3-Share Threshold Implementation of AES S-box without Fresh Randomness

Takeshi Sugawara

The University of Electro-Communications, Tokyo

Abstract. Threshold implementation is studied as a countermeasure against side-channel attack. There had been no threshold implementation for the AES and Keccak S-boxes that satisfies an important property called uniformity. In the conventional implementations, intermediate values are remasked to compensate for the lack of uniformity. The remasking consumes thousands of fresh random bits and its implementation cost is a serious concern. Daemen recently proposed a 3-share uniform threshold implementation of the Keccak S-box. This is enabled by a new technique called the *changing of the guards* which can be applied to any invertible functions. Subsequently, Wegener et al. proposed a 4-share threshold implementation of the AES S-box based on the *changing of the guards* technique. However, a 3-share threshold implementation of AES S-box remains open. The difficulty stays in 2-input multiplication, used in decomposed S-box representations, which is non-invertible because of different input and output sizes. In this study, this problem is addressed by introducing a certain generalization of the *changing of the guards* technique. The proposed method provides a generic way to construct a uniform sharing for a target function having different input and output sizes. The key idea is to transform a target function into an invertible one by adding additional inputs and outputs. Based on the proposed technique, the first 3-share threshold implementation of AES S-box without fresh randomness is presented. Performance evaluation and simulation-based leakage assessment of the implementation are also presented.

Keywords: Threshold Implementation, AES, Side-channel Attack, Changing of the Guards, Differential Power Analysis

1 Introduction

Cryptography can be used in a hostile environment in which an attacker has physical access to a computational device. In such an environment, the attacker can obtain information leakage via physical side-channels such as execution time, power consumption, and electromagnetic radiation. Side-channel attack (SCA) introduced by Kocher et al. [KJJ99] exploits such information leakage to break cryptography. Subsequently, new attacks and countermeasures have been studied for more than two decades.

SCA is a serious threat in the real world, and thus, countermeasures are indispensable in applications such as smartcards. The need for countermeasures against SCA is increasing because embedded devices are increasingly used in hostile environments for the Internet of things [RSWO18].

Correlation between side-channel leakage and secret data being processed should be removed to counteract SCA. Accordingly, countermeasures based on multi-party computation (MPC) are intensively studied as a promising approach [ISW03]. In the countermeasures, a target variable is split into a set of variables called a share in such a way that a proper subset will not leak information of the original variable. Subsequently, cryptographic computation is performed by using the shares without reconstructing the original value.

IACR Transactions on Cryptographic Hardware and Embedded Systems ISSN 2569-2925, Vol. 2019, No. 1, pp. 123–145
DOI:10.13154/tches.v2019.i1.123-145

Table 1: Performance evaluation of the proposed AES implementations and comparison with the performance of conventional designs with 1st order security. The entries with † are based on [WM18].

Design	Latency [cycles]	S-box [kGE]	AES [kGE]	Rand. [bits/S-box]	AL-product [cycles ·kGE]	Standard-cell library
[MPL+11]	266	4.2	11.1	44	2952.6	UMC 180-nm
[BGN+15]†	246	2.2	7.2	16	1771.2	UMC 180-nm
[GMK17]†	246	2.6	6.0	18	1476.0	UMC 90-nm
[CRB+16]†	276	1.9	6.3	54	1738.8	NanGate 45-nm
[UHA17]†	219	1.4	6.3	64	1379.7	TSMC 65-nm
[WM18]†	2,804	4.2	7.6	0	21310.0	UMC 180-nm
This work	266	3.5	17.1	0	4548.6	NanGate 45-nm

Nikova et al. proposed an MPC-based countermeasure called *threshold implementation* (TI) [NRR06]. Owing to its efficiency, it is considered a promising countermeasure against SCA. Since the number of shares has a significant impact to the implementation cost, constructing a target function with a minimum number of shares is a critical challenge in TI. It is known that, for an S-box with an algebraic degree of d, $d+1$ shares are sufficient [NRS11]. As $d \geq 2$ for non-linear functions, realizing 3-share TI has been an important research challenge. To reduce the number of shares, an S-box with a high algebraic degree (e.g., $d = 7$ for the AES S-box) is decomposed into sub-functions with lower algebraic degrees [MPL+11, BGN+15, GMK17, CRB+16, UHA17, WM18].

Unfortunately, a 3-share realization is not always available because of uniformity—a property regarding uniform distribution of shares. In particular, there have been no 3-share TIs for the NIST-standardized algorithms Keccak and AES. Therefore, fresh randomness is added during the execution to compensate for the lack of uniformity [MPL+11, BGN+15, UHA17]. The treatment is called remasking. Accordingly, 16 to 64 fresh random bits are required for a single S-box lookup as summarized in Table 1.

The cost of the fresh random bits is a serious concern. We refer the paper by Papagiannopoulos [Pap18] for a good survey on the cost of randomness in software platforms. Generating random numbers is a challenging task in hardware platforms, too. The conventional implementations in Table 1 consume dozens of random bits for each cycle. One possible way to generate random numbers at such a high rate is to use a low-latency cryptography. De Cnudde used an unrolled PRINCE implementation as a pseudo-random number generator [CRB+16][1]. An unrolled PRINCE implementation by the algorithm designers uses 8.2 [kGE] and 556 pJ/cycle (8.3 mW @ 67 ns) [BCG+12]. On the other hand, the proposed AES circuit which will appear in this paper use 9.9 pJ/cycle. That means the random number generator consumes 50 times more energy per cycle than the main AES circuit. That can be a serious problem in chips with limited energy or power budgets such as *near-field communication (NFC)* and battery-powered devices. Another way is to generate all the random bits in advance. However, the implementations in Table 1 consume 2,560–10,240 random bits for each encryption. To store these random bits, 17.9–71.7 [kGE] of registers are needed. That is more expensive than the main AES circuit again. Consequently, randomness optimization is considered as an important research challenge [BBP+16, FPS17, Pap18].

Recently, Daemen addressed the problem for Keccak by introducing a new technique called the *changing of the guards* [Dae17]. The idea behind the *changing of the guards* technique is to construct a TI of a layer of S-boxes instead of a distinct S-box. The technique enables uniform TI for a layer of any invertible S-boxes. Consequently, a 3-share

[1] Note that De Cnudde et al. used a round-reduced PRINCE, but the number of rounds was not given.

uniform TI is successfully constructed for the Keccak S-box. However, there is an obstacle to applying the method to the AES S-box. As discussed previously, the AES S-box should be decomposed to reduce the number of shares. Such a decomposed S-box involves 2-input multiplication that is non-invertible, and thus the *changing of the guards* technique cannot be applied.

More recently, Wegener et al. decomposed the AES S-box into stages having an algebraic degree of 3 in such a way that the *changing of the guards* technique is applicable. That enabled the first 4-share uniform sharing for the AES S-box [WM18]. However, implementing a 4-share TI is expensive: a straightforward implementation required more than 20 [kGE] for a single S-box. Wegener et al. tackled the problem by proposing a sophisticated circuit architecture that can be realized with only 4.2 [kGE]. However, the area reduction is achieved at the cost of long latency as shown in Table 1. Therefore, a 3-share uniform TI of the AES S-box is still an important challenge.

Contributions In this study, we address the problem by introducing a certain generalization of the *changing of the guards* technique, which provides a generic way to construct a uniform sharing for any function. The proposed method is based on transforming a target function into an invertible one while maintaining its essential functionality unchanged. Subsequently, a TI of the transformed function is constructed. We propose the first 3-share uniform TI of the AES S-box based on the proposed technique. Subsequently, we design and evaluate a concrete 3-share AES implementation with the proposed S-box technique. Owing to the smaller number of shares, the cost of the proposed design is smaller than 1/4 that of the conventional design by Wegener et al. [WM18] in terms of the area–latency product as shown in Table 1. The security of the proposed method is evaluated through theoretical analysis and a simulation-based leakage assessment.

Organization The rest of this paper is organized as follows. In Sect. 2, we briefly review the conventional works: TI, the *changing of the guards* technique, and the Canright's S-box implementation [Can05]. Subsequently, the proposed method is described in Sect. 3.1. Its relation to the *changing of the guards* technique is discussed in Sect. 3.2. The proposed method is applied to the Canright's AES S-box implementation in Sect. 4.1 and 4.2. The overall design and analysis of the proposed AES S-box implementation are shown in Sect. 4.3. Sect. 4.4 shows an AES circuit with the proposed S-box implementation. Sect. 4.4 also provides the performance and security evaluation of the AES circuit. All the proofs are given in the Appendix.

2 Preliminary

2.1 Notation

In this paper, we discuss shares with three elements unless otherwise stated. Given a variable x, its share is represented as

$$\overline{x} = [x_a, x_b, x_c] \quad \text{s.t.} \quad x = x_a + x_b + x_c.$$

Some proper subsets of $\overline{x}$ are denoted by

$$\tilde{x}_a = [x_b, x_c], \qquad \tilde{x}_b = [x_a, x_c], \qquad \tilde{x}_c = [x_a, x_b].$$

The terms "function" and "mapping" are used interchangeably. Sharing of a mapping ψ is given by a set of component mappings $\{\psi_a, \psi_b, \psi_c\}$. The inputs and outputs of a mapping are denoted by small and large letters, respectively. A mapping from x to X can be expressed as either $X = \psi(x)$ or $x \overset{\psi}{\longmapsto} X$. We denote addition over $GF(2)$ by $+$.

2.2 Threshold Implementation

A mapping from x to X namely $X = \psi(x)$ is considered. In TI, a variable x is represented as a share $\overline{x} = [x_a, x_b, x_c]$ s.t. $x = x_a + x_b + x_c$. The overall cryptographic algorithm is implemented by using shares without reconstructing the original values. To ensure the requirement, ψ is split into three component functions namely

$$X_a = \psi_a(\tilde{x}_a), \qquad X_b = \psi_b(\tilde{x}_b), \qquad X_c = \psi_c(\tilde{x}_c), \tag{1}$$

where $[X_a, X_b, X_c]$ is an output share. In TI, there are three important properties namely *correctness*, *non-completeness*, and *uniformity*.

Correctness The sharing $\{\psi_a, \psi_b, \psi_c\}$ is said to be correct if the output share represents the output of the original function ψ, i.e.,

$$\psi(x) = X = X_a + X_b + X_c = \psi_a(\tilde{x}_a) + \psi_b(\tilde{x}_b) + \psi_c(\tilde{x}_c).$$

Non-Completeness The sharing $\{\psi_a, \psi_b, \psi_c\}$ is said to be non-complete if each of $\{\psi_a, \psi_b, \psi_c\}$ uses only a proper subset of the input share $[x_a, x_b, x_c]$. The component functions shown in Eq. (1) are non-complete because the component functions ψ_a, ψ_b, and ψ_c are independent of x_a, x_b, and x_c, respectively.

Uniformity The probabilistic distributions of the original and shared inputs are denoted by $P_I(x)$ and $\overline{P}_I(\overline{x})$, respectively. The input share is said to be uniform if and only if, for any x, its shares occur at the same probability

$$\overline{P}_I(\overline{x}) = \frac{P_I(x)}{\alpha} = \frac{P_I(x_a + x_b + x_c)}{\alpha} \tag{2}$$

where α is a constant. Let $P_O(X)$ be the distribution of the original output, and let $\overline{P}_O(\overline{X})$ be the distribution of output shares, given uniform input shares. The sharing $\{\psi_a, \psi_b, \psi_c\}$ is said to be uniform if and only if, for any output X, its shares occur at the same probability

$$\overline{P}_O(\overline{X}) = \frac{P_O(X)}{\alpha} = \frac{P_O(X_a + X_b + X_c)}{\alpha}. \tag{3}$$

Security of (1st order) TI is proved based on the single probing model [NRS11] in which an attacker is allowed to probe an arbitrary wire in a target. Non-completeness guarantees that leakage from either ψ_a, ψ_b, or ψ_c is independent of x. This is because each component function has only a proper subset of the input share. More specifically, there is no information leakage about the original input if the input share is uniform. Uniformity ensures that shares are distributed uniformly even after component functions are applied.

For a mapping having an algebraic degree of d, correct and non-complete sharing can be constructed with $d + 1$ shares [NRS11]. The number of shares is a significant concern because the implementation cost increases exponentially to d [Dae17]. Therefore, an original function is usually decomposed into sub-functions having lower algebraic degrees to reduce the number of shares [MPL+11, BGN+15, CRB+16, UHA17]. As $d \geq 2$ for non-linear functions, TI with three shares has been an important research challenge. There is a recent study on a countermeasure with a smaller number of shares [CRB+16]. However, three is still the minimum number of shares that can satisfy uniformity.

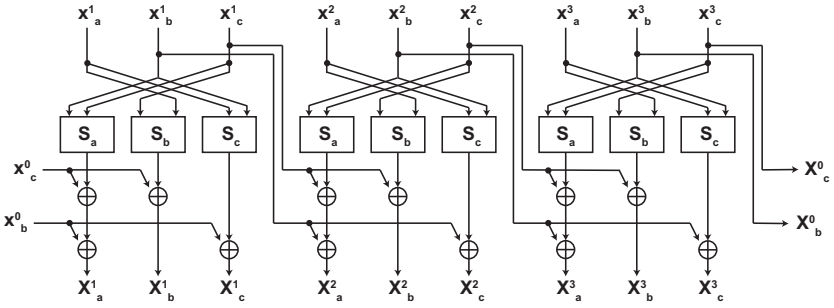

Figure 1: *Changing of the Guards sharing*

2.3 Changing of the Guards

Recently, Daemen introduced a technique called the *changing of the guards* that enables a 3-share uniform TI for the Keccak S-box [Dae17]. The technique can be applied to any invertible S-box. Let S be an invertible S-box. Assume S has a correct and non-complete (but non-uniform) sharing given by

$$\tilde{x}_a \overset{S_a}{\longmapsto} X_a, \qquad \tilde{x}_b \overset{S_b}{\longmapsto} X_b, \qquad \tilde{x}_c \overset{S_c}{\longmapsto} X_c.$$

A layer comprising L-parallel S-boxes is considered. The layer maps from $[x^1, \ldots, x^L]$ to $[X^1, \ldots, X^L]$ where $X^i = S(x^i)$ i.e., x^i and X^i are input and output of the i-th S-box, respectively. The *changing of the guards* sharing of the S layer is defined as follows:

Definition 1 (Changing of the guards [Dae17])**.** *The changing of the guards sharing of the S-box layer mapping from* $[[x_a^1, x_b^1, x_c^1], \ldots, [x_a^L, x_b^L, x_c^L]]$ *to* $[[X_a^1, X_b^1, X_c^1], \ldots, [X_a^L, X_b^L, X_c^L]]$ *is given by*

$$X_a^i = S_a(\tilde{x}_a^i) + x_b^{i-1} + x_c^{i-1}, \qquad X_b^i = S_b(\tilde{x}_b^i) + x_c^{i-1}, \qquad X_c^i = S_c(\tilde{x}_c^i) + x_b^{i-1}$$

for $i > 0$ *and* $X_b^0 = x_b^L$, $X_c^0 = x_c^L$.

Fig. 1 shows a diagram of the *changing of the guards* sharing for $L = 3$. The idea behind the *changing of the guards* is to obtain a correct, non-complete, and uniform TI of a layer of S-boxes instead of a distinct S-box. An essential part is to remask the outputs from the component functions by using a neighboring share. More specifically, the non-uniform output share $[S_a(\tilde{x}_a^i), S_b(\tilde{x}_b^i), S_c(\tilde{x}_c^i)]$ is remasked by adding a share $[x_b^{i-1} + x_c^{i-1}, x_c^{i-1}, x_b^{i-1}]$ that represents 0 and is generated by the neighboring input share.

Although the AES S-box is invertible, a straightforward application of the *changing of the guards* is very inefficient [Dae17]. This is because the AES S-box has an algebraic degree of 7 and thus, 8 shares are required. Conventionally, the AES S-box is decomposed into sub-functions having lower algebraic degrees. However, the *changing of the guards* technique cannot be applied to such a decomposed AES S-box. The difficulty stays in 2-input multiplication, used in the decomposed S-boxes, which is non-invertible because of different input and output sizes. Therefore, efficient application of the *changing of the guards* technique to the AES S-box remains open [Dae17].

More recently, Wegener et al. proposed the decomposition of the AES S-box into invertible sub-functions having an algebraic degree of 3 [WM18]. By applying the *changing of the guards* technique to the decomposed S-box, the first 4-share uniform TI for the AES S-box is proposed. However, 3-share uniform TI for the AES S-box remains open.

Table 2: Irreducible polynomials and normal bases for the tower field representation

Extension	Irreducible polynomial	Normal basis
$GF(2^8)/GF(2^4)$	$r(y) = y^2 + y + \mu$	$[Y, Y^{16}]$
$GF(2^4)/GF(2^2)$	$s(z) = z^2 + z + N$	$[Z, Z^4]$
$GF(2^2)/GF(2)$	$t(w) = w^2 + w + 1$	$[W, W^2]$

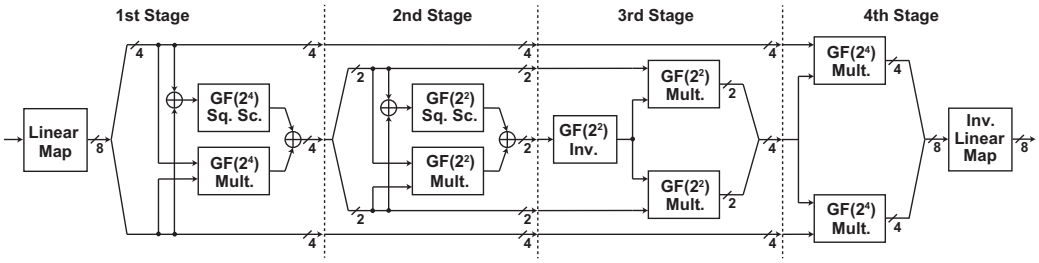

Figure 2: Inversion over $GF(2^8)$ in four stages

2.4 Canright's AES S-box Implementation

The AES S-box is defined based on inversion over $GF(2^8)$. Efficient implementations exploiting its algebraic property have been studied so far. Notably, Canright proposed a compact implementation based on tower field representation with normal bases [Can05]. Table 2 summarizes the field extensions and bases used in the Canright's S-box implementation.

The inverse of $x \in GF(2^8)$ is considered. There exist unique $\alpha, \beta \in GF(2^4)$ such that $x = \alpha Y + \beta Y^{16}$. x^{-1} is obtained as

$$x^{-1} = (\alpha Y + \beta Y^{16})^{-1} = (\theta^{-1}\beta)Y + (\theta^{-1}\alpha)Y^{16} \tag{4}$$

$$\text{where} \quad \theta = \alpha\beta + (\alpha + \beta)^2\mu \in GF(2^4). \tag{5}$$

The inverse of $\theta \in GF(2^4)$ is obtained similarly. There exist unique $a, b \in GF(2^2)$ such that $\theta = aZ + bZ^4$. θ^{-1} is obtained as

$$\theta^{-1} = (aZ + bZ^4)^{-1} = (\zeta^{-1}b)Z + (\zeta^{-1}a)Z^4, \tag{6}$$

$$\text{where} \quad \zeta = ab + (a + b)^2 N \in GF(2^2). \tag{7}$$

There exist $s, t \in GF(2)$ such that $\zeta = sW + tW^2$. Here, the inverse is easily obtained as

$$(sW + tW^2)^{-1} = (sW + tW^2)^2 = tW + sW^2. \tag{8}$$

Fig. 2 shows a circuit diagram for the inversion over $GF(2^8)$ as described above. Note that the operations for $(a + b)^2 N$ and $(\alpha + \beta)^2\mu$ are called *squaring and scaling* and are indicated by `Sq.Sc.` in Fig. 2.

In this study, we consider the 4-stage partitioning as shown in Fig. 2. The partitioning is based on the design by De Cnudde et al. [CRB+16] because its symmetry is appropriate for our design. However, the number of stages is changed from 6 to 4 by merging the linear maps to the neighboring stages to reduce latency. The 1st and 2nd stages are devoted to calculating Eq. (5) and Eq. (7), respectively. The 3rd and 4th stages correspond to Eq. (6) and Eq. (4), respectively. The linear maps for the affine transformation and field isomorphism are placed in the 1st and 4th stages.

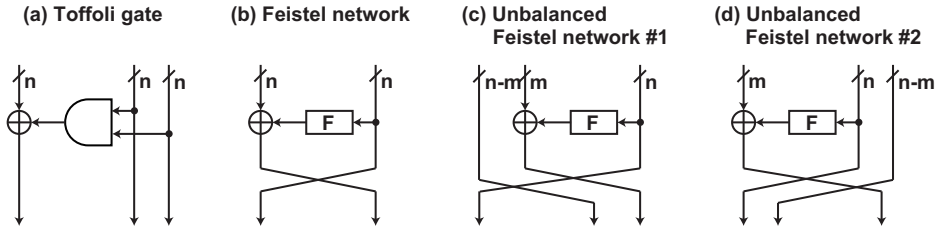

Figure 3: Construction of invertible functions from functions

3 Proposed Technique

3.1 Extension and Restriction

Uniformity is closely related to invertibility. On the one hand, a sharing is uniform if it is invertible. On the other hand, the *changing of the guards* technique can be used if a target function is invertible. The basic idea behind the proposed method is to extend a target function into an invertible one. Invertibility ensures uniform sharing of the extended function.

Obtaining an invertible function from a non-invertible function is the main topic of *reversible computing* in which computers composed of invertible (i.e., reversible) primitives are studied for quantum and energy-efficient computing [Tah16]. A common strategy for obtaining a reversible circuit is to add additional input and output to carry sufficient information required for inversion. An important example is a reversible 2-input AND gate known as Toffoli gate, shown in Fig. 3-(a). The Toffoli gate has one additional input and two additional outputs for the sake of invertibility.

Fig 3-(b) shows the Feistel network, which is also a technique to obtain an invertible function (i.e., permutation) from a non-invertible F-function. The original Feistel network requires an F-function having the same input and output sizes. Fig 3-(c) and -(d) shows a generalization called the unbalanced Feistel network [SK96]. The unbalanced Feistel network enables the construction of an invertible function from an F-function having different input and output sizes.

Let $\psi : \{0,1\}^n \to \{0,1\}^m$ be a target function having a 3-share correct and non-complete sharing. ψ is extended to an invertible mapping in the same manner as the Toffoli gate and Feistel network

Definition 2 (Extended mapping)**.** *Let $x, X \in \{0,1\}^n$ and $y, Y \in \{0,1\}^m$. The extended mapping $[x,y] \overset{\psi^E}{\longmapsto} [X,Y]$ is defined by*

$$X = x, \qquad Y = \psi(x) + y.$$

Fig. 4-(left) shows a diagram for ψ^E. We further define a sharing of ψ^E denoted by $\{\psi_a^E, \psi_b^E, \psi_c^E\}$.

Definition 3 (Sharing of the extended mapping)**.** *A sharing $\{\psi_a^E, \psi_b^E, \psi_c^E\}$ such that*

$$[\tilde{x}_a, y_a] \overset{\psi_a^E}{\longmapsto} [X_a, Y_a], \qquad [\tilde{x}_b, y_b] \overset{\psi_b^E}{\longmapsto} [X_b, Y_b], \qquad [\tilde{x}_c, y_c] \overset{\psi_c^E}{\longmapsto} [X_c, Y_c]$$

is given by

$$\begin{aligned} X_a &= x_b, & Y_a &= \psi_a(\tilde{x}_a) + y_a, \\ X_b &= x_c, & Y_b &= \psi_b(\tilde{x}_b) + y_b, \\ X_c &= x_a, & Y_c &= \psi_c(\tilde{x}_c) + y_c. \end{aligned} \tag{9}$$

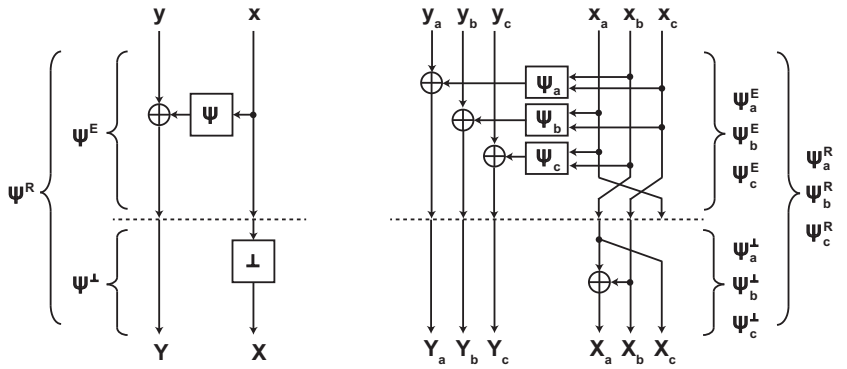

Figure 4: Extension and restriction of a mapping (left) and the corresponding sharing (right)

Fig. 4-(right) shows a diagram for $\{\psi_a^E, \psi_b^E, \psi_c^E\}$. Fig. 4-(right) shows that the invertible structure is preserved in sharing. Therefore, $\{\psi_a^E, \psi_b^E, \psi_c^E\}$ is invertible and thus uniform.

Lemma 1. *$\{\psi_a^E, \psi_b^E, \psi_c^E\}$ is a correct, non-complete, and uniform sharing of ψ^E.*

As discussed previously, the input shares should be uniform for security. If the condition is satisfied, it implies that the input shares $\overline{x}$ and $\overline{y}$ in Definition 3 are independent. It is interesting to remark that, the discussion on the uniformity of the generalized Feistel Network by Faust et al. [FGP+18], found independently in the context of high-order masking, closely relates to this Lemma.

So far, ψ is extended to ψ^E by adding m-bit additional input and n-bit additional output. ψ^E provides an expected output only when the additional input satisfies $y = 0$. Similarly, its sharing $\{\psi_a^E, \psi_b^E, \psi_c^E\}$ requires an input share satisfying $y_a + y_b + y_c = 0$ to provide a correct output. Here, we consider the method to obtain such an additional input share. The idea is to convert the unnecessary additional output $[X_a, X_b, X_c]$ to a share representing zero in the same way as the original *changing of the guards* technique.

We first consider the following map $\psi^\perp$:

Definition 4. *Let $x, X \in \{0,1\}^n$ and $y, Y \in \{0,1\}^m$. A mapping $[x, y] \overset{\psi^\perp}{\longmapsto} [X, Y]$ is given by*

$$X = 0 \qquad\qquad Y = y.$$

A diagram for $\psi^\perp$ is shown in Fig. 4-(left) in which $\perp$ represents a zero function such that $\perp(x) = 0$ for all x. Subsequently, its sharing is considered.

Definition 5. *A sharing $\{\psi_a^\perp, \psi_b^\perp, \psi_c^\perp\}$ such that*

$$[\tilde{x}_c, y_a] \overset{\psi_a^\perp}{\longmapsto} [X_a, Y_a], \qquad [\tilde{x}_a, y_b] \overset{\psi_b^\perp}{\longmapsto} [X_b, Y_b], \qquad [\tilde{x}_b, y_c] \overset{\psi_c^\perp}{\longmapsto} [X_c, Y_c]$$

is given by

$$X_a = x_a + x_b, \quad X_b = x_b, \quad X_c = x_a, \quad Y_a = y_a, \quad Y_b = y_b, \quad Y_c = y_c.$$

In the sharing form, the zero function $\perp$ is realized by $[X_a, X_b, X_c] = [x_a + x_b, x_b, x_a]$ in the same way as the original *changing of the guards*. The probability of observing an output share $[X_a, X_b, X_c]$ is constant because

$$\overline{P}_O(X_a, X_b, X_c) = \sum_t \overline{P}_I(X_c, X_b, t) = \sum_t \frac{P_I(X_c + X_b + t)}{\alpha} = constant. \tag{10}$$

Therefore, the sharing is uniform.

Lemma 2. *$\{\psi_a^\perp, \psi_b^\perp, \psi_c^\perp\}$ is a correct, non-complete, and uniform sharing of $\psi^\perp$.*

Finally, we define a restricted mapping composed of ψ^E and $\psi^\perp$.

Definition 6 (Restricted mapping)**.** *A restricted mapping ψ^R is defined as $\psi^R = \psi^\perp \circ \psi^E$.*

Fig. 4-(left) shows a diagram for ψ^R. A sharing of the restricted mapping is considered.

Definition 7 (A sharing of the restricted mapping)**.** *A sharing $\{\psi_a^R, \psi_b^R, \psi_c^R\}$ s.t.*

$$[\tilde{x}_a, y_a] \xmapsto{\psi_a^R} [X_a, Y_a], \qquad [\tilde{x}_b, y_b] \xmapsto{\psi_b^R} [X_b, Y_b], \qquad [\tilde{x}_c, y_c] \xmapsto{\psi_c^R} [X_c, Y_c]$$

is defined as $\psi_a^R = \psi_a^\perp \circ \psi_a^E$, $\psi_b^R = \psi_b^\perp \circ \psi_b^E$, and $\psi_c^R = \psi_c^\perp \circ \psi_c^E$, or equivalently

$$\begin{aligned} X_a &= x_b + x_c, & Y_a &= \psi_a(\tilde{x}_a) + y_a, \\ X_b &= x_c, & Y_b &= \psi_b(\tilde{x}_b) + y_b, \\ X_c &= x_b, & Y_c &= \psi_c(\tilde{x}_c) + y_c. \end{aligned}$$

$\{\psi_a^R, \psi_b^R, \psi_c^R\}$ is uniform because it is composed of the uniform sharings $\{\psi_a^E, \psi_b^E, \psi_c^E\}$ and $\{\psi_a^\perp, \psi_b^\perp, \psi_c^\perp\}$ as shown in Fig. 4-(right).

Theorem 1. *$\{\psi_a^R, \psi_b^R, \psi_c^R\}$ is a correct, non-complete, and uniform sharing of ψ^R.*

Consequently, a uniform sharing with an additional output share $[X_a, X_b, X_c]$ satisfying $X_a + X_b + X_c = 0$ is obtained. The additional output is used as an additional input to the next sharing. Thus, ψ^R can be calculated without using fresh randomness. The above discussion provides a generic way of constructing a uniform sharing for any function.

Security Claim The proposed method is about constructing a uniform sharing. Therefore, the constructed sharings are secure up to the 1st-order probing model with glitches similarly to the original TI and the original *changing of the guards*. This is because uniformity is not enough for resistance against high-order attacks. Its extension to either 2-share schemes or high-order schemes are non-trivial and opened for future research.

Related works There are conventional works on recycling randomness in high-order masking schemes [FPS17, Pap18]. The original *changing of the guards* technique, as well as the proposed generalization, are different from these works on the point they focus on constructing uniform sharing. In other words, in the original and generalized *changing of the guards* techniques, randomness is treated similarly to other inputs and recycling is a part of a uniform sharing.

3.2 The Changing of the Guards Revisited

The proposed method is a generalization of the *changing of the guards* technique. Assume that a target mapping ψ has the same input and output size i.e., $n = m$. We consider connecting the restricted mapping ψ^R in cascade as shown in Fig 5-(left). The mapping is written as

$$[0, x^1, x^2, x^3] \longmapsto [X^1, X^2, X^3, 0] \qquad \text{s.t.} \quad X^i = \psi(x^i).$$

Recall that ψ^R has a uniform sharing $\{\psi_a^R, \psi_b^R, \psi_c^R\}$ according to Theorem 1. By substituting ψ^R with $\{\psi_a^R, \psi_b^R, \psi_c^R\}$, we obtain a sharing as shown in Fig. 5-(right). By construction, the sharing in Fig 5-(right) is uniform. Note that a glitch should be considered when non-linear layers are connected. However, this is not a problem for the connection

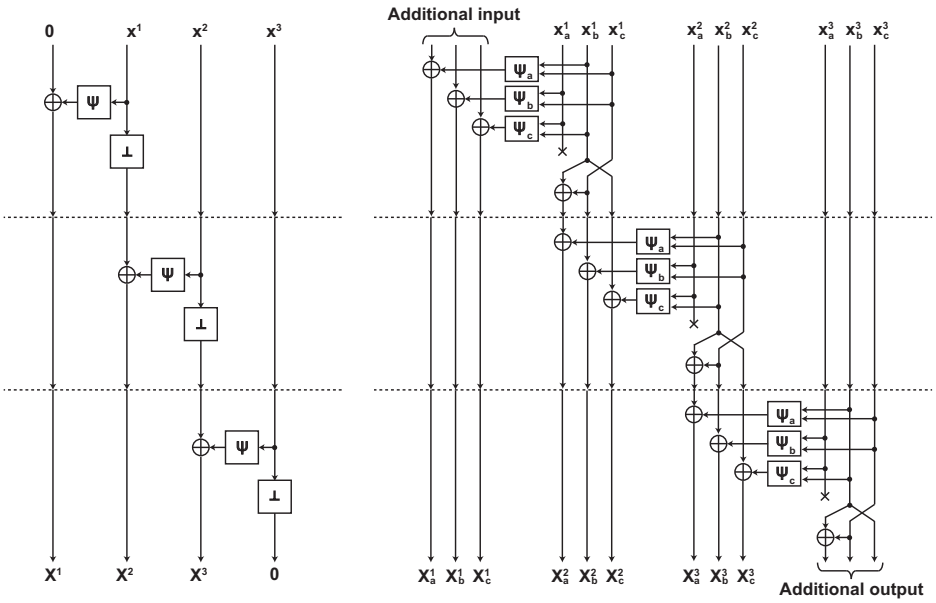

Figure 5: Cascaded ψ^R (left) and its sharing (right)

shown in Fig 5-(right) because there is only XOR after the component functions ψ_a, ψ_b, and ψ_c.

Notice that the sharing in Fig 5-(right) and the *changing of the guards* sharing in Fig. 1 are identical. In other words, the above discussion provides yet another proof of the uniformity of the *changing of the guards* sharing. However, it implies more because ψ is not necessarily invertible in the new proof. Furthermore, the proposed method can be used for a target mapping ψ having different input and output sizes. Its concrete example is given in Sect. 4.2.

4 Application to Canright's S-box Implementation

A 3-share threshold implementation of the Canright's S-box implementation, discussed in Sect. 2.4, is designed based on the proposed technique. As we are interested in the non-linear functions, we ignore the linear mappings in the following discussion.

4.1 1st and 2nd Stages

The 2nd stage shown in Fig. 2 is considered. The same discussion applies to the 1st stage for $GF(2^4)$. The stage can be expressed as

$$[x, y] \longmapsto [x, y, f(x, y)] \quad \text{s.t.} \quad f(x, y) = xy + (x + y)^2 N.$$

Note that f is a mapping for obtaining ζ in Eq. (7). x and y are preserved for a later stage. There is no uniform sharing for a mapping having an output larger than an input [WM18]. Now, we consider an extended mapping of f given by

$$[x, y, z] \overset{f^E}{\longmapsto} [x, y, f(x, y) + z]. \tag{11}$$

As f has an algebraic degree of 2, there is a 3-share correct and non-complete sharing $\{f_a, f_b, f_c\}$. Based on Theorem 1, a uniform sharing of f^E can be constructed.

Definition 8 (Sharing of f^E)**.** *A sharing* $\{f_a^E, f_b^E, f_c^E\}$ *s.t.*

$$[\tilde{x}_a, \tilde{y}_a, z_a] \overset{f_a^E}{\longmapsto} [X_a, Y_a, Z_a], \quad [\tilde{x}_b, \tilde{y}_b, z_b] \overset{f_b^E}{\longmapsto} [X_b, Y_b, Z_b], \quad [\tilde{x}_c, \tilde{y}_c, z_c] \overset{f_c^E}{\longmapsto} [X_c, Y_c, Z_c]$$

is given by

$$\begin{aligned} X_a &= x_b, & Y_a &= y_b, & Z_a &= f_a(\tilde{x}_a, \tilde{y}_a) + z_a, \\ X_b &= x_c, & Y_b &= y_c, & Z_b &= f_b(\tilde{x}_b, \tilde{y}_b) + z_b, \\ X_c &= x_a, & Y_c &= y_a, & Z_c &= f_c(\tilde{x}_c, \tilde{y}_c) + z_c. \end{aligned} \tag{12}$$

Corollary 1. $\{f_a^E, f_b^E, f_c^E\}$ *is a correct, non-complete, and uniform sharing of* f^E*.*

Note that there is no additional output because both $[X_a, X_b, X_c]$ and $[Y_a, Y_b, Y_c]$ should be preserved for later use. Therefore, an additional input $[z_a, z_b, z_c]$ satisfying $z_a + z_b + z_c = 0$ should be supplied from outside so that the stage works correctly. As discussed in the next section, there are additional outputs obtained in the 3rd and 4th stages. By using them as additional inputs, the 1st and 2nd stages can be executed without using fresh randomness.

4.2 3rd and 4th Stages

The 3rd stage shown in Fig. 2 is considered. The same discussion applies to the 4th stage for $GF(2^4)$. In this stage, $\zeta^{-1}b$ and $\zeta^{-1}a$ are obtained given a, b, and ζ (see Eq. (6)). Transforming from ζ to ζ^{-1} is linear and thus easy to implement, as discussed in Eq. (8). Therefore, we consider the remaining part.

The stage is represented by a mapping h given by

$$[x, y, z] \overset{h}{\longmapsto} [g(x, z), g(y, z)] \quad \text{s.t.} \quad g(x, y) = xy. \tag{13}$$

As g has an algebraic degree of 2, there is a correct and non-complete sharing $\{g_a, g_b, g_c\}$. However, $\{g_a, g_b, g_c\}$ cannot be uniform because it is 2-input multiplication over $GF(2^2)$ [NRS11]. Therefore, the technique discussed in Sect. 3.2 is considered. Note that this is an example of the generalized *changing of the guards* for a target function having different input and output sizes ($n > m$). The extended and restricted mappings of h are given by

$$\begin{aligned} [x, y, z, v, w] &\overset{h^E}{\longmapsto} [g(x, z) + v, g(y, z) + w, x, y, z], \\ [x, y, z, v, w] &\overset{h^R}{\longmapsto} [g(x, z) + v, g(y, z) + w, 0, 0, 0]. \end{aligned}$$

h^R has a uniform sharing namely $\{h_a^R, h_b^R, h_c^R\}$ according to Theorem 1.

Definition 9. *A sharing* $\{h_a^R, h_b^R, h_c^R\}$ *s.t.*

$$\begin{aligned} [\tilde{x}_a, \tilde{y}_a, \tilde{z}_a, v_a, w_a] &\overset{h_a^R}{\longmapsto} [X_a, Y_a, Z_a, V_a, W_a], \\ [\tilde{x}_b, \tilde{y}_b, \tilde{z}_b, v_b, w_b] &\overset{h_b^R}{\longmapsto} [X_b, Y_b, Z_b, V_b, W_b], \\ [\tilde{x}_c, \tilde{y}_c, \tilde{z}_c, v_c, w_c] &\overset{h_c^R}{\longmapsto} [X_c, Y_c, Z_c, V_c, W_c] \end{aligned}$$

is given by

$$\begin{aligned} V_a &= g_a(\tilde{x}_a, \tilde{z}_a) + v_a, & W_a &= g_a(\tilde{y}_a, \tilde{z}_a) + w_a, & X_a &= x_b + x_c, & Y_a &= y_b + y_c, & Z_a &= z_b + z_c, \\ V_b &= g_b(\tilde{x}_b, \tilde{z}_b) + v_b, & W_b &= g_b(\tilde{y}_b, \tilde{z}_b) + w_b, & X_b &= x_c, & Y_b &= y_c, & Z_b &= z_c, \\ V_c &= g_c(\tilde{x}_c, \tilde{z}_c) + v_c, & W_c &= g_c(\tilde{y}_c, \tilde{z}_c) + w_c, & X_c &= x_b, & Y_c &= y_b, & Z_c &= z_b. \end{aligned}$$

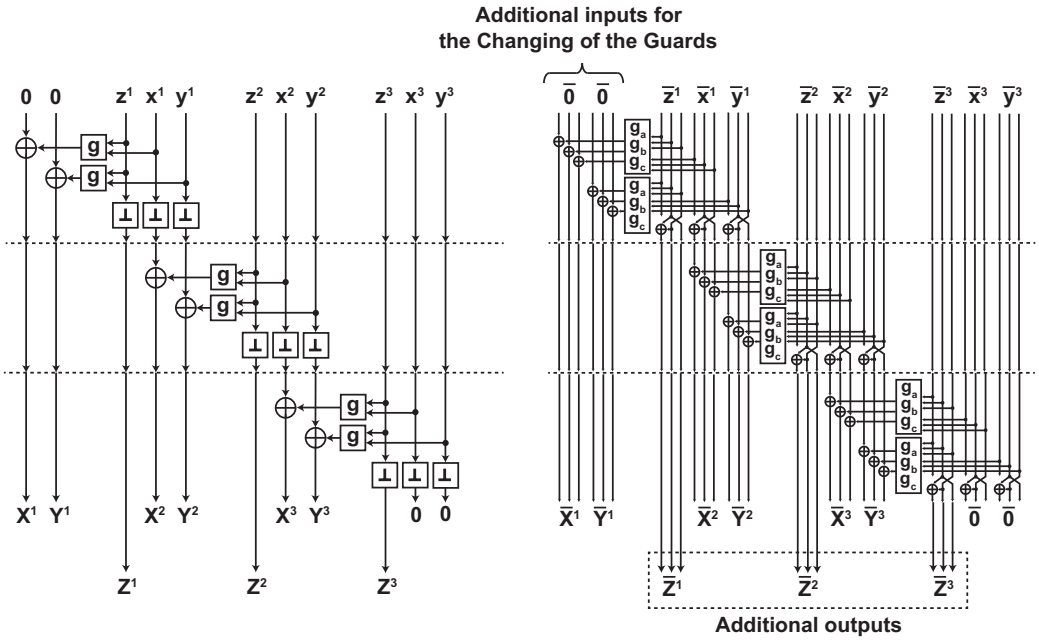

Figure 6: Cascaded h^R (left) and its sharing (right)

Subsequently, we consider connecting the multiple h^R in cascade as shown in Fig 6-(left). By substituting h^R with $\{h_a^R, h_b^R, h_c^R\}$ in Fig. 6-(left), we obtain the one in Fig. 6-(right).

Definition 10 (The generalized *changing of the guards* sharing of h^R)**.** *The generalized changing of the guards sharing of the h^R mapping from $\{[\overline{x}^1, \overline{y}^1, \overline{z}^1], \ldots, [\overline{x}^L, \overline{y}^L, \overline{z}^L]\}$ to $\{[\overline{X}^1, \overline{Y}^1, \overline{Z}^1], \ldots, [\overline{X}^L, \overline{Y}^L, \overline{Z}^L]\}$ is given by*

$$
\begin{array}{lll}
X_a^i = g_a(\tilde{x}_a^i, \tilde{z}_a^i) + x_b^{i-1} + x_c^{i-1}, & Y_a^i = g_a(\tilde{y}_a^i, \tilde{z}_a^i) + y_b^{i-1} + y_c^{i-1}, & Z_a^i = z_b^i + z_c^i \\
X_b^i = g_b(\tilde{x}_b^i, \tilde{z}_b^i) + x_c^{i-1}, & Y_b^i = g_b(\tilde{y}_b^i, \tilde{z}_b^i) + y_c^{i-1}, & Z_b^i = z_c^i \\
X_c^i = g_c(\tilde{x}_c^i, \tilde{z}_c^i) + x_b^{i-1} & Y_c^i = g_c(\tilde{y}_c^i, \tilde{z}_c^i) + y_b^{i-1} & Z_c^i = z_b^i
\end{array}
$$

for $i \in [1, L]$ and $X_b^0 = x_b^L$, $Y_b^0 = y_b^L$, $X_c^0 = x_c^L$, $Y_c^0 = y_c^L$.

Fig. 7 shows a detailed diagram of the sharing in Definition 10. The sharing in Definition 10 is uniform through construction as discussed in Sect. 3.2. It can also be understood in comparison with the original *changing of the guards*. The upper half of Fig. 7 is a straightforward application of the *changing of the guards* except that there is no guard for Z_a^i, Z_b^i, and Z_c^i. Therefore, we can show that the upper half is invertible in the same manner as the original work [Dae17]. The lower half is $\{\psi_a^\perp, \psi_b^\perp, \psi_c^\perp\}$ in Definition 5 and thus preserves uniformity.

Corollary 2. *Definition 10 is a correct, non-complete, and uniform TI of the following mapping:*

$$
\begin{aligned}
[[x^1, y^1, z^1], \ldots, [x^L, y^L, z^L]] \longmapsto [[X^1, Y^1, Z^1], \ldots, [X^L, Y^L, Z^L]] \\
s.t. \quad X^i = g(x^i, z^i), Y^i = g(y^i, z^i), Z^i = 0.
\end{aligned}
$$

Notably, the unbalance between input and output sizes results in additional outputs $[Z_a^i, Z_b^i, Z_c^i]$ satisfying $Z_a^i + Z_b^i + Z_c^i = 0$. They can be used in the 1st and 2nd stages as additional inputs.

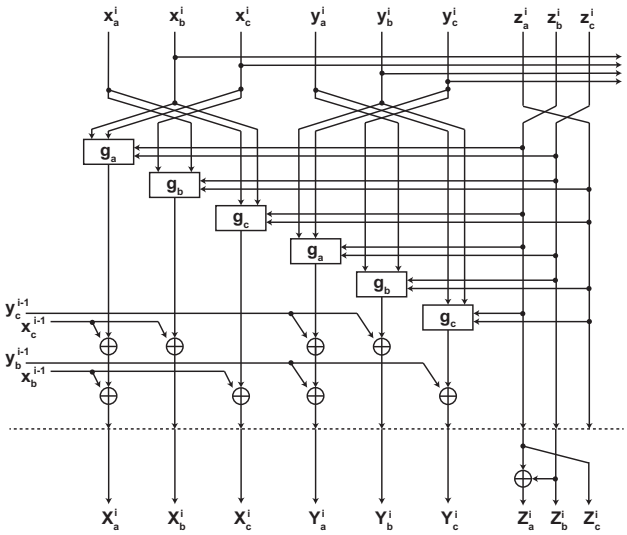

Figure 7: *Changing of the guards* sharing of h^R (Definition 10)

4.3 Putting it Together

Fig. 8 shows the threshold implementation of the AES S-box based on the proposed method. The datapath width is 42 bits: $3 \times 8 = 24$ bits for a 3-share 8-bit AES state, 3×4 bits for an additional input in $GF(2^4)$, and 3×2 bits for an additional input in $GF(2^2)$. At the final stage, a share representing an S-box output is obtained along with $3 \times (4+2) = 18$-bit additional outputs. As discussed previously, the additional output is forwarded to the next S-box calculation as an additional input. Therefore, fresh randomness is required only for the first additional input and is not required during execution.

Note that there are two ways of remasking in the 3rd and 4th stages. One way is $[x_b^{i-1} + x_c^{i-1}, x_c^{i-1}, x_b^{i-1}]$ and $[y_b^{i-1} + y_c^{i-1}, y_c^{i-1}, y_b^{i-1}]$ in Definition 10. They are forwarded from a previous cycle using the temporary registers in between the 2nd/3rd and 3rd/4th stages. Another way is the additional output $[Z_a^i, Z_b^i, Z_c^i]$ in Definition 10. They are carried to the end of the pipeline and then used in another S-box calculation in the next AES round.

Here, we show that the proposed S-box implementation satisfies non-completeness. Table 3 shows the relationship between the intermediate values and the inputs. The rows represent the intermediate values: (i) the stage input/output shares $\{X_a^i, X_b^i, X_c^i\}$, $\{Y_a^i, Y_b^i, Y_c^i\}$, and $\{Z_a^i, Z_b^i, Z_c^i\}$, and (ii) the outputs from the non-linear functions $\{t_a^i, t_b^i, t_c^i\}$. The columns represent the input shares to the S-box implementation: $\{x_a, x_b, x_c\}$, $\{y_a, y_b, y_c\}$, $\{z_a, z_b, z_c\}$, $\{v_a, v_b, v_c\}$, and $\{w_a, w_b, w_c\}$. Note that $\{v_a, v_b, v_c\}$ and $\{w_a, w_b, w_c\}$ are the masks for the *changing of the guards*. The intermediate values and inputs are also indicated in Fig. 8. $\diamondsuit$ or ♠ in the table shows that the intermediate value depends on the input. In particular, ♠ shows that the intermediate value is either (i) masked by the input share or (ii) the input itself.

In the proposed design, the outputs from the non-linear functions are immediately refreshed by adding some of the input shares. More specifically, for any i, the stage input/output shares $\{X_a^i, X_b^i, X_c^i\}$, $\{Y_a^i, Y_b^i, Y_c^i\}$, and $\{Z_a^i, Z_b^i, Z_c^i\}$ have distinct masks i.e., having ♠ on different columns on Table 3. Therefore, in each stage, there is no information leakage unless all the three elements of an input share of the stage are combined. However, the condition is never satisfied because each stage satisfies non-completeness. Consequently, the proposed S-box implementation satisfies non-completeness. The non-completeness ensures the security of the proposed implementation in the presence of glitches.

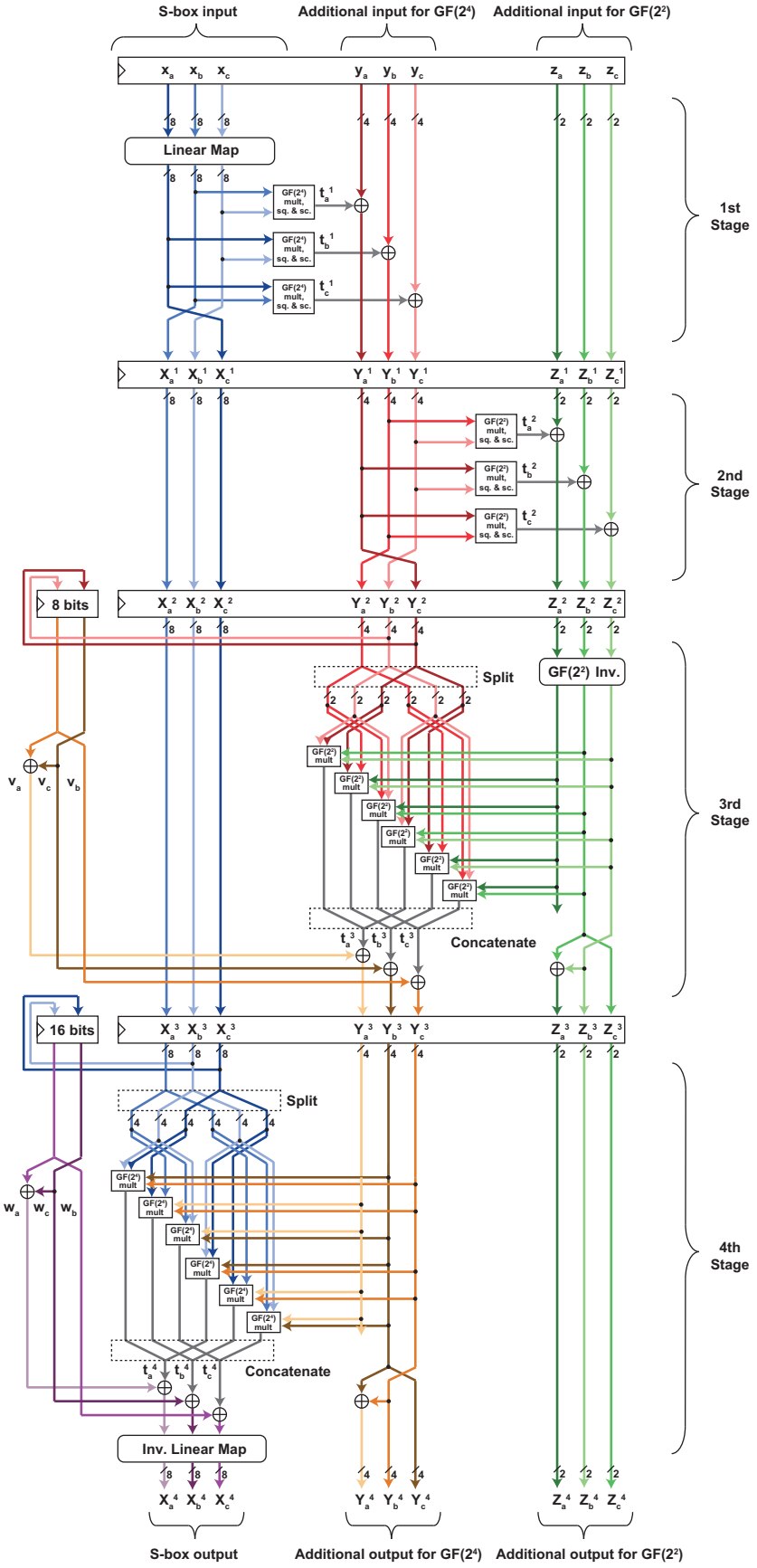

Figure 8: Proposed 3-share TI of the AES S-box. Edges are colored based on Table 3.

Table 3: Data propagation in the proposed S-box implementation. Row: intermediate values, column: inputs. The names of intermediate values and inputs follow Fig. 8. ♢/♠ is placed if the intermediate result depends on the input. ♠ is placed if the intermediate result is masked by the input.

	x_a	x_b	x_c	y_a	y_b	y_c	z_a	z_b	z_c	v_a	v_b	v_c	w_a	w_b	w_c
t_a^1	—	♢	♢	—	—	—	—	—	—	—	—	—	—	—	—
t_b^1	♢	—	♢	—	—	—	—	—	—	—	—	—	—	—	—
t_c^1	♢	♢	—	—	—	—	—	—	—	—	—	—	—	—	—
X_a^1	—	♠	—	—	—	—	—	—	—	—	—	—	—	—	—
X_b^1	—	—	♠	—	—	—	—	—	—	—	—	—	—	—	—
X_c^1	♠	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Y_a^1	—	♢	♢	♠	—	—	—	—	—	—	—	—	—	—	—
Y_b^1	♢	—	♢	—	♠	—	—	—	—	—	—	—	—	—	—
Y_c^1	♢	♢	—	—	—	♠	—	—	—	—	—	—	—	—	—
Z_a^1	—	—	—	—	—	—	♠	—	—	—	—	—	—	—	—
Z_b^1	—	—	—	—	—	—	—	♠	—	—	—	—	—	—	—
Z_c^1	—	—	—	—	—	—	—	—	♠	—	—	—	—	—	—
t_a^2	♢	♢	♢	—	♢	♢	—	—	—	—	—	—	—	—	—
t_b^2	♢	♢	♢	♢	—	♢	—	—	—	—	—	—	—	—	—
t_c^2	♢	♢	♢	♢	♢	—	—	—	—	—	—	—	—	—	—
X_a^2	—	♠	—	—	—	—	—	—	—	—	—	—	—	—	—
X_b^2	—	—	♠	—	—	—	—	—	—	—	—	—	—	—	—
X_c^2	♠	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Y_a^2	♢	—	♢	—	♠	—	—	—	—	—	—	—	—	—	—
Y_b^2	♢	♢	—	—	—	♠	—	—	—	—	—	—	—	—	—
Y_c^2	—	♢	♢	♠	—	—	—	—	—	—	—	—	—	—	—
Z_a^2	♢	♢	♢	—	♢	♢	♠	—	—	—	—	—	—	—	—
Z_b^2	♢	♢	♢	♢	—	♢	—	♠	—	—	—	—	—	—	—
Z_c^2	♢	♢	♢	♢	♢	—	—	—	♠	—	—	—	—	—	—
t_a^3	♢	♢	♢	♢	♢	♢	—	♢	♢	—	—	—	—	—	—
t_b^3	♢	♢	♢	♢	♢	♢	♢	—	♢	—	—	—	—	—	—
t_c^3	♢	♢	♢	♢	♢	♢	♢	♢	—	—	—	—	—	—	—
X_a^3	—	♠	—	—	—	—	—	—	—	—	—	—	—	—	—
X_b^3	—	—	♠	—	—	—	—	—	—	—	—	—	—	—	—
X_c^3	♠	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Y_a^3	♢	♢	♢	♢	♢	♢	—	♢	♢	♠	—	—	—	—	—
Y_b^3	♢	♢	♢	♢	♢	♢	♢	—	♢	—	—	♠	—	—	—
Y_c^3	♢	♢	♢	♢	♢	♢	♢	♢	—	—	♠	—	—	—	—
Z_a^3	—	—	—	—	—	—	♠	—	—	—	—	—	—	—	—
Z_b^3	—	—	—	—	—	—	—	—	♠	—	—	—	—	—	—
Z_c^3	—	—	—	—	—	—	—	♠	—	—	—	—	—	—	—
t_a^4	♢	♢	♢	♢	♢	♢	♢	♢	♢	—	♢	♢	—	—	—
t_b^4	♢	♢	♢	♢	♢	♢	♢	♢	♢	♢	♢	—	—	—	—
t_c^4	♢	♢	♢	♢	♢	♢	♢	♢	♢	♢	—	♢	—	—	—
X_a^4	♢	♢	♢	♢	♢	♢	♢	♢	♢	—	♢	♢	♠	—	—
X_b^4	♢	♢	♢	♢	♢	♢	♢	♢	♢	♢	♢	—	—	—	♠
X_c^4	♢	♢	♢	♢	♢	♢	♢	♢	♢	♢	—	♢	—	♠	—
Y_a^4	—	—	—	—	—	—	—	—	—	♠	—	—	—	—	—
Y_c^4	—	—	—	—	—	—	—	—	—	—	♠	—	—	—	—
Y_b^4	—	—	—	—	—	—	—	—	—	—	—	♠	—	—	—
Z_a^4	—	—	—	—	—	—	♠	—	—	—	—	—	—	—	—
Z_b^4	—	—	—	—	—	—	—	—	♠	—	—	—	—	—	—
Z_c^4	—	—	—	—	—	—	—	♠	—	—	—	—	—	—	

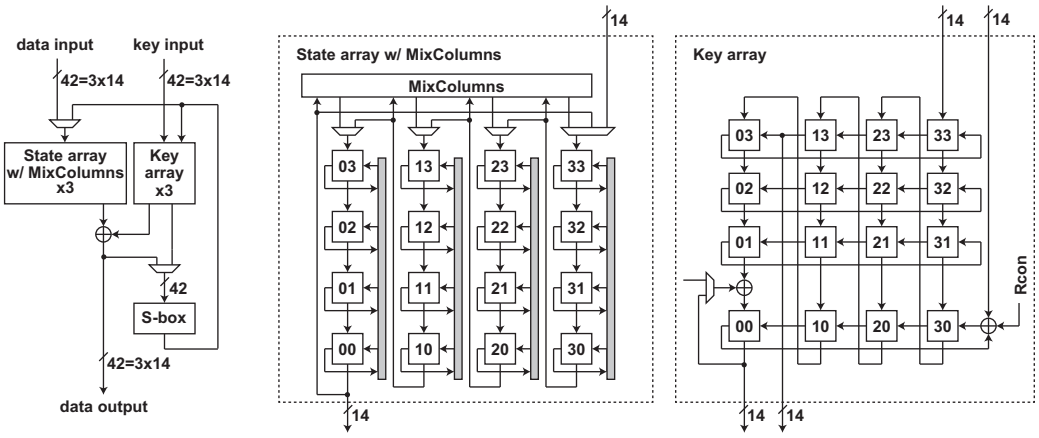

Figure 9: AES circuit using the proposed S-box implementation

Table 3 is visualized by coloring the edges in Fig. 8. In the figure, we colored the edges corresponding to the intermediate value marked with ♠ in Table 3. More specifically, the same color is assigned to the intermediate values having the same mask. Using Fig. 8, we can verify that all the stage inputs/outputs have distinct masks.

4.4 Implementation and Evaluation

4.4.1 Circuit Architecture

Fig. 9 shows the circuit for AES encryption based on the proposed S-box implementation. The design is based on the one by Moradi et al. [MPL+11] that uses the state and key arrays as basic construction blocks. Three arrays are used to store 3-share representations of the AES state and key. Note that three key arrays are used to protect key scheduling as well. As previously discussed, the additional output is forwarded to the next AES round. To carry the additional outputs, the data width of the state and key arrays is extended from 8 to 14 bits. Accordingly, each array has $14 \times 16 = 224$ bits. Thus, $224 \times 6 = 1,344$ bits of registers are needed in total for storing the shared representations of the state and key. The AES circuit works similarly to the original design [MPL+11]. An exception is MixColumns in which only the state elements are processed while the additional inputs are unchanged.

To convert a message into its 3-share representation, we need $128 \times 2 + 6 \times 16 \times 2 = 448$ bits of randomness. To convert a key into its 3-share representation, on the other hand, we need $128 \times 2 + 6 \times 4 \times 2 = 304$ bits of randomness because there are only four S-boxes in key scheduling. Also, 24-bit random number is needed to initialize the temporary registers in between the 2nd/3rd and 3rd/4th stages. As a result, 776 initial random bits are needed for single AES processing.

The key and plaintext are fed to the circuit in the 3-share representation with additional inputs added. The I/O ports have 42-bit width similar to the datapath. Therefore, feeding the key and plaintext requires 16 cycles. The AES round is executed in 25 cycles: 16 cycles for S-boxes, four cycles for MixColumns, one cycle for ShiftRows, and four additional cycles for pipeline latency. Single AES encryption requires 266 cycles in total.

4.4.2 Performance Evaluation and Comparison

Table 4: Circuit-area breakdown of the proposed AES circuit

Module	Unit area [kGE]	# units	Total area [kGE]
Key array	2.1	3	6.2
State array w/ MixColumns	2.3	3	6.8
S-box (including pipeline registers)	3.5	1	3.5
Others	—	—	0.6
Total	—	—	17.1

The design is implemented in HDL and synthesized using the NanGate 45-nm standard cell library [Nan] with Synopsys Design Compiler. Table 1 shows the performance evaluation. Note that the implementations in Table 1 are evaluated using different standard cell libraries. We should consider the difference of the libraries in comparing results.

The proposed S-box circuit uses 3.5 [kGE], which is comparable to that of conventional works. However, the size of the AES circuit is 17.1 [kGE], which is much larger than that of conventional designs. This is explained by a large number of registers. As shown in the circuit-area breakdown in Table 4, the state and key arrays occupy 76% (13.0 [kGE]) of the total circuit area. As discussed earlier, in the proposed design, the data are stored in the 3-share form along with the additional inputs. Consequently, $1,344$-bit registers are required in the state and key arrays. In comparison, the designs [BGN+15, GMK17, CRB+16] store only two shares and thus, $8 \times 16 \times 2 \times 2 = 512$ bits are required. Furthermore, the designs [BGN+15, CRB+16, WM18] use unprotected key scheduling, and thus, only 384 bits are required.

Despite the large circuit area, the proposed design has an advantage over the conventional design by Wegener et al. [WM18]. The latency of the proposed design is 266 cycles, which is smaller than 1/10 that of the conventional design. As Wegener et al. used area–latency trade-off for a compact circuit area, it would be fair to compare these designs in terms of the area–latency product. As shown in Table 1, the cost of the proposed design is smaller than 1/4 that of the conventional design even if the designs are compared in terms of the area–latency product.

The design has a room for further improvement. Since there are only four S-boxes in key scheduling, in the key arrays, $(16-4) \times 6 \times 3 = 216$ bits of the registers are wasted. The waste comes from the restriction to make the design as close as the original one [MPL+11]. By redesigning the key array, we have a room for saving 216 bits or 1.5 [kGE]. Moreover, some of the conventional works use unprotected key scheduling [UHA17, WM18]. Under such a design policy, we can save roughly 5.0 [kGE] by removing the two key arrays and some additional inputs.

4.4.3 Simulation-based Leakage Assessment

The security of the AES circuit is evaluated using a simulation-based leakage assessment. The post-synthesis simulation with back annotation is performed using the Cadence NC-Verilog logic simulator at a precision of 1 ps. The circuit is operated with a clock interval longer than a critical path delay. During the simulation, the number of $0 \to 1$ output transitions in all the standard cells is measured for each cycle. The measured data are used as an approximation of the dynamic current consumption measured at V_{DD} [TV05]. Thus, they are used as a simulated power trace. As switching in each standard cell is considered, the simulation captures glitches.

The test vector leakage assessment (TVLA) [BCD+13] is conducted using the simulated power traces. For each of the fixed and variable test vectors, $100,000$ simulated power traces are obtained. Subsequently, the two sets of traces are compared with T-statistic.

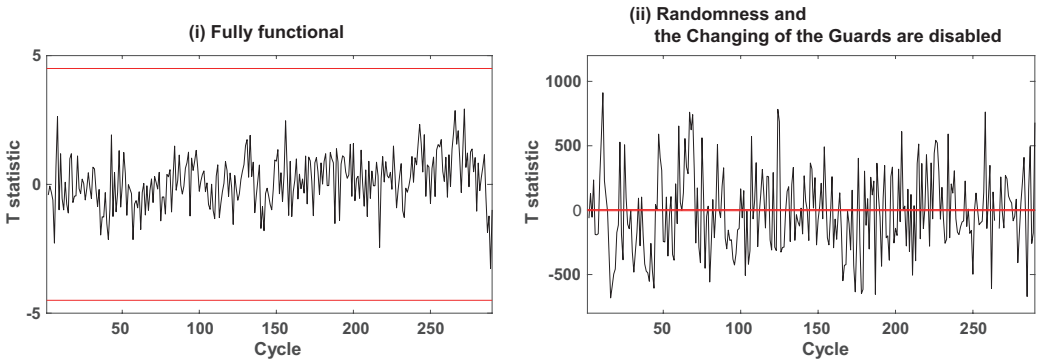

Figure 10: Results of the simulation-based leakage assessment. Horizontal: T-statistics, vertical: cycle. Subgraphs (i) and (ii) correspond to two different operating conditions.

For comparison, the target is operated under two different conditions: (i) fully functional and (ii) randomness is disabled. For the second case, the random numbers for creating a share and additional inputs are all zero. In addition, the two mask registers in the 3rd and 4th stages are set to zero.

Fig. 10 (i) and (ii) show the traces of T-statistics. The horizontal and vertical axes represent the cycle and T-statistics, respectively. In the fully functional case in Fig. 10-(i), the obtained T-statistics fit within the range $[-4.5, 4.5]$. In Fig. 10-(ii), the T-statistics are far above and beyond the borders. The results show that the target AES circuit passes the leakage assessment if it is fully functional.

5 Conclusion

In this paper, we discussed how to construct a uniform sharing of a target mapping having different input and output sizes. We introduced two techniques namely extension and restriction. In extension, a target mapping is transformed in such a way that the extended mapping has a uniform sharing. However, it requires an additional input representing zero i.e., $x_a + x_b + x_c = 0$. In restriction, the additional output obtained as a side effect of extension is transformed into a share representing 0. Subsequently, the zero share is reused as the additional input in the next sharing. By combining extension and restriction, sharing is realized without remasking. The proposed method is a generalization of the *changing of the guards* technique [Dae17]. By applying the above methods to the Canright's AES S-box implementation, the first 3-share TI of the AES S-box without using remasking is obtained.

As shown in Table 1, the proposed AES design was larger than the conventional designs. Optimizing its performance is an important future research direction. As discussed in Sect. 4.4.2, the presented design has a room for further optimization. Moreover, there is a possibility of sharing the additional inputs between consecutive S-box calculations, but the security under such optimizations remains open. Also, evaluating the proposed design with real measurement is an important future research direction.

Acknowledgements

I thank the anonymous reviewers and the shepherd for their valuable comments. I appreciate Rei Ueno for inspiring discussions that motivated this study. A part of the study is supported by JSPS KAKENHI Grant Number 17H06681 and JP18H05289. The

CAD tools used in the paper are supported by VLSI Design and Education Center (VDEC), the University of Tokyo with the collaboration with CADENCE Corporation and SYNOPSYS Corporation.

References

[BBP+16] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 616–648. Springer, 2016.

[BCD+13] G. Becker, J. Cooper, E. DeMulder, G. Goodwill, J. Jaffe, G. Kenworthy, T. Kouzminov, A. Leiserson, M. Marson, P. Rohatgi, and S. Saab. Test vector leakage assessment (TVLA) methodology in practice. In *International Cryptographic Module Conference*, 2013.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

[BGN+15] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Trade-offs for threshold implementations illustrated on AES. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 34(7):1188–1200, 2015.

[Can05] David Canright. A very compact S-Box for AES. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 441–455. Springer, 2005.

[CRB+16] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with d+1 shares in hardware. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 194–212. Springer, 2016.

[Dae17] Joan Daemen. Changing of the guards: A simple and efficient method for achieving uniformity in threshold sharing. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 137–153. Springer, 2017.

[FGP+18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence

of physical defaults & the robust probing model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.

[FPS17] Sebastian Faust, Clara Paglialonga, and Tobias Schneider. Amortizing randomness complexity in private circuits. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 781–810. Springer, 2017.

[GMK17] Hannes Groß, Stefan Mangard, and Thomas Korak. An efficient side-channel protected AES implementation with arbitrary protection order. In Helena Handschuh, editor, *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, volume 10159 of *Lecture Notes in Computer Science*, pages 95–112. Springer, 2017.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[MPL+11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the limits: A very compact and a threshold implementation of AES. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 2011.

[Nan] NanGate. NanGate FreePDK45 open cell library. http://www.nangate.com.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.

[NRS11] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptology*, 24(2):292–321, 2011.

[Pap18] Kostas Papagiannopoulos. Low randomness masking and shuffling: An evaluation using mutual information. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):524–546, 2018.

[RSWO18] Eyal Ronen, Adi Shamir, Achi-Or Weingarten, and Colin O'Flynn. IoT goes nuclear: Creating a Zigbee chain reaction. *IEEE Security & Privacy*, 16(1):54–62, 2018.

[SK96] Bruce Schneier and John Kelsey. Unbalanced feistel networks and block cipher design. In Dieter Gollmann, editor, *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *Lecture Notes in Computer Science*, pages 121–144. Springer, 1996.

[Tah16] Saleem Mohammed Ridha Taha. *Fundamentals of Reversible Logic*, pages 7–16. Springer International Publishing, Cham, 2016.

[TV05] Kris Tiri and Ingrid Verbauwhede. Simulation models for side-channel information leaks. In William H. Joyner Jr., Grant Martin, and Andrew B. Kahng, editors, *Proceedings of the 42nd Design Automation Conference, DAC 2005, San Diego, CA, USA, June 13-17, 2005*, pages 228–233. ACM, 2005.

[UHA17] Rei Ueno, Naofumi Homma, and Takafumi Aoki. Toward more efficient dpa-resistant AES hardware architecture based on threshold implementation. In Sylvain Guilley, editor, *Constructive Side-Channel Analysis and Secure Design - 8th International Workshop, COSADE 2017, Paris, France, April 13-14, 2017, Revised Selected Papers*, volume 10348 of *Lecture Notes in Computer Science*, pages 50–64. Springer, 2017.

[WM18] Felix Wegener and Amir Moradi. A first-order SCA resistant AES without fresh randomness. In Junfeng Fan and Benedikt Gierlichs, editors, *Constructive Side-Channel Analysis and Secure Design - 9th International Workshop, COSADE 2018, Singapore, April 23-24, 2018, Proceedings*, volume 10815 of *Lecture Notes in Computer Science*, pages 245–262. Springer, 2018.

Appendix

Proof of Lemma 1

Proof. The sharing $\{\psi_a^E, \psi_b^E, \psi_c^E\}$ is correct because

$$X_a + X_b + X_c = x, \qquad Y_a + Y_b + Y_c = \psi(x) + y.$$

The sharing is non-complete because ψ_a^E, ψ_b^E, and ψ_c^E are independent of $[x_a, \tilde{y}_a]$, $[x_b, \tilde{y}_b]$, and $[x_c, \tilde{y}_c]$, respectively. The sharing is invertible because

$$\begin{aligned} x_a &= X_c, & y_a &= \psi_a(\tilde{X}_c) + Y_a, \\ x_b &= X_a, & y_b &= \psi_b(\tilde{X}_a) + Y_b, \\ x_c &= X_b, & y_c &= \psi_c(\tilde{X}_b) + Y_c, \end{aligned}$$

and thus, it is uniform. □

Proof of Lemma 2

Proof. The sharing $\{\psi_a^\perp, \psi_b^\perp, \psi_c^\perp\}$ is correct because

$$X_a + X_b + X_c = 0, \qquad Y_a + Y_b + Y_c = y.$$

The sharing is non-complete because $\psi_a^\perp$, $\psi_b^\perp$, and $\psi_c^\perp$ are independent of $[x_c, \tilde{y}_a]$, $[x_a, \tilde{y}_b]$, and $[x_b, \tilde{y}_c]$, respectively.

For uniformity, distributions of input and output shares are considered. Let $P_I(x, y)$ and $P_O(X, Y)$ be the probabilistic distributions of the original inputs and outputs, respectively.

Similarly, the distributions of the shared inputs and outputs are denoted as $\overline{P_I}(\overline{x}, \overline{y})$ and $\overline{P_O}(\overline{X}, \overline{Y})$, respectively. As the input is distributed uniformly, we have

$$\overline{P_I}(\overline{x}, \overline{y}) = \frac{P_I(x, y)}{\alpha} = \frac{P_I(x_a + x_b + x_c, y_a + y_b + y_c)}{\alpha}. \qquad \because Eq.\ (2) \qquad (14)$$

First, we consider the case $X \neq 0$ i.e., $X_a + X_b + X_c \neq 0$. These outputs are prohibited by construction and thus,

$$P_O(X, Y) = \overline{P_O}(\overline{X}, \overline{Y}) = 0 \qquad (15)$$

The remaining case $X = 0$ i.e., $X_a + X_b + X_c = 0$ is considered. In this case,

$$P_O(X, Y) = \sum_{t \in \mathcal{X}} P_I(t, Y). \qquad (16)$$

where $\mathcal{X}$ is a set of possible values that X can take. The probability of observing an output share $[X_a, X_b, X_c]$ is solely determined by $[x_a, x_b] = [X_c, X_b]$. Therefore, we observe that

$$\overline{P_O}(\overline{X}, \overline{Y}) = \overline{P_O}([X_a, X_b, X_c], \overline{Y}) = \sum_{t \in \mathcal{X}} \overline{P_I}([X_c, X_b, t], \overline{Y}). \qquad \because Definition(6) \qquad (17)$$

Therefore,

$$\begin{aligned}
\overline{P_O}(\overline{X}, \overline{Y}) &= \sum_{t \in \mathcal{X}} \overline{P_I}([X_c, X_b, t], \overline{Y}) && \because Eq.\ (17) \\
&= \frac{1}{\alpha} \sum_{t \in \mathcal{X}} P_I(X_c + X_b + t, Y) && \because Eq.\ (14) \\
&= \frac{1}{\alpha} \sum_{t \in \mathcal{X}} P_I(t, Y) \\
&= \frac{1}{\alpha} P_O(X, Y). && \because \text{Eq. } (16) \qquad (18)
\end{aligned}$$

From Eq. (15) and (18), we obtain $\overline{P_O}(\overline{X}, \overline{Y}) = \frac{P_O(X,Y)}{\alpha}$ for all the cases. Therefore, the sharing is uniform according to Eq. (3). □

Proof of Theorem 1

Proof. The sharing $\{\psi_a^R, \psi_b^R, \psi_c^R\}$ is correct because

$$X_a + X_b + X_c = 0, \qquad Y_a + Y_b + Y_c = \psi(x) + y.$$

The sharing is non-complete because ψ_a^R, ψ_b^R, and ψ_c^R are independent of $[x_a, \tilde{y}_a]$, $[x_b, \tilde{y}_b]$, and $[x_c, \tilde{y}_c]$, respectively. $\{\psi_a^E, \psi_b^E, \psi_c^E\}$ and $\{\psi_a^\perp, \psi_b^\perp, \psi_c^\perp\}$ are uniform according to Lemma 1 and 2, respectively. Therefore, $\{\psi_a^R, \psi_b^R, \psi_c^R\}$ being a composition of $\{\psi_a^E, \psi_b^E, \psi_c^E\}$, and $\{\psi_a^\perp, \psi_b^\perp, \psi_c^\perp\}$ is uniform. □

Proof of Corollary 1

Proof. As the sharing $\{f_a^E, f_b^E, f_c^E\}$ is a sharing as specified in Definition 3, it is a correct, non-complete, and uniform sharing according to Lemma 1. □

Proof of Corollary 2

Proof. The sharing is correct because

$$X_a^i + X_b^i + X_c^i = g(x^i, z^i), \qquad Y_a^i + Y_b^i + Y_c^i = g(y^i, z^i), \qquad Z_a^i + Z_b^i + Z_c^i = 0$$

for $i \in [1, L]$. The sharing is non-complete because, for any $i, j \in [1, L]$, $\{X_a^i, Y_a^i, Z_a^i\}$, $\{X_b^i, Y_b^i, Z_b^i\}$, and $\{X_c^i, Y_c^i, Z_c^i\}$ are independent of $\{x_a^j, y_a^j, z_a^j\}$, $\{x_b^j, y_b^j, z_b^j\}$, and $\{x_c^j, y_c^j, z_c^j\}$, respectively. The sharing is uniform through construction. □