Guarding the First Order: The Rise of AES Maskings

Amund Askeland[2], Siemen Dhooghe[1], Svetla Nikova[1], Vincent Rijmen[1,2], Zhenda Zhang[1]

[1] imec-COSIC, ESAT, KU Leuven, Belgium
firstname.lastname@esat.kuleuven.be
[2] University of Bergen, Bergen, Norway
firstname.lastname@uib.no

Abstract We provide three first-order hardware maskings of the AES, each allowing for a different trade-off between the number of shares and the number of register stages. All maskings use a generalization of the changing of the guards method enabling the re-use of randomness between masked S-boxes. As a result, the maskings do not require fresh randomness while still allowing for a minimal number of shares and providing provable security in the glitch-extended probing model. The low-area variant has five cycles of latency and a serialized area cost of 8.13 kGE. The low-latency variant reduces the latency to three cycles while increasing the serialized area by 67.89% compared to the low-area variant. The maskings of the AES encryption are implemented on FPGA and evaluated with Test Vector Leakage Assessment (TVLA).

Keywords: AES, Hardware, Probing Security, Threshold Implementations

1 Introduction

The Advanced Encryption Standard (AES) [8] is one of the most used cryptographic building blocks in practice. The cipher has secured many applications, including the world wide web. However, for some applications, like embedded devices, naive implementations of the AES are vulnerable to side-channel attacks such as Differential Power Analysis (DPA) due to Kocher *et al.* [18]. The current agreed-upon method to protect implementations against DPA is masking. In masking, each key-dependent variable is split into several random shares such that an adversary needs to view the power consumption of each share to gain information on the secret variable.

Several maskings of the AES appeared in the literature in the past twenty years. The efficiency and security of them have been significantly improved over time. Threshold implementations by Nikova *et al.* [21] allowed for maskings that protect against glitches in hardware. The uniformity aspect of threshold implementations allows for the reduction of randomness. The changing of the guards

technique by Daemen [7] showed how to make a masking uniform without significantly increasing costs. Canright [4] proposed an efficient tower field decomposition of the AES S-box in order to improve hardware costs. This decomposition was then used by De Cnudde *et al.* [9] to create efficient threshold implementation maskings of the AES. However, the authors noted that the randomness cost of their designs is high, which makes it infeasible to generate the randomness in a cryptographic secure way.

Contributions. In this paper, we generalize the changing of the guards technique to include maskings that use randomness in Section 3. The method allows the re-use of randomness between all masked S-boxes and retains first-order probing security. As a result, the generalization tackles the open question by De Cnudde *et al.* as we significantly reduce the randomness cost of their masking, at least for first-order security. We then provide three variants of the masking in Section 4, which show the trade-off between the number of register stages and the number of shares.

We apply these maskings of S-boxes to both the serialized and round-based hardware architecture of the AES-128 encryption. These constructions are proven secure in the first-order glitch-extended robust probing model. The low-area variation of the serialized AES, described in Section 4.2, has an area cost of 8.13 *kGE* and 5 cycles of latency. The latency is reduced to 4 cycles for the S-box in Section 4.3 and 3 cycles for the S-box in Section 4.4 with the area cost of the serialized AES increased by 56.75% and 67.89%, respectively. The serialized AES implementations are tested on our side-channel leakage assessment setup to show the first-order security of the designs. Our implementations amortize the cost of online randomness, at the same time provide provable and physically tested first-order probing security, and achieve one of the most efficient area versus latency trade-offs in the literature.

2 Preliminaries

In this section, we go over the used notation, introduce the AES, the probing side-channel security model, and threshold implementations.

2.1 Notation

We denote bits by subscript and shares by superscript. We denote the most significant bit by a bigger subscript. For example, given $(a_1, ..., a_8)$, a_8 denotes the most significant bit (MSB) and a_1 the least significant bit (LSB). Finally, the concatenation of bits is denoted by $\|$.

2.2 Description of AES

We quickly introduce the standardized AES cipher by Daemen and Rijmen [8]. There are three levels of security 128, 192, and 256. AES consists of a 128-bit

state and 128, 192, or 256-bit key, respectively, divided into bytes. The cipher is composed of 10, 12, or 14 rounds, respectively, each applying an addition of a subkey, a bricklayer of S-Boxes, a `ShiftRows` operation, and a `MixColumns` operation. The AES S-Box consists of an inversion in the field $\mathbb{F}_{2^8}$ and the application of an affine layer. This is visually represented in Figure 1.

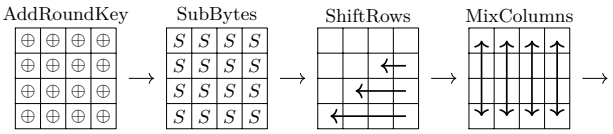


Figure 1: Representation of the AES.

The key schedule for AES-128, which operates on 4 columns of 32 bits each, is depicted in Figure 2. Each round of the AES state function has a parallel round of the key schedule. We provide a description for the AES-128 key schedule. Denote V_j, with $j \in \{1, ..., 4\}$, the jth word of the key state at round i and W_j the jth word of the key state at round $i+1$. Then a round of the key schedule is defined as

$$
\begin{aligned}
W_1 &= V_1 \oplus \texttt{RotWord}(\texttt{SubWord}(V_4)) + C_{i+1}\,, \\
W_2 &= V_2 \oplus W_1\,, \\
W_3 &= V_3 \oplus W_2\,, \\
W_4 &= V_4 \oplus W_3\,.
\end{aligned}
$$

With `RotWord`, the left circular shift, `SubWord`, the application of four AES S-boxes, and C_{i+1}, the round constants for round $i+1$.

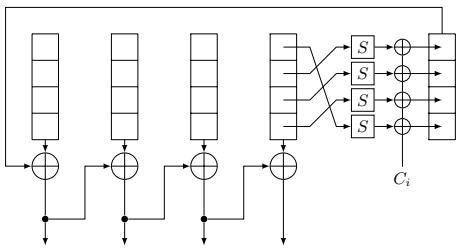


Figure 2: The AES-128 key schedule. $C_i \in \mathbb{F}_2^{32}$ denotes the ith round constants, and S denotes the AES S-box.

2.3 The Threshold Glitch-Extended Probing Model

This section introduces the threshold probing model.

Threshold probing. A dth-order threshold probing adversary $\mathcal{A}$, as first proposed by Ishai *et al.* [17], can view the values present on up to d gates or wires in a circuit implementing a cipher during a single execution (cipher evaluation). We note that by "probe" we do not mean a physical probe such as an EM probe. Instead, the word probe is used as an abstract concept through which an adversary can perfectly observe a part of the computation.

The adversary $\mathcal{A}$ is computationally unbounded, and must specify the location of the probes before querying the circuit. However, the adversary can change the location of the probes over multiple cipher queries. The adversary's interaction with the circuit is mediated through the encoder and decoder algorithms, neither of which can be probed.

The security model is a simulation model where the simulator needs to simulate the probed values from scratch, more specifically, the simulator is not given the input (including the key of a block cipher) of the circuit. The adversary needs to distinguish the probed values from the real circuit with the returned values from the simulator. A failure in doing so proves that the adversary can not learn anything from the circuit's input via the probes. In a security proof, this essentially comes down to proving that the probed values follow a distribution which is independent of the value of the circuit's input.

Glitches. The above model is extended to capture the effect of glitches on hardware. Whereas one of the adversary's probes normally results in the value of a single wire, a glitch-extended probe allows obtaining all the registered inputs leading to the gate/wire which is probed. This extension of the probing model has been discussed in the work of Reparaz *et al.* [22] and formalized by Faust *et al.* [12]. The formulation of the latter work is as follows: "For any ϵ-input circuit gadget G, combinatorial recombinations (aka glitches) can be modeled with specifically ϵ-extended probes so that probing any output of the function allows the adversary to observe all its ϵ inputs."

2.4 Boolean Masking and Threshold Implementations

Boolean masking was independently introduced by Goubin and Patarin [15] and Chari *et al.* [5]. It serves as a sound and widely-deployed countermeasure against side-channel attacks. The technique is based on splitting each secret variable $x \in \mathbb{F}_2$ in the circuit into shares $\bar{x} = (x^1, x^2, \ldots, x^{s_x})$ such that $x = \sum_{i=1}^{s_x} x^i$ over $\mathbb{F}_2$. A random Boolean masking of a fixed secret is uniform if all maskings of that secret are equally likely.

There are several approaches to protect a circuit by masking. In this work, we make use of threshold implementations, proposed by Nikova *et al.* [21]. In particular, we focus on "first-order threshold implementations" as those which protect against first-order side-channel attacks. The interested reader is referred to the works by Bilgin *et al.* [2] and Beyne *et al.* [1] for more information on how to use threshold implementations to secure against higher-order attacks. In the following, the main properties of threshold implementations as introduced by Nikova *et al.* are reviewed.

A threshold implementation consists of several layers of Boolean functions, as shown in Figure 3. As for any masked design, a black-box encoder function generates a uniform random masking of the input before it enters the masked circuit. At the end of each layer, synchronization is ensured by means of registers which stop the propagation of glitches.

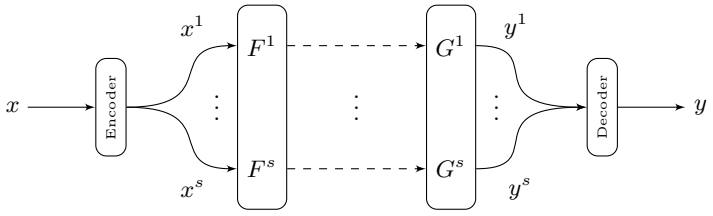


Figure 3: Schematic illustration of a threshold implementation assuming an equal number of input and output shares [11].

Let $\bar{F}$ be a layer in the threshold implementation corresponding to a part of the circuit $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$. For example, F might be the linear layer of a block cipher. The function $\bar{F} : \mathbb{F}_2^{ns_x} \to \mathbb{F}_2^{ms_y}$, where we assume s_x shares per input bit and s_y shares per output bit, will be called a *masking* of F. The ith share of the function $\bar{F}$ is denoted by $F^i : \mathbb{F}_2^{ns_x} \to \mathbb{F}_2^m$, for $i \in \{1, .., s_y\}$. Maskings can have a number of properties that are relevant in the security argument for a threshold implementation; these properties are summarized in Definition 1.

Definition 1 (Properties of threshold implementations [21]). *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 masking of F. The masking $\bar{F}$ is said to be*

correct
if $\sum_{i=1}^{s_y} F^i(x^1, \ldots, x^{s_x}) = F\left(\sum_{i=1}^{s_x} x^i\right)$ for all shares $x^1, \ldots, x^{s_x} \in \mathbb{F}_2^n$,

non-complete
if any function F^i, measured between register stages, depends on at most $s_x - 1$ input shares,

uniform
if $\bar{F}$ maps a uniform random masking of any $x \in \mathbb{F}_2^n$ to a uniform random masking of $F(x) \in \mathbb{F}_2^m$.

Considering that, in a threshold implementation, all input/outputs of the functions are stored in registers, placing a glitch-extended probe in a layer of a threshold implementation returns all inputs of the probed masked Boolean function. If all layers of a threshold implementation are non-complete and uniform, the resulting masked circuit can be proven secure in the first-order probing model with glitches [11].

3 Changing of the Guards with Randomness

The changing of the guards method proposed by Daemen [7] is a technique that transforms a non-complete masking into a uniform and non-complete masking. The technique works by embedding the masking into a Feistel-like structure. In this paper, we slightly generalize the method by considering a first-order probing secure masking. Such a masking potentially requires multiple register stages and extra randomness to guarantee its security. The adapted changing of the guards method still ensures uniformity while allowing the re-use of the randomness. An example of the method with two shares is shown in Figure 4.

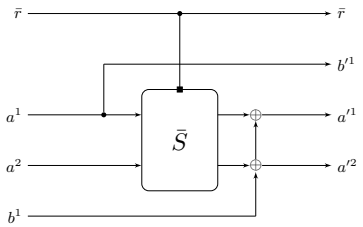


Figure 4: Changing of the guards method with two shares where the masked S-box $\bar{S}$ uses the randomness $\bar{r}$.

With the original changing of the guards method, the function $\bar{S}$ was non-complete. Meaning that $\bar{S}$ typically did not use fresh randomness and was computed in one cycle. Instead, with the generalized method, $\bar{S}$ can use fresh randomness (which can be recycled and used for different S-boxes) and be computed in multiple stages (in this work the whole tower-field decomposition of the AES S-box).

We give the changing of the guards method formally in Definition 2.

Definition 2. *The generalized changing of the guards method applied to a masked map $\bar{S}$ given inputs $(a^1, ..., a^s)$, $(b^1, ..., b^{s-1})$, and randomness $\bar{r}$ is calculated as follows*

$$\bar{r}' = \bar{r} \tag{1}$$

$$a'^1 = S^1(a^1, ..., a^s, \bar{r}) \oplus b^1, \quad \dots \quad , \quad a'^{s-1} = S^{s-1}(a^1, ..., a^s, \bar{r}) \oplus b^{s-1}, \tag{2}$$

$$a'^s = S^s(a^1, ..., a^s, \bar{r}) \oplus b^1 \oplus \dots \oplus b^{s-1} \tag{3}$$

$$b'^1 = a^1, \quad \dots \quad , \quad b'^{s-1} = a^{s-1}. \tag{4}$$

In general, we refer to the $(b^1, ..., b^{s-1})$ as the *guards* of the masked S-box $\bar{S}$. We show that the changing of the guards construction with randomness retains the correctness and probing security properties from $\bar{S}$ and makes the masking uniform.

Theorem 1. *The method from Definition 2 is correct, first-order probing secure, and uniform.*

Proof. The correctness of the construction follows from the correctness of $\bar{S}$ and the fact that each share b^i is added to two different output shares in Eq. (2)-(3).

First-order probing security of the construction, assuming a joint uniform input, follows from the first-order probing security of $\bar{S}$ and the facts that the share b^s is not used in the construction and that each share b'^i is calculated using only one share a^i using Eq. (2)-(3).

For the proof of uniformity, we first take an arbitrary input secret a. We show that the above construction is invertible. In other words, given the secret a and the outputs $(a'^1, ..., a'^s)$, $(b'^1, ..., b'^{s-1})$, $\bar{r}'$, we show it is possible to construct the inputs $(a^1, ..., a^s)$, $(b^1, ..., b^{s-1})$, $\bar{r}$.

Since the input secret a is given, we can construct the inputs $(a^1, ..., a^s)$ from $(b'^1, ..., b'^{s-1})$ using Eq. (4). From $\bar{r}'$ we can evidently construct $\bar{r}$ since the two are equal (see Eq. (1)). By running $(a^1, ..., a^s)$ and $\bar{r}$ through $\bar{S}$ and XORing the output $(a'^1, ..., a'^s)$, we can also construct $(b^1, ..., b^s)$ (see Eq. (2)-(3)) which concludes the proof. □

Thus, the changing of the guards method allows for the transformation of any first-order probing secure masking into a uniform one which allows the re-use of the randomness used in the S-box.

4 Maskings of the S-Box

This section describes the three first-order glitch-extended probing secure S-box designs of the AES. We first go over the components which are masked between all designs and then provide the three designs themselves.

4.1 Overarching Components

We quickly review the functions used in the tower-field decomposition of the S-box.

Input/output isomorphism. The first operation occurring in the decomposed S-box performs a change of basis through a linear map. This mapping is implemented in combinational logic, and it maps the 8-bit input $(a_1^i, ..., a_8^i)$ to the 8-bit output $(y_1^i, ..., y_8^i)$ for each share $i \in \{1, 2\}$ as follows:

$$
\begin{aligned}
y_8^i &= a_8^i \oplus a_7^i \oplus a_6^i \oplus a_3^i \oplus a_2^i \oplus a_1^i & \qquad y_4^i &= a_8^i \oplus a_5^i \oplus a_4^i \oplus a_2^i \oplus a_1^i \\
y_7^i &= a_7^i \oplus a_6^i \oplus a_5^i \oplus a_1^i & \qquad y_3^i &= a_1^i \\
y_6^i &= a_7^i \oplus a_6^i \oplus a_2^i \oplus a_1^i & \qquad y_2^i &= a_7^i \oplus a_6^i \oplus a_1^i \\
y_5^i &= a_8^i \oplus a_7^i \oplus a_6^i \oplus a_1^i & \qquad y_1^i &= a_7^i \oplus a_4^i \oplus a_3^i \oplus a_2^i \oplus a_1^i
\end{aligned}
$$

The inverse linear map (including the AES affine transformation) maps the 8-bit input $(u_1^i, ..., u_8^i)$ to the 8-bit output $(y_1^i, ..., y_8^i)$ for each share i as follows:

$$
\begin{aligned}
y_8^i &= u_6^i \oplus u_4^i & y_4^i &= u_8^i \oplus u_7^i \oplus u_6^i \oplus u_5^i \oplus u_4^i \\
y_7^i &= u_8^i \oplus u_4^i & y_3^i &= u_7^i \oplus u_6^i \oplus u_4^i \oplus u_3^i \oplus u_1^i \\
y_6^i &= u_7^i \oplus u_1^i & y_2^i &= u_6^i \oplus u_5^i \oplus u_2^i \\
y_5^i &= u_8^i \oplus u_6^i \oplus u_4^i & y_1^i &= u_7^i \oplus u_5^i \oplus u_2^i
\end{aligned}
$$

The constant 0x63 is then added to the first share $(y_8^1, ..., y_1^1)$.

Finite field multipliers. In the computation of the masking of the S-box, we make use of multipliers over $\mathbb{F}_{2^2}$ and $\mathbb{F}_{2^4}$. We note that these equations only hold for the multipliers in the masked S-box and not for any other operation in the AES like the `MixColumns`. We recall the equations for the multiplication over $\mathbb{F}_{2^2}$. These map the two 2-bit inputs $(a_1, a_2), (b_1, b_2)$ to the 2-bit output (c_1, c_2) as follows:

$$c_1 = (a_2 \oplus a_1)(b_2 \oplus b_1) \oplus a_1 b_1 \qquad c_2 = (a_2 \oplus a_1)(b_2 \oplus b_1) \oplus a_2 b_2$$

We then recall the equations for the multiplication over $\mathbb{F}_{2^4}$. This maps the two 4-bit inputs $(a_1, a_2, a_3, a_4), (b_1, b_2, b_3, b_4)$ to the 4-bit output (c_1, c_2, c_3, c_4) as follows:

$$
\begin{aligned}
c_1 =& a_4b_4 \oplus a_2b_4 \oplus a_3b_3 \oplus a_1b_3 \oplus a_4b_2 \oplus a_1b_2 \oplus a_3b_1 \oplus a_2b_1 \oplus a_1b_1 \\
c_2 =& a_4b_4 \oplus a_3b_4 \oplus a_2b_4 \oplus a_1b_4 \oplus a_4b_3 \oplus a_2b_3 \oplus a_4b_2 \oplus a_3b_2 \oplus a_2b_2 \oplus a_4b_1 \oplus a_1b_1 \\
c_3 =& a_3b_4 \oplus a_2b_4 \oplus a_4b_3 \oplus a_3b_3 \oplus a_1b_3 \oplus a_4b_2 \oplus a_2b_2 \oplus a_3b_1 \oplus a_1b_1 \\
c_4 =& a_4b_4 \oplus a_2b_4 \oplus a_1b_4 \oplus a_3b_3 \oplus a_2b_3 \oplus a_4b_2 \oplus a_3b_2 \oplus a_2b_2 \oplus a_1b_2 \oplus a_4b_1 \oplus a_2b_1
\end{aligned}
$$

Scaling functions. In the decomposed S-box, there are two scaling functions, a "square scale function" and a "scale" function. The linear square scaling $SqSc$ maps the 4-bit input $(x_1, ..., x_4)$ to the 4-bit output $(y_1, ..., y_4) = (x_1, x_1 \oplus x_2, x_2 \oplus x_4, x_1 \oplus x_3)$. The scaling operation $Scale$ maps a 2-bit input (x_1, x_2) to the 2-bit output $(y_1, y_2) = (x_1 \oplus x_2, x_2)$.

Two-bit inverter. The inversion Inv in $\mathbb{F}_{2^2}$ is linear and is implemented by swapping the bits. The inversion operation maps a 2-bit input (x_1, x_2) to the 2-bit output $(y_1, y_2) = (x_2, x_1)$.

4.2 Design I: Two-Share S-Box

The first design uses two shares and is divided into five cycles. The S-box uses a total of 54 random bits which can be re-used over all the S-boxes. The design of the masking is given in Figure 5.

The masked multipliers are calculated by first computing all cross products of the input shares and then re-masking them with a random masking of zero before compressing the cross products back to the input number of shares. More specifically, the two-shared multiplier is given by

$$\begin{array}{lcllcl} a^0 & \rightarrow & a^0b^0 + r^0 & & & \\ a^1 & \rightarrow & a^0b^1 + r^1 & \rightarrow & x^0 = a^0b + r^0 + r^1 & \\ b^0 & \rightarrow & a^1b^0 + r^2 & \rightarrow & x^1 = a^1b + r^2 + r^3 & , \\ b^1 & \rightarrow & a^1b^1 + r^3 & & & \end{array} \tag{5}$$

where $r^3 = r^0 + r^1 + r^2$. Note that the above multiplier is non-complete and uniform in both stages when fresh randomness r^i is used. More specifically, the first stage expands the two input shares to a four-sharing and the second stage compresses the uniform four shares back to two output shares.

This masking is the most efficient in terms of area considering the three designs in this paper. However, the decrease in area is traded for by an increase in latency.

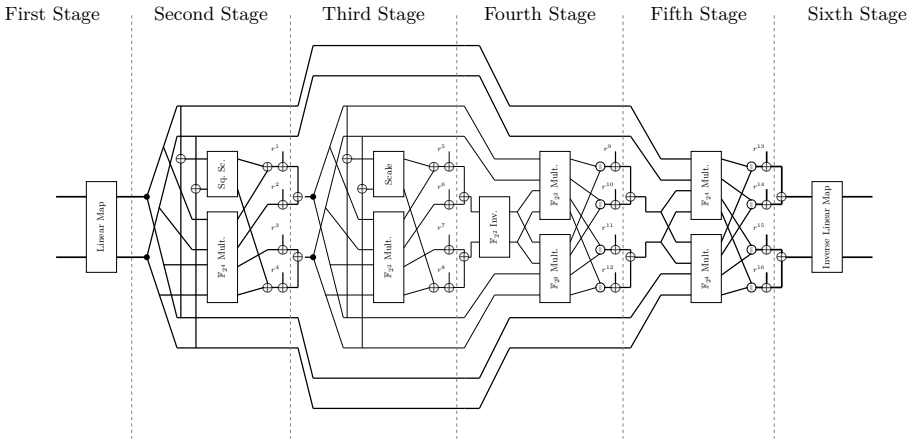


Figure 5: Representation of the S-box of design I. Register stages are denoted by dashed vertical lines.

4.3 Design II: Three-Share S-Box

The second design uses three shares and is divided into four cycles. The S-box uses a total of 36 random bits which can be re-used in every S-box. The masking is shown in Figure 6.

The masked multipliers are calculated by combining the cross-products in a non-complete way and re-masking them with a random masking of zero. More

specifically, the three-shared multiplier is given by

$$\begin{aligned} a^0, b^0 &\rightarrow a^0b^0 + a^0b^1 + a^1b^0 + r^0 \\ a^1, b^1 &\rightarrow a^1b^1 + a^1b^2 + a^2b^1 + r^1 \\ a^2, b^2 &\rightarrow a^2b^2 + a^2b^0 + a^0b^2 + r^2 \end{aligned} , \tag{6}$$

where $r^2 = r^0 + r^1$.

Compared to the previous design in Section 4.2, this S-box is larger in terms of area but has a reduced latency of four cycles compared to five of the previous design. This design works well with both a serialized and a round-based architecture.

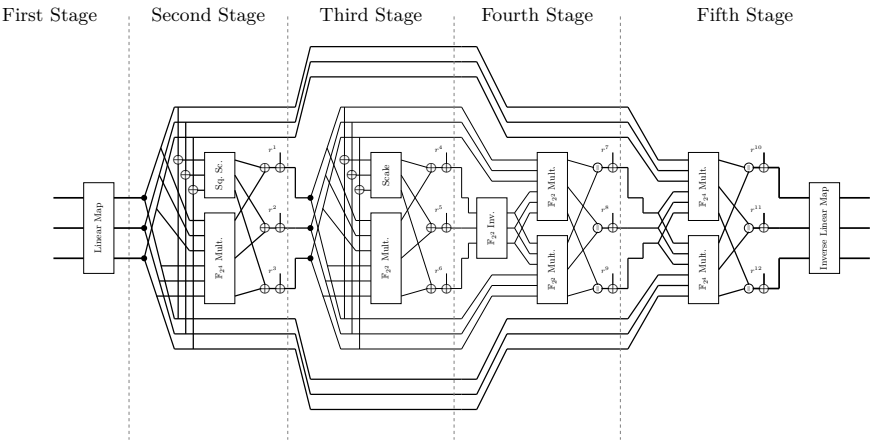


Figure 6: Representation of the S-box of design II. Register stages are denoted by dashed vertical lines.

4.4 Design III: Three-Share S-Box

The third design works over three shares and is divided into three cycles. The masking requires a total of 40 random bits which can be re-used over all S-boxes. The design is shown in Figure 7. The design uses the three-shared multipliers from design II, but the design switches to four shares in order to make a non-complete masking of the inversion over $\mathbb{F}_{2^4}$. For the inversion Inv in $\mathbb{F}_{2^4}$, the resulting equations are given by:

$$\begin{aligned} y_1 &= x_1x_3 \oplus x_1x_4 \oplus x_2x_3x_4 \oplus x_2x_4 \oplus x_3 \\ y_2 &= x_1x_3x_4 \oplus x_1x_4 \oplus x_2x_4 \oplus x_3 \oplus x_4 \\ y_3 &= x_1x_2x_4 \oplus x_1x_3 \oplus x_1 \oplus x_2x_3 \oplus x_2x_4 \\ y_4 &= x_1x_2x_3 \oplus x_1 \oplus x_2x_3 \oplus x_2x_4 \oplus x_2 \end{aligned}$$

For each cubic term $x_a x_b x_c$, the ith share for $i \in \{1,2,3,4\}$ is calculated as follows, where the convention is used that the superscripts wrap around at four.

$$\begin{aligned} x_a x_b x_c^i = {} & x_a^i x_b^i x_c^i \oplus x_a^i x_b^{i+1} x_c^i \oplus x_a^i x_b^i x_c^{i+1} \oplus x_a^i x_b^{i+1} x_c^{i+1} \oplus x_a^i x_b^{i+1} x_c^{i+2} \\ & \oplus x_a^i x_b^{i+2} x_c^{i+1} \oplus x_a^i x_b^i x_c^{i+2} \oplus x_a^i x_b^{i+2} x_c^i \oplus x_a^i x_b^{i+2} x_c^{i+2} \oplus x_a^{i+2} x_b^{i+1} x_c^i \\ & \oplus x_a^{i+1} x_b^{i+2} x_c^i \oplus x_a^{i+2} x_b^i x_c^{i+1} \oplus x_a^{i+1} x_b^i x_c^{i+2} \oplus x_a^{i+2} x_b^{i+1} x_c^{i+1} \\ & \oplus x_a^{i+2} x_b^{i+2} x_c^{i+1} \oplus x_a^{i+2} x_b^{i+1} x_c^{i+2} \end{aligned}$$

The ith share for $i \in \{1,2,3,4\}$ of a quadratic term $x_a x_b$ is calculated as follows:

$$x_a x_b^i = x_a^i x_b^i \oplus x_a^i x_b^{i+1} \oplus x_a^i x_b^{i+2} \oplus x_a^{i+2} x_b^{i+1}$$

The linear terms are added share-wise. The output of the inversion is then refreshed with a zero-masking.

Compared to the previous designs, this one reduces the latency to three cycles, but it trades off a larger area cost. This design is better suited for a round-based architecture.

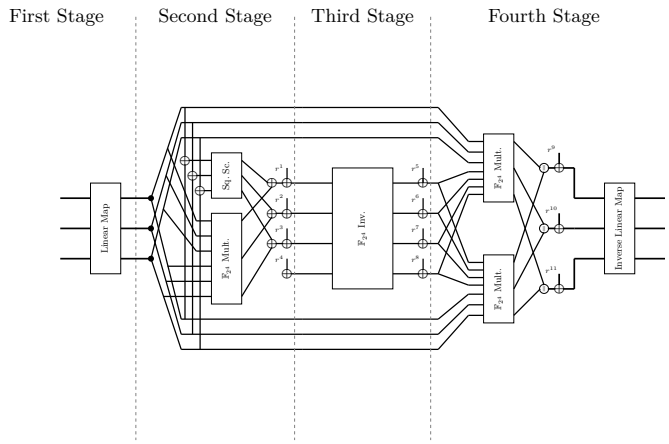


Figure 7: Representation of the S-box of design III. Register stages are denoted by dashed vertical lines.

5 Architecture

Our designs of the masked AES-128 encryption are implemented in both a serialized architecture and a round-based architecture.

In the round-based AES-128 architecture, 20 S-boxes are instantiated in which 16 S-boxes process `SubBytes` for the state function and 4 for `SubWord` in the key expansion. The bus width of `MixColumns` and `ShiftRows` is $n_shares \times 128$ bits. After each round of the AES encryption, the state of the cipher is stored in a register array. Thus, the whole encryption needs 10 rounds of $sbox_latency+1$ cycles plus the latency by the control logic. The changing of the guards method is applied to the bricklayer of S-boxes following Definition 1 by Daemen [6].

The implementations that are evaluated in Section 7.2 use the serialized architecture where there is one S-box for both the key expansion and the state. The bus width is thus $n_shares \times 8$ bits. The state registers and key registers can be viewed as 4×4 arrays, similar to the serialized encryption modules by Shahmirzadi and Moradi [24] and De Meyer *et al.* [10]. Each masked byte after `AddRoundKey`, or after `RotWord` in the key expansion, is fed into the masked S-box. Thus, the serialized architecture needs at least 20 cycles per round. If the latency of the S-box is 4 cycles, the stages of the S-box are pipelined without wasting a cycle. If the latency is larger than 4 cycles, waiting cycles are inserted at the end of each round. If the latency is less than 4 cycles, the round-based architecture is preferred for lower latency applications.

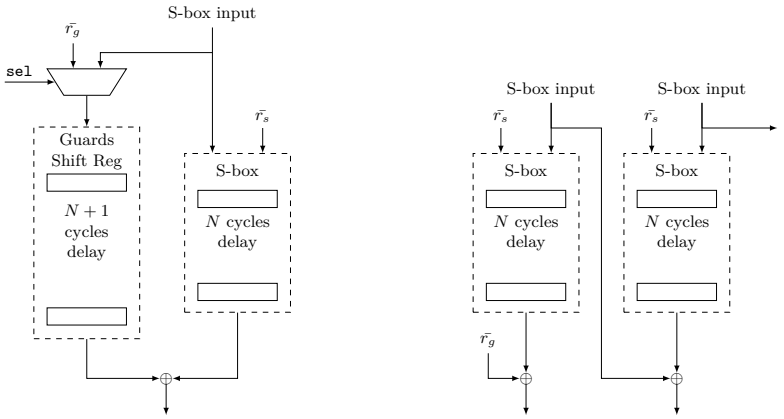
The guard shares are initialized with randomness and the subsequent guards are taken from the input of the S-box. Shift registers are used to store these shares in such a way that they are delayed by one more cycle than the S-box delay before they are applied to the output. This is depicted in Figure 8. A total of $8 \times (n_shares - 1)$ random bits are required to initialize the guard shares. These can be static throughout the AES encryption. The random bits in the masked S-boxes are rotated per applied S-box. This prevents transitional leakages from happening in the pipeline registers. Taking the two-shared multiplication of Eq. (5) as an example, by rotating (r^0, r^1, r^2, r^3), two consecutive executions (first with a, b and then with c, d) calculating x^0 gives Hamming distance leakage $HD(a^0b + r^0 + r^1, c^0d + r^1 + r^2)$ which is masked by r^0, r^2.

Before the AES starts for the serialized implementations, the masked key and plaintext are loaded to the register array in 16 cycles. The ciphertext is read out from the state register array in 16 cycles when the computation is finished. Table 1 depicts the latency in the number of cycles from when the load signal arrives to when the ciphertext is ready at the output.

6 First-Order Probing Security

In this section, we argue the first-order probing security of the three designs. We show this by arguing that the designs are threshold implementations, see Definition 1. We refer to Dhooghe *et al.* [11] for the proof that a threshold implementation is first-order robust probing secure.

The masked S-boxes are first-order probing secure due to the extra randomness added to each register stage. This refreshing works as follows. Given an input $(a^1, ..., a^s)$, an arbitrary masked map $\bar{F}$, and randomness $r^1, ..., r^{s-1}$, refreshing



(a) Application of the guards in the serialized architecture.

(b) Application of the guards in the round-based architecture.

Figure 8: Application of changing of the guards in our implementations.

is done as follows, for $i \in \{1, ..., s-1\}$

$$a'^i = F^i(a^1, ..., a^s) \oplus r^i, \qquad a'^s = F^s(a^1, ..., a^s) \oplus r^1 \oplus ... \oplus r^{s-1}, \tag{7}$$

where each F^i is non-complete.

Lemma 1. *The refreshing following Eq (7) gives a uniform output.*

Proof. We show that the function, taking $(a^1, ..., a^s)$ and $(r^1, ..., r^{s-1})$ as input and $(a^1, ..., a^s)$, $F^i(a^1, ..., a^s) \oplus r^i$ for $i \in \{1, ..., s-1\}$, and $F^s(a^1, ..., a^s) \oplus r^1 \oplus ... \oplus r^{s-1}$ as output, is invertible. Removing the $(a^1, ..., a^s)$ then gives a balanced (or uniform) output of the refreshing detailed in Eq (7).

The derivation is straightforward. Since $(a^1, ..., a^s)$ is given in the output, one can calculate $F^i(a^1, ..., a^s)$ for $i \in \{1, ..., s\}$. Subtracting this from the output $(a^1, ..., a^{s-1})$ then gives $(r^1, ..., r^{s-1})$ showing the map is invertible. □

Theorem 2. *Designs I, II, and III from Sections 4.2,4.3, and 4.4 are threshold implementations as given by Definition 1.*

Proof. First, each design uses a changing of the guards method. We refer to Theorem 1 for the proof that the masked input and output of each masked S-box is uniform.

Since firstly, the linear layers of the construction are evidently non-complete; secondly, they work share-wise and uniform; and thirdly, the unmasked linear functions are permutations, these comply with the properties from Definition 1.

We then show that the masked S-box from designs I, II, and III are first-order probing secure. Since a probe in the designs can only view one masked S-box, it suffices to show that each stage in the masked S-box complies with the threshold implementation properties.

Each stage in the masked S-box either maps a part of the input to the output (such as in the outer wires of Figure 5) or the output is masked using the randomness $\bar{r}$. Due to the changing of the guards structure this randomness $\bar{r}$ is joint uniform with the input of the masked S-box. From Lemma 1, we find that each stage of the masked S-box of design I, II, or III is uniform.

Finally, each stage in the masked S-box is also non-complete. As a result, the masked S-boxes from designs I, II, and III are itself threshold implementations and thus first-order robust probing secure. □

7 Implementations and Physical Evaluations

In this section, we explain the hardware implementations of the three AES designs and their side-channel analysis security.

7.1 Implementations and Comparison to Related Work

We implement our three AES designs for a Xilinx Kintex-7 FPGA mounted on a Sakura-X [19] evaluation board. The implementations are synthesized and programmed using Xilinx ISE. The `KEEP_HIERACHY` option is enabled in the Xilinx ISE to prevent optimization across modules in the synthesis step.

The area of the masked ciphers is measured in gate equivalences (GE), i.e., the cipher area normalized to the area of a 2-input `NAND` gate in a given standard cell library. The cell library we use is the `NANGATE` 45nm Open Cell Library [20], and the synthesis results are obtained with the Synopsis Design Compiler v2021.06. Comparing the area cost in gate equivalences can reduce the impact of different cell libraries. However, the delay on the critical path, or the maximum frequency, depends on the timing metrics of the used cell library. The latency is measured in the number of cycles to get from the input to the output (*e.g.* from an S-box or a cipher).

Table 1 and Table 2 show the results of our implementation. Table 1 contains the serialized AES implementations and their comparison to other implementations in the literature. Design I gives a 2-share masked AES with the changing of the guards techniques and opts for a low area cost and a moderate latency. Compared to the 4-share AES by Wegener and Moradi [26] and the 3-share AES by Sugawara [25] which also uses the changing of the guards technique, this implementation costs 91.37% fewer cycles and 52.46% less area, respectively. Design II reduces 10 cycles of latency in the AES encryption with the trade-off on a 56.75% larger area and a 37.11% lower maximum frequency. Shahmirzadi and Moradi [24] present a randomness-free 2-share AES implementation with a 5.2% smaller area and a design using one bit per S-box with a 12.2% smaller area. Both designs require a lower maximum frequency compared to design I, although the UMC 180 standard cell library was used. However, we wish to emphasize that both designs are based on using a non-uniform masked S-box whereas our S-box is uniform allowing for a stronger security argument. Moreover, in the work of Shahmirzadi and Moradi, it is mentioned that “the application of changing of

Table 1: Implementation cost of the serialized AES.

Design	Area (S-box) [kGE]	Area (AES) [kGE]	Latency (S-box) [cc]	Latency (AES) [cc]	Random[1] [bpc]	f_{max} [MHz][2]
Design I: Sec. 4.2	2.71	8.13	6	242	0	820
Design II: Sec. 4.3	4.33	12.75	5	232	0	515
Design III: Sec. 4.4	5.83	13.66	4	232	0	534
Bilgin *et al.* [3][4]	2.84	8.12	4	246	32	-
De Cnudde *et al.* [9]	1.98	6.68	6	276	54	-
Wegener-Moradi [26]	4.20	7.60	16	2 804	0	-
Sugawara [25]	3.50	17.10	4	266	0	-
Shahmirzadi-Moradi [24][3]	-	7.14	6	246	1	160
Shahmirzadi-Moradi [24][3]	-	7.71	6	246	0	160

1. Cost of online fresh random bits per cycle (*bpc*)
2. Depending on different standard cell libraries
3. With a non-uniform masked S-box
4. With an additional register stage in the S-box to store the state

the guards on 2-share implementations to nullify the required fresh randomness does not seem trivial (or even possible)". As a result, our work indicates that this is possible. Table 2 covers the round-based implementations. The low-latency S-box reduces the latency of the whole encryption down to 42 cycles using design III. Although the same S-boxes are used in the round-based AES compared to the serialized implementations in Table 1, each estimated f_{max} is 3% to 20% lower than its serialized counterpart. The reason of this reduction is the wider bus in the round-based architecture making the fan-out in control logic cost an extra delay on the critical path. Design III costs 26.52% less area compared to the work by Sasdrich *et al.* [23]. Note that Sasdrich *et al.* used LUT-based Masked Dual-Rail with Pre-charge Logic (LMDPL) and included the mask-table generation circuit in the implementation.

7.2 Evaluation

In this section, we describe the practical evaluation of the three masked designs. We collect the amplified power traces from the measurement point on the Sakura-X board. The power measurements are amplified by a 30 dB SMA pre-amplifier [13]. The traces are captured by an oscilloscope at a sample rate of 500 MS/s while the FPGA is clocked at 6.144 MHz.

The non-specific leakage detection test from Goodwill *et al.* [14] is performed which verifies that our implementations do not show first-order leakage. The measured power traces are partitioned into two sets, where the first set $\mathcal{S}_0$ receives fixed plaintexts and the second set $\mathcal{S}_1$ contains random plaintexts. The

Table 2: Implementation cost of the round-based AES.

Design	Area (S-box) $[kGE]$	Area (AES) $[kGE]$	Latency (S-box) $[cc]$	Latency (AES) $[cc]$	Random[1] $[bpc]$	f_{max} $[MHz]$[2]
Design I: Sec. 4.2	2.71	63.91	6	62	0	800
Design II: Sec. 4.3	4.33	102.44	5	52	0	505
Design III: Sec. 4.4	5.84	115.73	4	42	0	426
Gross *et al.* [16]	60.76	-	1	-	2 048	356
Gross *et al.* [16]	6.74	-	2	-	416	584
Sasdrich *et al.* [23]	3.48	157.50	1	10	720	400

1. Cost of online fresh random bits per cycle (*bpc*)
2. Depending on different standard cell libraries

two sets of measurements are compared using the t-test statistic:

$$t = \frac{\mu(\mathcal{S}_0) - \mu(\mathcal{S}_1)}{\sqrt{\frac{\sigma^2(\mathcal{S}_0)}{|\mathcal{S}_0|} + \frac{\sigma^2(\mathcal{S}_1)}{|\mathcal{S}_1|}}} \tag{8}$$

The t-test verifies whether the two sets have the same first-order moment. Its null hypothesis H_0 states that "the sets $\mathcal{S}_0$ and $\mathcal{S}_1$ are drawn from populations with the same mean." Large absolute values of this t-statistic indicate that the null hypothesis can be rejected with a high degree of confidence. The threshold value of the t-test commonly used by the side-channel research community is 4.5. If the t-test value of the measured power trace grows over 4.5, the implementation under test is considered as insecure.

Figure 9, 11, and 12 illustrate the TVLA result of the three serialized masked AES implementations. Figure 10 shows the maximum univariate t-test value changing over time which is denoted by the number of traces. The first-order probing security is verified physically for the implementations. The second-order leakages shows as anticipated. The exception is the third order leakage for design I which does not leak due to the noise amplification for third-order moments.

8 Conclusion

We proposed three first-order secure maskings of the AES with a different number of shares and register stages. These maskings allow the area versus latency trade-off in the hardware design of AES. The maskings use the generalized changing of the guards technique, which allows for the re-use of their randomness between the S-boxes. As a result, all designs proposed in this paper do not require fresh randomness in their calculation.

The three variations of the masked AES S-boxes show the area versus latency trade-off. Design I needs 5 cycles to compute the S-box and achieves a low-area serialized AES which costs only 8.13 kGE. Design II balances the area and

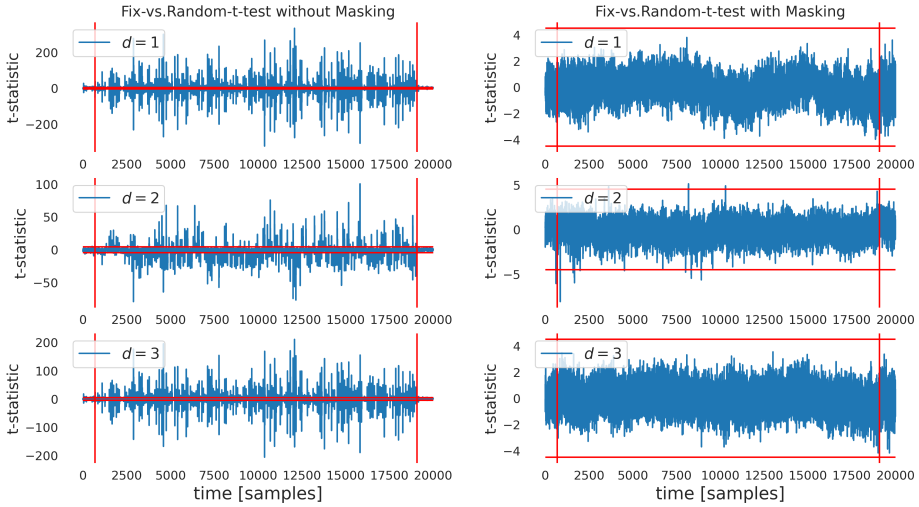
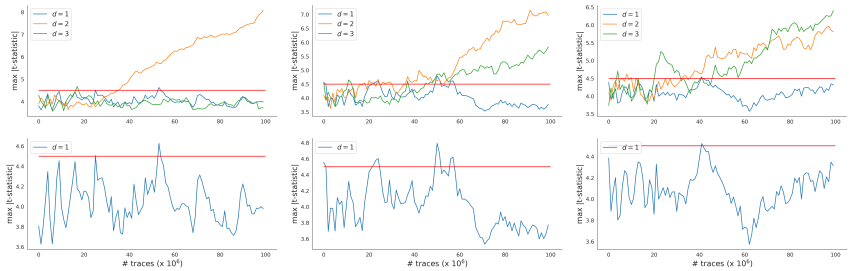


Figure 9: Univariate fixed-vs.-random t-test results for the serialized AES-128 encryption for design I. The right three plots show the results for the first to the third order statistical moments of the masked implementations at 10^8 traces. The results in the left column are for the unmasked implementations at 10^4 traces. The regions of interest are between vertical red lines, which indicate the start and end of the AES encryption.



(a) Design I. (b) Design II. (c) Design III.

Figure 10: The Y-axes of each subfigure is the maximum value of the univariate t-test results for the serialized AES-128 designs in this paper. The X-axes are the numbers of traces. The first two figures show the maximum t-test values of the first to the third statistical moments. The last one shows the first-order maximum t-test values only.

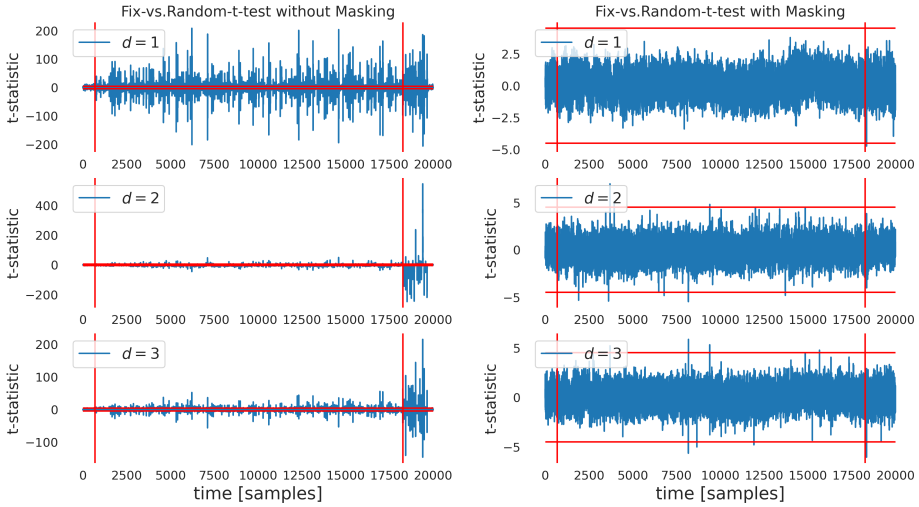


Figure 11: Univariate fixed-vs.-random t-test results for the serialized AES-128 encryption for design II. The layout is the same as in Figure 9.

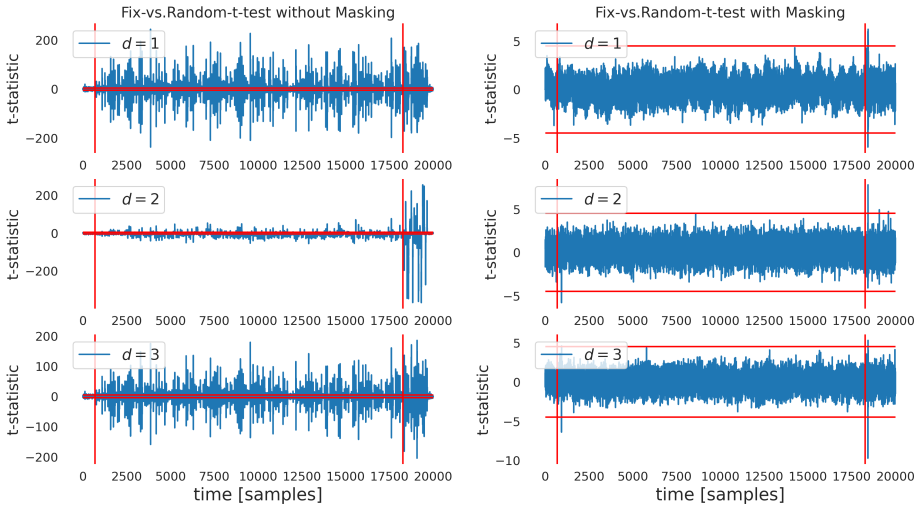


Figure 12: Univariate fixed-vs.-random t-test results for the serialized AES-128 encryption for design III. The layout is the same as in Figure 9.

latency cost. It reduces the S-box latency to 4 cycles and increases the area cost to 12.75 kGE. Design III is made for low latency applications, which requires 3 cycles for the S-box and costs 13.66 kGE for the serialized AES encryption. The designs do not require online randomness and are proven to be probing secure. The maskings are also implemented on FPGA and tested in practice. The TVLA results verify that the designs are first-order secure.

Acknowledgments. This work was supported by CyberSecurity Research Flanders with reference number VR20192203. Siemen Dhooghe is supported by a PhD Fellowship from the Research Foundation – Flanders (FWO). Zhenda Zhang is funded by a research grant from KU Leuven.

References

1. Beyne, T., Dhooghe, S., Zhang, Z.: Cryptanalysis of masked ciphers: A not so random idea. In: ASIACRYPT 2020, Part I. pp. 817–850. LNCS, Springer, Heidelberg (Dec 2020). https://doi.org/10.1007/978-3-030-64837-4_27
2. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Higher-order threshold implementations. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 326–343. Springer, Heidelberg (Dec 2014). https://doi.org/10.1007/978-3-662-45608-8_18
3. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Trade-offs for threshold implementations illustrated on AES. IEEE Trans. on CAD of ICs and Systems **34**(7), 1188–1200 (2015). https://doi.org/10.1109/TCAD.2015.2419623, https://doi.org/10.1109/TCAD.2015.2419623
4. Canright, D.: A very compact S-box for AES. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 441–455. Springer, Heidelberg (Aug / Sep 2005). https://doi.org/10.1007/11545262_32
5. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_26
6. Daemen, J.: Changing of the guards: a simple and efficient method for achieving uniformity in threshold sharing. Cryptology ePrint Archive, Report 2016/1061 (2016), https://eprint.iacr.org/2016/1061
7. Daemen, J.: Changing of the guards: A simple and efficient method for achieving uniformity in threshold sharing. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 137–153. Springer, Heidelberg (Sep 2017). https://doi.org/10.1007/978-3-319-66787-4_7
8. Daemen, J., Rijmen, V.: Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce (Nov 2001)
9. De Cnudde, T., Reparaz, O., Bilgin, B., Nikova, S., Nikov, V., Rijmen, V.: Masking AES with d+1 shares in hardware. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 194–212. Springer, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53140-2_10
10. De Meyer, L., Reparaz, O., Bilgin, B.: Multiplicative masking for AES in hardware. IACR TCHES **2018**(3), 431–468 (2018). https://doi.org/10.13154/tches.v2018.i3.431-468, https://tches.iacr.org/index.php/TCHES/article/view/7282

11. Dhooghe, S., Nikova, S., Rijmen, V.: Threshold implementations in the robust probing model. In: Bilgin, B., Petkova-Nikova, S., Rijmen, V. (eds.) Proceedings of ACM Workshop on Theory of Implementation Security, TIS@CCS 2019, London, UK, November 11, 2019. pp. 30–37. ACM (2019). https://doi.org/10.1145/3338467.3358949, https://doi.org/10.1145/3338467.3358949
12. Faust, S., Grosso, V., Pozo, S.M.D., Paglialonga, C., Standaert, F.X.: Composable masking schemes in the presence of physical defaults & the robust probing model. IACR TCHES **2018**(3), 89–120 (2018). https://doi.org/10.13154/tches.v2018.i3.89-120, https://tches.iacr.org/index.php/TCHES/article/view/7270
13. GmbH, L.E.T.: Langer emv - pa 303 sma, preamplifier 100 khz up to 3 ghz, https://www.langer-emv.de/en/product/preamplifier/37/pa-303-sma-set-preamplifier-100-khz-up-to-3-ghz/520/pa-303-sma-preamplifier-100-khz-up-to-3-ghz/42
14. Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A testing methodology for side-channel resistance validation (September 2011)
15. Goubin, L., Patarin, J.: DES and differential power analysis (the "duplication" method). In: Koç, Çetin Kaya., Paar, C. (eds.) CHES'99. LNCS, vol. 1717, pp. 158–172. Springer, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48059-5_15
16. Gross, H., Iusupov, R., Bloem, R.: Generic low-latency masking in hardware. IACR TCHES **2018**(2), 1–21 (2018). https://doi.org/10.13154/tches.v2018.i2.1-21, https://tches.iacr.org/index.php/TCHES/article/view/871
17. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (Aug 2003). https://doi.org/10.1007/978-3-540-45146-4_27
18. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_25
19. Lab./UEC, S.: Sakura (sasebo-giii) (2014), https://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-X.html
20. NANGATE: The NanGate 45nm Open Cell Library, version: PDKv1.3_v2010_12.Apache.CCL. Available at https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts/tree/master/flow/platforms/nangate45
21. Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: Ning, P., Qing, S., Li, N. (eds.) ICICS 06. LNCS, vol. 4307, pp. 529–545. Springer, Heidelberg (Dec 2006)
22. Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating masking schemes. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 764–783. Springer, Heidelberg (Aug 2015). https://doi.org/10.1007/978-3-662-47989-6_37
23. Sasdrich, P., Bilgin, B., Hutter, M., , Marson, M.E.: Low-latency hardware masking with application to AES. IACR TCHES **2020**(2), 300–326 (2020). https://doi.org/10.13154/tches.v2020.i2.300-326, https://tches.iacr.org/index.php/TCHES/article/view/8553
24. Shahmirzadi, A.R., Moradi, A.: Re-consolidating first-order masking schemes. IACR TCHES **2021**(1), 305–342 (2021). https://doi.org/10.46586/tches.v2021.i1.305-342, https://tches.iacr.org/index.php/TCHES/article/view/8736

25. Sugawara, T.: 3-share threshold implementation of AES s-box without fresh randomness. IACR TCHES **2019**(1), 123–145 (2018). https://doi.org/10.13154/tches.v2019.i1.123-145, https://tches.iacr.org/index.php/TCHES/article/view/7336
26. Wegener, F., Moradi, A.: A first-order SCA resistant AES without fresh randomness. In: Fan, J., Gierlichs, B. (eds.) COSADE 2018. LNCS, vol. 10815, pp. 245–262. Springer, Heidelberg (Apr 2018). https://doi.org/10.1007/978-3-319-89641-0_14