0 + b^1c^0 + a^1d^0 + c^0d^0 + r^0$	$\rightarrow x'^1$	$x'^0 + x'^1 + x'^2 = x^0$
$f^2(a^0,b^0,c^2,d^2) = a^2 + b^2 + c^0 + d^0 + a^2c^0 + b^2c^0 + a^2d^0 + r^1$	$\rightarrow x'^2$	
$f^3(a^1,b^1,c^0,d^0) = b^0 + a^0c^1 + b^0c^1 + a^0d^1 + r^0$	$\rightarrow x'^3$	
$f^4(a^1,b^1,c^1,d^1) = a^1c^1 + b^1c^1 + a^1d^1$	$\rightarrow x'^4$	$x'^3 + x'^4 + x'^5 = x^1$
$f^5(a^1,b^1,c^2,d^2) = a^2 + d^1 + a^2c^1 + b^2c^1 + a^2d^1 + r^2$	$\rightarrow x'^5$	
$f^6(a^2,b^2,c^0,d^0) = a^0 + c^2 + a^0c^2 + b^0c^2 + a^0d^2 + r^1$	$\rightarrow x'^6$	
$f^7(a^2,b^2,c^1,d^1) = b^1 + a^1c^2 + b^1c^2 + a^1d^2 + r^2$	$\rightarrow x'^7$	$x'^6 + x'^7 + x'^8 = x^2$
$f^8(a^2,b^2,c^2,d^2) = c^2 + d^2 + a^2c^2 + b^2c^2 + a^2d^2$	$\rightarrow x'^8$	

$g^0(a^0,b^0,c^0,d^0) = b^0 + c^0 + d^0 + b^0c^0 + a^0d^0 + b^0d^0$	$\rightarrow y'^0$	
$g^1(a^0,b^0,c^1,d^1) = a^1 + c^0 + d^0 + b^1c^0 + a^1d^0 + b^1d^0 + r^3$	$\rightarrow y'^1$	$y'^0 + y'^1 + y'^2 = y^0$
$g^2(a^0,b^0,c^2,d^2) = c^0 + d^0 + b^2c^0 + a^2d^0 + b^2d^0 + r^4$	$\rightarrow y'^2$	
$g^3(a^1,b^1,c^0,d^0) = b^0 + c^1 + d^1 + b^0c^1 + a^0d^1 + b^0d^1 + r^3$	$\rightarrow y'^3$	
$g^4(a^1,b^1,c^1,d^1) = b^1c^1 + a^1d^1 + b^1d^1$	$\rightarrow y'^4$	$y'^3 + y'^4 + y'^5 = y^1$
$g^5(a^1,b^1,c^2,d^2) = a^2 + b^2 + b^2c^1 + a^2d^1 + b^2d^1 + r^5$	$\rightarrow y'^5$	
$g^6(a^2,b^2,c^0,d^0) = a^0 + b^0 + c^2 + d^2 + b^0c^2 + a^0d^2 + b^1d^2 + r^4$	$\rightarrow x'^6$	
$g^7(a^2,b^2,c^2,d^1) = b^1 + b^1c^2 + a^2d^2 + b^1d^2 + r^5$	$\rightarrow y'^7$	$y'^6 + y'^7 + y'^8 = y^2$
$g^8(a^2,b^2,c^2,d^2) = b^2c^2 + a^2d^2 + b^2d^2$	$\rightarrow y'^8$	