First-Order Hardware Sharings of the AES

Siemen Dhooghe, Svetla Nikova, Vincent Rijmen

imec-COSIC, ESAT, KU Leuven, Belgium
firstname.lastname@esat.kuleuven.be

Abstract We provide three first-order sharings of the AES each allowing for a different trade-off between the number of shares and the number of register stages. All sharings use a generalization of the changing of the guards method by allowing randomness to be used in the shared S-box. As a result, the sharings have minimal randomness requirements. The sharings are written out in detail to ease implementation efforts.

Keywords: AES, DPA, Hardware, Probing Security, Threshold Implementations

1 Introduction

The Advanced Encryption Standard (AES) [13] is one of the most used cryptographic building blocks these days. The cipher has secured many applications including the world wide web. However, for some applications, like in embedded devices, naive implementations of the AES are vulnerable to side-channel attacks such as Differential Power Analysis (DPA) due to Kocher *et al.* [12]. The current agreed-upon method to protect implementations against DPA is sharing. In sharing, 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.

The past twenty years several sharings of the AES appeared in the literature. Several significant steps were made to improve the efficiency and security of the sharing. Threshold implementations by Nikova *et al.* [14] allowed for sharings which protect against glitches in hardware. The uniformity aspect of threshold implementations allows for the reduction of randomness in the sharing. The changing of the guards technique by Daemen [5] showed how to easily make a sharing uniform without significantly increasing costs. Canright [3] 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.* [7] to create efficient threshold implementation sharings of the AES. However, the authors noted that the randomness cost of their designs are high making it infeasible to generate the randomness in a cryptographic secure way.

Contributions. We provide a generalization of the changing of the guards to include sharings which use randomness. The technique allows the re-use of randomness between all masked S-boxes and still retain 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 sharing at least for first-order security. We then provide three variants of the sharing, each trading off the number of register stages with the number of shares. The sharings are written in such a way that it is easy to implement them reading the document. Software implementations of the masked S-boxes can be found on GitLab.

Currently no implementation costs or practical leakage tests are included. However, we welcome input from the community.

Outline. The paper starts introducing AES, the probing model, sharing, and threshold implementations in Section 2. We then introduce a generalization of the changing of the guards technique in Section 3. Since the three introduced sharings have some similarities, we introduce those in Section 4. Section 5 provides the AES sharing with minimal area requirements. Section 6 provides another sharing with lower latency. Section 7 provides a further trade-off between area and latency.

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 $(a_1, ..., a_8)$ then a_8 denotes the most significant bit and a_1 the least significant.

2.2 Description of AES

We quickly introduce the AES cipher designed by Daemen and Rijmen [6] and standardized by NIST [13]. 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.

The key schedule for AES-128, which operates on 4 columns of 32 bits each, is depicted in Figure 2. For each round of the AES state function, there is 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 j^{th} word of the key state at round i and W_j the j^{th} word of the key state at round $i + 1$. Then a round of

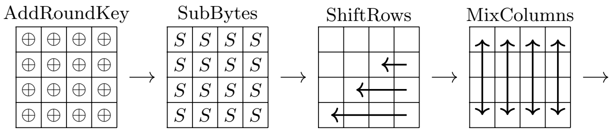

Figure 1: Representation of the AES.

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 $i+1$.

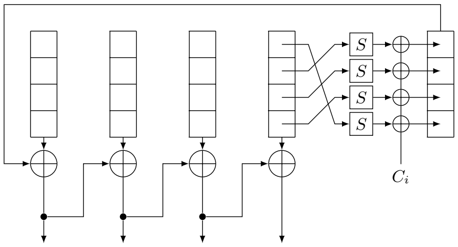

Figure 2: The AES-128 key schedule. The i^{th} round constants are denoted by $C_i \in \mathbb{F}_2^{32}$ and S denotes the AES S-box.

2.3 The Threshold Glitch-Extended Probing Model

This section introduces the threshold probing model.

Threshold Probing. A d^{th}-order threshold probing adversary $\mathcal{A}$, as first proposed by Ishai *et al.* [11], 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 encoder and decoder algorithms, neither of which can be probed.

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.* [15] and formalized by Faust *et al.* [9]. 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 Sharing and Threshold Implementations

Boolean sharing was independently introduced by Goubin and Patarin [10] and Chari *et al.* [4]. 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 sharing of a fixed secret is uniform if all sharings of that secret are equally likely.

There are several approaches to sharing a circuit. In this work, we make use of threshold implementations, proposed by Nikova *et al.* [14]. 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 sharing of the input before it enters the shared circuit and the output shares are recombined by a decoder function. At the end of each layer, synchronization is ensured by means of registers which are assumed to stop glitches.

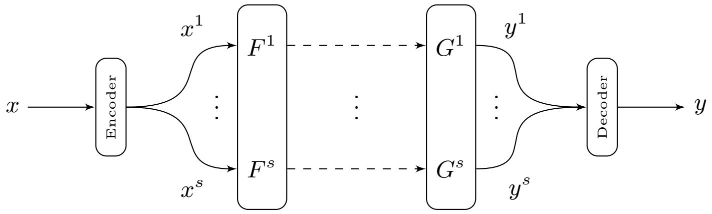

Figure 3: Schematic illustration of a threshold implementation assuming an equal number of input and output shares.

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 *sharing* of F. The i^{th} share of the function $\bar{F}$ is denoted by $F^i : \mathbb{F}_2^{ns_x} \to \mathbb{F}_2^m$, for $i \in \{1, .., s_y\}$. Sharings 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 [14]). *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 $\sum_{i=1}^{s_y} F^i(x^1, \ldots, x^{s_x}) = F(x)$ for all $x \in \mathbb{F}_2^n$ and for all shares $x^1, \ldots, x^{s_x} \in \mathbb{F}_2^n$ such that $\sum_{i=1}^{s_x} x^i = x$,*
2. non-complete *if any function F^i, measured between register stages, 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 $F(x) \in \mathbb{F}_2^m$.*

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 shared Boolean function. If all layers of a threshold implementation are non-complete and uniform, the resulting shared circuit can be proven secure in the first-order probing model with glitches [8].

3 Changing of the Guards With Randomness

The changing of the guards method proposed by Daemen [5] 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. In this paper, we slightly generalize the method by considering a first-order probing secure sharing. Such a sharing potentially requires multiple register stages and extra randomness to guarantee its security. The adapted changing of the guards method still ensures uniformity and first-order probing security while allowing the re-use of the randomness. An example of the method with two shares is shown in Figure 4.

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

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

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

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

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

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

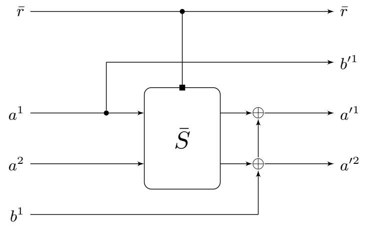

Figure 4: Changing of the guards method with two shares where the shared S-box $\bar{S}$ uses the randomness $\bar{r}$.

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

We show that the changing of the guards construction with randomness retains the correctness and probing security properties from $\bar{S}$, but makes the sharing uniform.

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

Proof. 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 Equations 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 Equations 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 Equation 4. From $\bar{r}'$ we can evidently construct $\bar{r}$ since the two are equal (see Equation 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 Equations 2-3) which concludes the proof. □

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

4 Overarching Structure of the AES Sharings

In this section we provide the common parts of all first-order designs. Only the sharing of the S-box differs for each design. These are detailed in Sections 5, 6, and 7.

4.1 Multipliers

In the computation of the sharing 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 shared 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:

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

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:

$$\begin{aligned}
c_1 =&(a_4 \otimes b_4) \oplus (a_2 \otimes b_4) \oplus (a_3 \otimes b_3) \oplus (a_1 \otimes b_3) \oplus (a_4 \otimes b_2) \oplus (a_1 \otimes b_2) \\
&\oplus (a_3 \otimes b_1) \oplus (a_2 \otimes b_1) \oplus (a_1 \otimes b_1) \\
c_2 =&(a_4 \otimes b_4) \oplus (a_3 \otimes b_4) \oplus (a_2 \otimes b_4) \oplus (a_1 \otimes b_4) \oplus (a_4 \otimes b_3) \oplus (a_2 \otimes b_3) \\
&\oplus (a_4 \otimes b_2) \oplus (a_3 \otimes b_2) \oplus (a_2 \otimes b_2) \oplus (a_4 \otimes b_1) \oplus (a_1 \otimes b_1) \\
c_3 =&(a_3 \otimes b_4) \oplus (a_2 \otimes b_4) \oplus (a_4 \otimes b_3) \oplus (a_3 \otimes b_3) \oplus (a_1 \otimes b_3) \oplus (a_4 \otimes b_2) \\
&\oplus (a_2 \otimes b_2) \oplus (a_3 \otimes b_1) \oplus (a_1 \otimes b_1) \\
c_4 =&(a_4 \otimes b_4) \oplus (a_2 \otimes b_4) \oplus (a_1 \otimes b_4) \oplus (a_3 \otimes b_3) \oplus (a_2 \otimes b_3) \oplus (a_4 \otimes b_2) \\
&\oplus (a_3 \otimes b_2) \oplus (a_2 \otimes b_2) \oplus (a_1 \otimes b_2) \oplus (a_4 \otimes b_1) \oplus (a_2 \otimes b_1)
\end{aligned}$$

4.2 Sharing the State and Key

For the sharing of the AES state and key, we use classical Boolean sharing. The key and state are each extended by $8 \cdot (s-1)$ random bits, with s the number of shares, for the changing of the guards technique as introduced in Section 3.

For two-share designs, each byte of the state or key x is shared using a random byte r as follows:

$$x^1 = x \oplus r \qquad\qquad x^2 = r$$

For three-share designs, each byte of the state or key x is shared using random bytes (r_1, r_2) as follows:

$$x^1 = x \oplus r_1 \oplus r_2 \qquad\qquad x^2 = r_1 \qquad\qquad x^3 = r_2$$

4.3 Sharing the Affine Transformations

The sharing of the linear transformations (like `MixColumns`, `ShiftRows`, or the affine layer in the S-box) is simply done share-wise. Constants are added to the first share of the relevant variable.

4.4 Changing of the Guards Implemented

In this section we explain how to implement the changing of the guards method. We instantiate an extra random sharing of zero at the start of execution. The output of each shared S-box is remasked using the current guards. The guards are then replaced by the input of that shared S-box.

For two-share designs, given a guard g, a shared S-box with inputs (a^1, a^2) and the changing of the guards is calculated as follows:

$$a'^1 = S^1(a^1, a^2) \oplus g \qquad a'^2 = S^2(a^1, a^2) \oplus g,$$

the guard is then replaced as follows

$$g' = a^1.$$

For three-share designs, given guards g^1, g^2, a shared S-box with inputs (a^1, a^2, a^3) and the changing of the guards is calculated as follows:

$$\begin{aligned} a'^1 &= S^1(a^1, a^2, a^3) \oplus g^1 \qquad & a'^2 &= S^2(a^1, a^2, a^3) \oplus g^2 \\ a'^3 &= S^3(a^1, a^2, a^3) \oplus g^1 \oplus g^2, & & \end{aligned}$$

the guards are then replaced as follows

$$g'^1 = a^1 \qquad g'^2 = a^2.$$

The designer is free to choose which guards are used to refresh an S-box. For example, the designer can choose to calculate the S-boxes row by row or column by column, it does not affect the first-order probing security of the design.

4.5 Key Schedule

The AES S-boxes in the key schedule are shared in the same way as the one from the state function, this is detailed in Sections 5, 6, and 7. The shared linear layers work share-wise. Finally, a changing of the guards structure is applied over the S-boxes in `SubWord`, this changing of the guards method follows the explanation above. The guards used in the state function can also be used in the key schedule. Meaning that the changing of the guards technique can be used across both the state function as the key schedule.

In case the shared key schedule is run to refresh the shared secret key, one first refreshes the shared master key after which, the shared key schedule is run to produce the refreshed round keys.

Thus, for two-share designs the key (k^1, k^2) is refreshed using 128 fresh random bits r, as follows:

$$k'^1 = k^1 \oplus r \qquad k'^2 = k^2 \oplus r$$

For three-share designs the key (k^1, k^2, k^3) is refreshed using 256 fresh random bits (r_1, r_2), as follows:

$$k'^1 = k^1 \oplus r_1 \oplus r_2 \qquad k'^2 = k^2 \oplus r_1 \qquad k'^3 = k^3 \oplus r_2$$

5 Design I: First-Order Two-Share AES

This section describes the first S-box design of a two-share first-order AES. Compared to the three-share AES in Sections 6 and 7, this sharing is lower in in the number of logic gates and higher in the number of register stages.

5.1 S-Box Sharing

The method is shown in Figure 5 and is divided into six stages and uses 54 random bits in total. These random bits can be re-used over all shared S-boxes in both the AES state function as the key schedule.

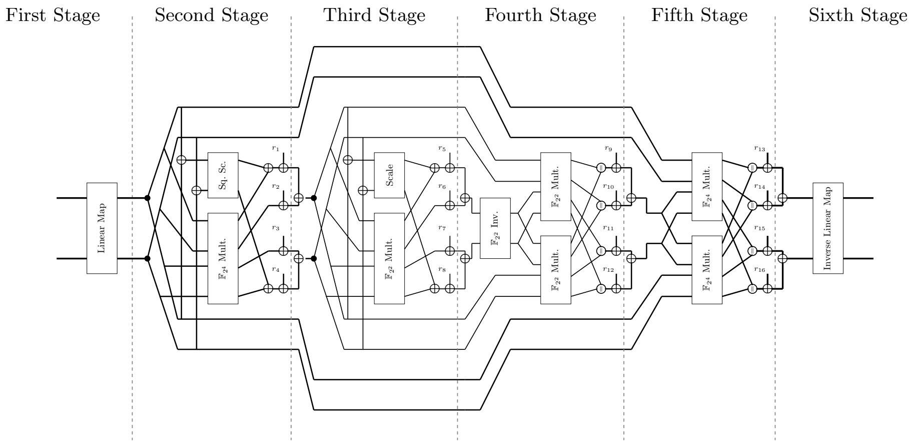

Figure 5: Representation of the S-box of design I. Register stages are denoted by dashed vertical lines.

First Stage. The first operation occurring in the decomposed S-box performs a change of basis through a linear map. Its sharing requires instantiating this linear map once for each share. 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:

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

Second Stage. We consider the parallel application of nonlinear multiplication and affine Square Scaling (Sq. Sc.) as one single function $d = b \otimes c \oplus SqSc(b \oplus c)$. The affine square scaling $SqSc$ maps the 4-bit input $(x_1, ..., x_4)$ to the 4-bit output $(y_1, ..., y_4)$ as follows:

$$y_1 = x_1 \qquad\qquad y_3 = x_2 \oplus x_4$$
$$y_2 = x_1 \oplus x_2 \qquad\qquad y_4 = x_1 \oplus x_3$$

For the parallel multiplier and square scaling with random nibbles $(r_1, ..., r_4)$, the resulting equations are given by:

$$d^1 = b^1 \otimes c^1 \oplus SqSc(b^1 \oplus c^1) \oplus r_1 \qquad d^3 = b^2 \otimes c^1 \oplus r_3$$
$$d^2 = b^1 \otimes c^2 \oplus r_2 \qquad d^4 = b^2 \otimes c^2 \oplus SqSc(b^2 \oplus c^2) \oplus r_4$$

where $r_4 = r_1 \oplus r_2 \oplus r_3$.

Third Stage. This stage is similar to the second stage. First, the four output shares $(d^1, ..., d^4)$ from the previous stage are compressed to reduce the number of output shares back to two. More specifically, the shares $(d^1, ..., d^4)$ are mapped to (e^1, e^2) as follows:

$$e^1 = d^1 \oplus d^2 \qquad\qquad e^2 = d^3 \oplus d^4$$

These nibbles are then split in 2-bit couples for further operation. The Scaling operation (Sc) replaces the similar affine Square Scaling and is executed alongside the multiplication in $\mathbb{F}_{2^2}$ as one function $h = f \otimes g \oplus Sc(f \oplus g)$. The scaling operation maps a 2-bit input (x_1, x_2) to the 2-bit output (y_1, y_2) as follows:

$$y_1 = x_1 \oplus x_2 \qquad\qquad y_2 = x_2$$

For the parallel multiplier and scaling with random two-bits $(r_5, ..., r_8)$, the resulting equations are given by:

$$h^1 = f^1 \otimes g^1 \oplus Sc(f^1 \oplus g^1) \oplus r_5 \qquad h^3 = f^2 \otimes g^1 \oplus r_7$$
$$h^2 = f^1 \otimes g^2 \oplus r_6 \qquad h^4 = f^2 \otimes g^2 \oplus Sc(f^2 \oplus g^2) \oplus r_8$$

where $r_8 = r_5 \oplus r_6 \oplus r_7$.

Fourth Stage. First, the four output shares $(h^1, ..., h^4)$ from the previous stage are compressed to reduce the number of output shares back to two. More specifically, the shares $(h^1, ..., h^4)$ are mapped to (k^1, k^2) as follows:

$$k^1 = h^1 \oplus h^2 \qquad\qquad k^2 = h^3 \oplus h^4$$

The rest of the fourth stage is composed of an inversion and two parallel multiplications in $\mathbb{F}_{2^2}$. 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) as follows:

$$y_1 = x_2 \qquad\qquad y_2 = x_1$$

The outputs of the multiplications are concatenated, denoted by $\circledparallel$ in Figure 5, to form 4-bit values in $\mathbb{F}_{2^4}$. For the inversion and the parallel multipliers $l = f \otimes Inv(k)$ and $m = Inv(k) \otimes g$ the resulting equations are given by:

$$\begin{aligned} l^1 &= f^1 \otimes Inv(k^1) & l^3 &= f^2 \otimes Inv(k^1) \\ l^2 &= f^1 \otimes Inv(k^2) & l^4 &= f^2 \otimes Inv(k^2) \\ m^1 &= Inv(k^1) \otimes g^1 & m^3 &= Inv(k^2) \otimes g^1 \\ m^2 &= Inv(k^1) \otimes g^2 & m^4 &= Inv(k^2) \otimes g^2 \end{aligned}$$

The two outputs of the parallel multipliers are then concatenated l being the most significant bits and m the least significant. The concatenated bits are refreshed with the random nibbles $(r_9, ..., r_{12})$, as follows:

$$\begin{aligned} n^1 &= l^1 \circledparallel m^1 \oplus r_9 & n^3 &= l^3 \circledparallel m^3 \oplus r_{11} \\ n^2 &= l^2 \circledparallel m^2 \oplus r_{10} & n^4 &= l^4 \circledparallel m^4 \oplus r_{12} \end{aligned}$$

where $r_{12} = r_9 \oplus r_{10} \oplus r_{11}$.

Fifth Stage. Stage 5 is similar to stage 4. The difference of the two stages lies in the absence of the inversion operation and the multiplications being performed in $\mathbb{F}_{2^4}$ instead of $\mathbb{F}_{2^2}$. First, the four output shares $(n^1, ..., n^4)$ from the previous stage are compressed to reduce the number of output shares back to two. More specifically, the shares $(n^1, ..., n^4)$ are mapped to (o^1, o^2) as follows:

$$o^1 = n^1 \oplus n^2 \qquad\qquad o^2 = n^3 \oplus n^4$$

For the parallel multipliers $p = b \otimes o$ and $q = o \otimes c$ the resulting equations are given by:

$$\begin{aligned} p^1 &= b^1 \otimes o^1 & p^3 &= b^2 \otimes o^1 \\ p^2 &= b^1 \otimes o^2 & p^4 &= b^2 \otimes o^2 \\ q^1 &= o^1 \otimes c^1 & q^3 &= o^2 \otimes c^1 \\ q^2 &= o^1 \otimes c^2 & q^4 &= o^2 \otimes c^2 \end{aligned}$$

The two outputs of the parallel multipliers are then concatenated p being the most significant bits and q the least significant. The concatenated bits are refreshed with the random nibbles $(r_{13}, ..., r_{16})$, as follows:

$$\begin{aligned} s^1 &= p^1 \circledparallel q^1 \oplus r_{13} & s^3 &= p^3 \circledparallel q^3 \oplus r_{15} \\ s^2 &= p^2 \circledparallel q^2 \oplus r_{14} & s^4 &= p^4 \circledparallel q^4 \oplus r_{16} \end{aligned}$$

where $r_{16} = r_{13} \oplus r_{14} \oplus r_{15}$.

The value from the changing of the guards method, described in Section 4.4, can already by added at this stage. Denoting this values by t^1, then this addition is performed as follows:

$$s'^1 = s^1 \oplus t^1 \qquad s'^3 = s^3$$
$$s'^2 = s^2 \qquad s'^4 = s^4 \oplus t^1$$

Sixth Stage. First, the four output shares $(s^1, ..., s^4)$ from the previous stage are compressed to reduce the number of output shares back to two. More specifically, the shares $(s^1, ..., s^4)$ are mapped to (t^1, t^2) as follows:

$$t^1 = s^1 \oplus s^2 \qquad t^2 = s^3 \oplus s^4$$

Then, the inverse linear map (including the AES affine transformation) is performed. This linear function maps the 8-bit input $(t^i_1, ..., t^i_8)$ to the 8-bit output $(y^i_1, ..., y^i_8)$ for each share i as follows:

$$\begin{aligned}
y^i_1 &= t^i_6 \oplus t^i_4 & y^i_5 &= t^i_8 \oplus t^i_7 \oplus t^i_6 \oplus t^i_5 \oplus t^i_4 \\
y^i_2 &= t^i_8 \oplus t^i_4 & y^i_6 &= t^i_7 \oplus t^i_6 \oplus t^i_4 \oplus t^i_3 \oplus t^i_1 \\
y^i_3 &= t^i_7 \oplus t^i_1 & y^i_7 &= t^i_6 \oplus t^i_5 \oplus t^i_2 \\
y^i_4 &= t^i_8 \oplus t^i_6 \oplus t^i_4 & y^i_8 &= t^i_7 \oplus t^i_5 \oplus t^i_2
\end{aligned}$$

The constant 0x63 is then added to the first share.

6 Design II: First-Order Three-Share AES

This section describes the second S-box design of a three-share first-order AES. Compared to the two-share AES in Section 5, this sharing is higher in the number of logic gates and lower in the number of register stages.

6.1 S-Box Sharing

The method is shown in Figure 6 and is divided into five stages and uses 36 random bits in total. These random bits can be re-used over all shared S-boxes in both the AES state function as the key schedule.

First Stage. The first operation occurring in the decomposed S-box performs a change of basis through a linear map. Its sharing requires instantiating this linear map once for each share. This mapping is implemented in combinational logic and it maps the 8-bit input $(a^i_1, ..., a^i_8)$ to the 8-bit output $(y^i_1, ..., y^i_8)$ for each share $i \in \{1, 2, 3\}$ as follows:

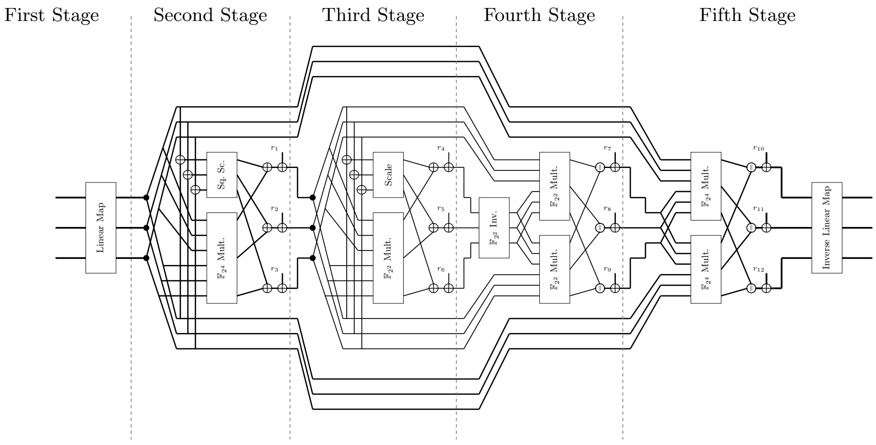

Figure 6: Representation of the S-box of design II. Register stages are denoted by dashed vertical lines.

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

Second Stage. We consider the parallel application of nonlinear multiplication and affine Square Scaling (Sq. Sc.) as one single function $d = b \otimes c \oplus SqSc(b \oplus c)$. The affine square scaling $SqSc$ maps the 4-bit input $(x_1, ..., x_4)$ to the 4-bit output $(y_1, ..., y_4)$ as follows:

$$
\begin{aligned}
y_1 &= x_1 & \qquad y_3 &= x_2 \oplus x_4 \\
y_2 &= x_1 \oplus x_2 & y_4 &= x_1 \oplus x_3
\end{aligned}
$$

For the parallel multiplier and square scaling with random nibbles (r_1, r_2, r_3), the resulting equations are given by:

$$
\begin{aligned}
d^1 &= b^1 \otimes c^1 \oplus b^1 \otimes c^2 \oplus b^2 \otimes c^1 \oplus SqSc(b^1 \oplus c^1) \oplus r_1 \\
d^2 &= b^2 \otimes c^2 \oplus b^2 \otimes c^3 \oplus b^3 \otimes c^2 \oplus SqSc(b^2 \oplus c^2) \oplus r_2 \\
d^3 &= b^3 \otimes c^3 \oplus b^3 \otimes c^1 \oplus b^1 \otimes c^3 \oplus SqSc(b^3 \oplus c^3) \oplus r_3
\end{aligned}
$$

where $r_3 = r_1 \oplus r_2$.

Third Stage. This stage is similar to the second stage. The nibbles are split in 2-bit couples for further operation. The Scaling operation (Sc) replaces the similar affine Square Scaling and is executed alongside the multiplication in $\mathbb{F}_{2^2}$ as one function $h = f \otimes g \oplus Sc(f \oplus g)$. The scaling operation maps a 2-bit input (x_1, x_2) to the 2-bit output (y_1, y_2) as follows:

$$y_1 = x_1 \oplus x_2 \qquad\qquad y_2 = x_2$$

For the parallel multiplier and scaling with random two-bits (r_4, r_5, r_6), the resulting equations are given by:

$$\begin{aligned}
h^1 &= f^1 \otimes g^1 \oplus f^1 \otimes g^2 \oplus f^2 \otimes g^1 \oplus Sc(f^1 \oplus g^1) \oplus r_4\\
h^2 &= f^2 \otimes g^2 \oplus f^2 \otimes g^3 \oplus f^3 \otimes g^2 \oplus Sc(f^2 \oplus g^2) \oplus r_5\\
h^3 &= f^3 \otimes g^3 \oplus f^3 \otimes g^1 \oplus f^1 \otimes g^3 \oplus Sc(f^3 \oplus g^3) \oplus r_6
\end{aligned}$$

where $r_6 = r_4 \oplus r_5$.

Fourth Stage. The fourth stage is composed of an inversion and two parallel multiplications in $\mathbb{F}_{2^2}$. The inversion Inv in $\mathbb{F}_{2^2}$ is linear and is implemented by swapping the bits using wires. The inversion operation maps a 2-bit input (x_1, x_2) to the 2-bit output (y_1, y_2) as follows:

$$y_1 = x_2 \qquad\qquad y_2 = x_1$$

The outputs of the multiplications are concatenated, denoted by $\circledparallel$ in Figure 6, to form 4-bit values in $\mathbb{F}_{2^4}$. For the inversion and the parallel multipliers $l = f \otimes Inv(h)$ and $m = Inv(h) \otimes g$ the resulting equations are given by:

$$\begin{aligned}
l^1 &= f^1 \otimes Inv(h^1) \oplus f^1 \otimes Inv(h^2) \oplus f^2 \otimes Inv(h^1)\\
l^2 &= f^2 \otimes Inv(h^2) \oplus f^2 \otimes Inv(h^3) \oplus f^3 \otimes Inv(h^2)\\
l^3 &= f^3 \otimes Inv(h^3) \oplus f^3 \otimes Inv(h^1) \oplus f^1 \otimes Inv(h^3)\\
m^1 &= Inv(h^1) \otimes g^1 \oplus Inv(h^1) \otimes g^2 \oplus Inv(h^2) \otimes g^1\\
m^2 &= Inv(h^2) \otimes g^2 \oplus Inv(h^2) \otimes g^3 \oplus Inv(h^3) \otimes g^2\\
m^3 &= Inv(h^3) \otimes g^3 \oplus Inv(h^3) \otimes g^1 \oplus Inv(h^1) \otimes g^3
\end{aligned}$$

The two outputs of the parallel multipliers are then concatenated l being the most significant bits and m the least significant. The concatenated bits are refreshed with the random nibbles (r_7, r_8, r_9), as follows:

$$\begin{aligned}
n^1 &= l^1 \circledparallel m^1 \oplus r_7\\
n^2 &= l^2 \circledparallel m^2 \oplus r_8\\
n^3 &= l^3 \circledparallel m^3 \oplus r_9
\end{aligned}$$

where $r_9 = r_7 \oplus r_8$.

Fifth Stage. Stage 5 is similar to stage 4. The difference of the two stages lies in the absence of the inversion operation and the multiplications being performed in $\mathbb{F}_{2^4}$ instead of $\mathbb{F}_{2^2}$. For the parallel multipliers $p = b \otimes n$ and $q = n \otimes c$ the resulting equations are given by:

$$\begin{aligned}
p^1 &= b^1 \otimes n^1 \oplus b^1 \otimes n^2 \oplus b^2 \otimes n^1 \\
p^2 &= b^2 \otimes n^2 \oplus b^2 \otimes n^3 \oplus b^3 \otimes n^2 \\
p^3 &= b^3 \otimes n^3 \oplus b^3 \otimes n^1 \oplus b^1 \otimes n^3 \\
q^1 &= n^1 \otimes c^1 \oplus n^1 \otimes c^2 \oplus n^2 \otimes c^1 \\
q^2 &= n^2 \otimes c^2 \oplus n^2 \otimes c^3 \oplus n^3 \otimes c^2 \\
q^3 &= n^3 \otimes c^3 \oplus n^3 \otimes c^1 \oplus n^1 \otimes c^3
\end{aligned}$$

The two outputs of the parallel multipliers are then concatenated p being the most significant bits and q the least significant. The concatenated bits are refreshed with the random nibbles (r_{10}, r_{11}, r_{12}), as follows:

$$\begin{aligned}
s^1 &= p^1 \textcircled{$\|$} q^1 \oplus r_{10} \\
s^2 &= p^2 \textcircled{$\|$} q^2 \oplus r_{11} \\
s^3 &= p^3 \textcircled{$\|$} q^3 \oplus r_{12}
\end{aligned}$$

where $r_{10} = r_{11} \oplus r_{12}$.

Finally, the inverse linear map (including the AES affine transformation) is performed. This linear function maps the 8-bit input $(s_1^i, ..., s_8^i)$ to the 8-bit output $(y_1^i, ..., y_8^i)$ for each share i as follows:

$$\begin{aligned}
y_1^i &= s_6^i \oplus s_4^i & y_5^i &= s_8^i \oplus s_7^i \oplus s_6^i \oplus s_5^i \oplus s_4^i \\
y_2^i &= s_8^i \oplus s_4^i & y_6^i &= s_7^i \oplus s_6^i \oplus s_4^i \oplus s_3^i \oplus s_1^i \\
y_3^i &= s_7^i \oplus s_1^i & y_7^i &= s_6^i \oplus s_5^i \oplus s_2^i \\
y_4^i &= s_8^i \oplus s_6^i \oplus s_4^i & y_8^i &= s_7^i \oplus s_5^i \oplus s_2^i
\end{aligned}$$

The constant 0x63 is then added to the first share.

7 Design III: First-Order Three-Share AES

This section describes the third S-box design of a three-share first-order AES. Compared to the three-share AES in Section 6, this sharing is even higher in the number of logic gates and lower in the number of register stages.

7.1 S-Box Sharing

The method is shown in Figure 7 and is divided into four stages and uses 40 random bits in total. These random bits can be re-used over all shared S-boxes in both the AES state function as the key schedule.

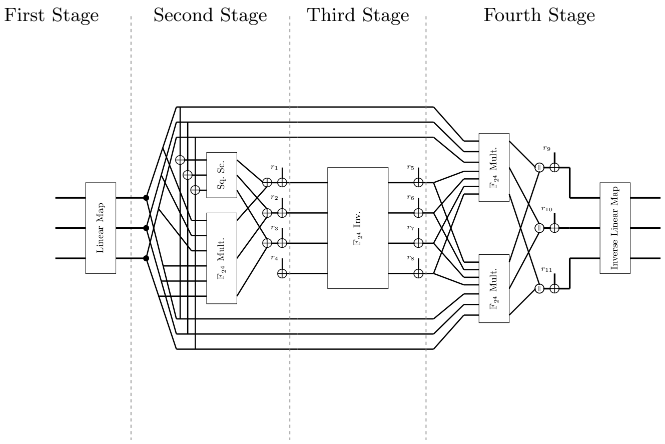

Figure 7: Representation of the S-box of design III. Register stages are denoted by dashed vertical lines.

First Stage. The first operation occurring in the decomposed S-box performs a change of basis through a linear map. Its sharing requires instantiating this linear map once for each share. 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 $i \in \{1, 2, 3\}$ as follows:

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

Second Stage. We consider the parallel application of nonlinear multiplication and affine Square Scaling (Sq. Sc.) as one single function $d = b \otimes c \oplus SqSc(b \oplus c)$. The affine square scaling $SqSc$ maps the 4-bit input $(x_1, ..., x_4)$ to the 4-bit output $(y_1, ..., y_4)$ as follows:

$$\begin{aligned} y_1 &= x_1 & y_3 &= x_2 \oplus x_4 \\ y_2 &= x_1 \oplus x_2 & y_4 &= x_1 \oplus x_3 \end{aligned}$$

For the parallel multiplier and square scaling with random nibbles (r_1, r_2, r_3, r_4), the resulting equations are given by:

$$\begin{aligned} d^1 &= b^1 \otimes c^1 \oplus b^1 \otimes c^2 \oplus b^2 \otimes c^1 \oplus SqSc(b^1 \oplus c^1) \oplus r_1 \\ d^2 &= b^2 \otimes c^2 \oplus b^2 \otimes c^3 \oplus b^3 \otimes c^2 \oplus SqSc(b^2 \oplus c^2) \oplus r_2 \\ d^3 &= b^3 \otimes c^3 \oplus b^3 \otimes c^1 \oplus b^1 \otimes c^3 \oplus SqSc(b^3 \oplus c^3) \oplus r_3 \\ d^4 &= r_4 \end{aligned}$$

where $r_4 = r_1 \oplus r_2 \oplus r_3$. Extra randomness is used to temporarily increase the number of shares in order to improve latency.

Third Stage. The third stage is composed of an inversion in $\mathbb{F}_{2^4}$. For the inversion Inv in $\mathbb{F}_{2^4}$, the resulting equations are given by:

$$\begin{aligned}
y_1 &= x_1 \otimes x_3 \oplus x_1 \otimes x_4 \oplus x_2 \otimes x_3 \otimes x_4 \oplus x_2 \otimes x_4 \oplus x_3 \\
y_2 &= x_1 \otimes x_3 \otimes x_4 \oplus x_1 \otimes x_4 \oplus x_2 \otimes x_4 \oplus x_3 \oplus x_4 \\
y_3 &= x_1 \otimes x_2 \otimes x_4 \oplus x_1 \otimes x_3 \oplus x_1 \oplus x_2 \otimes x_3 \oplus x_2 \otimes x_4 \\
y_4 &= x_1 \otimes x_2 \otimes x_3 \oplus x_1 \oplus x_2 \otimes x_3 \oplus x_2 \otimes x_4 \oplus x_2
\end{aligned}$$

As the equations for the sharing are too large, we simply explain how to share a cubic and quadratic term. Consider the cubic term xyz, the i^{th} share is calculated as follows, where the convention is used that the superscripts wrap around at four.

$$\begin{aligned}
xyz^i = {} & x^i y^i z^i \oplus x^i y^{i+1} z^i \oplus x^i y^i z^{i+1} \oplus x^i y^{i+1} z^{i+1} \oplus x^i y^{i+1} z^{i+2} \oplus x^i y^{i+2} z^{i+1} \\
& \oplus x^i y^i z^{i+2} \oplus x^i y^{i+2} z^i \oplus x^i y^{i+2} z^{i+2} \oplus x^{i+2} y^{i+1} z^i \oplus x^{i+1} y^{i+2} z^i \\
& \oplus x^{i+2} y^i z^{i+1} \oplus x^{i+1} y^i z^{i+2} \oplus x^{i+2} y^{i+1} z^{i+1} \oplus x^{i+2} y^{i+2} z^{i+1} \\
& \oplus x^{i+2} y^{i+1} z^{i+2}
\end{aligned}$$

The i^{th} share of a quadratic term xy is calculated as follows:

$$xy^i = x^i y^i \oplus x^i y^{i+1} \oplus x^{i+1} y^i \oplus x^i y^{i+2} \oplus x^{i+2} y^i$$

The linear terms are added share-wise.

The output of the inversion $(e^1, ..., e^4)$ is then refreshed with the random nibbles (r_5, r_6, r_7, r_8), as follows:

$$\begin{aligned}
f^1 &= e^1 \oplus r_5 \\
f^2 &= e^2 \oplus r_6 \\
f^3 &= e^3 \oplus r_7 \\
f^4 &= e^4 \oplus r_8
\end{aligned}$$

where $r_8 = r_5 \oplus r_6 \oplus r_7$.

Fourth Stage. In the fourth stage we have two parallel multipliers over $\mathbb{F}_{2^4}$. For the parallel multipliers $g = b \otimes f$ and $h = f \otimes c$ the resulting equations are given by:

$$\begin{aligned}
g^1 &= b^1 \otimes f^1 \oplus b^1 \otimes f^2 \oplus b^1 \otimes f^3 \oplus b^2 \otimes f^1 \\
g^2 &= b^2 \otimes f^2 \oplus b^2 \otimes f^3 \oplus b^2 \otimes f^4 \oplus b^3 \otimes f^2 \\
g^3 &= b^3 \otimes f^3 \oplus b^3 \otimes f^4 \oplus b^3 \otimes f^1 \oplus b^1 \otimes f^4 \\
h^1 &= f^1 \otimes c^1 \oplus f^1 \otimes c^2 \oplus f^1 \otimes c^3 \oplus f^2 \otimes c^1 \\
h^2 &= f^2 \otimes c^2 \oplus f^2 \otimes c^3 \oplus f^2 \otimes c^4 \oplus f^3 \otimes c^2 \\
h^3 &= f^3 \otimes c^3 \oplus f^3 \otimes c^4 \oplus f^3 \otimes c^1 \oplus f^1 \otimes c^4
\end{aligned}$$

The two outputs of the parallel multipliers are then concatenated g being the most significant bits and h the least significant. The concatenated bits are refreshed with the random nibbles (r_9, r_{10}, r_{11}), as follows:

$$\begin{aligned}
k^1 &= g^1 \textcircled{\scriptsize$\|$} h^1 \oplus r_9 \\
k^2 &= g^2 \textcircled{\scriptsize$\|$} h^2 \oplus r_{10} \\
k^3 &= g^3 \textcircled{\scriptsize$\|$} h^3 \oplus r_{11}
\end{aligned}$$

where $r_{11} = r_{10} \oplus r_9$.

Finally, the inverse linear map (including the AES affine transformation) is performed. This linear function maps the 8-bit input $(k_1^i, ..., k_8^i)$ to the 8-bit output $(y_1^i, ..., y_8^i)$ for each share i as follows:

$$\begin{aligned}
y_1^i &= k_6^i \oplus k_4^i & \qquad y_5^i &= k_8^i \oplus k_7^i \oplus k_6^i \oplus k_5^i \oplus k_4^i \\
y_2^i &= k_8^i \oplus k_4^i & \qquad y_6^i &= k_7^i \oplus k_6^i \oplus k_4^i \oplus k_3^i \oplus k_1^i \\
y_3^i &= k_7^i \oplus k_1^i & \qquad y_7^i &= k_6^i \oplus k_5^i \oplus k_2^i \\
y_4^i &= k_8^i \oplus k_6^i \oplus k_4^i & \qquad y_8^i &= k_7^i \oplus k_5^i \oplus k_2^i
\end{aligned}$$

The constant 0x63 is then added to the first share.

8 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.* [8] for the proof that a threshold implementation is first-order robust probing secure.

The shared 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 shared map $\bar{F}$, and randomness $r^1, ..., r^{s-1}$, refreshing is done as follows, for $i \in \{1, ..., s-1\}$

$$a'^i = F^i(a^1, ..., a^s) \oplus r^i, \qquad a'^s = F^s(a^1, ..., a^s) \oplus r^1 \oplus ... \oplus r^{s-1}\,. \tag{5}$$

Lemma 1. *The refreshing following Equation (5) 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 Equation 5.

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 5,6, and 7 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 shared input and output of each shared S-box is uniform.

Since the linear layers of the construction are evidently non-complete, since they work share-wise, and uniform, since the unshared linear functions are permutations, these comply to the properties from Definition 1.

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

Each stage in the shared 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 shared S-box. From Lemma 1, we find that each stage of the shared S-box of design I, II, or III is uniform.

The non-completeness is verified on sight from the equations given in Sections 5,6, 7. More specifically, it can be seen that each multiplication or inversion in the designs is shared in a non-complete way.

As a result, the shared S-boxes from designs I, II, and III are itself threshold implementations and thus first-order robust probing secure. □

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. 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
4. 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

5. 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
6. Daemen, J., Rijmen, V.: The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition. Information Security and Cryptography, Springer (2020)
7. 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
8. 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 Workshop, TIS@CCS 2019, London, UK, November 11, 2019. pp. 30–37. ACM (2019). https://doi.org/10.1145/3338467.3358949
9. 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
10. 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
11. 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
12. 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
13. National Institute of Standards and Technology: Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce (Nov 2001)
14. 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)
15. 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