Changing of the Guards: A Simple and Efficient Method for Achieving Uniformity in Threshold Sharing

Joan Daemen[1,2(✉)]

[1] Radboud University, Nijmegen, The Netherlands
J.Daemen@science.ru.nl
[2] STMicroelectronics, Diegem, Belgium

Abstract. Since they were first proposed as a countermeasure against differential power analysis (DPA) and differential electromagnetic analysis (DEMA) in 2006, threshold schemes have attracted a lot of attention from the community concentrating on cryptographic implementations. What makes threshold schemes so attractive from an academic point of view is that they come with an information-theoretic proof of resistance against a specific subset of side-channel attacks: first-order DPA. From an industrial point of view they are attractive as a careful threshold implementation forces adversaries to DPA of higher order, with all its problems such as noise amplification. A threshold scheme that offers the mentioned provable security must exhibit three properties: correctness, incompleteness and uniformity. A threshold scheme becomes more expensive with the number of shares that must be implemented and the required number of shares is lower bound by the algebraic degree of the function being shared plus 1. Defining a correct and incomplete sharing of a function of degree d in $d+1$ shares is straightforward. However, up to now there is no generic method to achieve uniformity and finding uniform sharings of degree-d functions with $d+1$ shares has been an active research area. In this paper we present a generic, simple and potentially cheap method to find a correct, incomplete and uniform $d+1$-share threshold scheme of any S-box layer consisting of degree-d invertible S-boxes. The uniformity is not implemented in the sharings of the individual S-boxes but rather at the S-box layer level by the use of feedforward and some expansion of shares. When applied to the Keccak-p nonlinear step χ, its cost is very small.

Keywords: Side-channel attacks · Threshold schemes · Uniformity · Keccak

1 Introduction

Systems such as digital rights management (DRM) or banking cards try to offer protection against adversaries that have physical access to platforms performing cryptographic computations, allowing them to measure computation time, power

W. Fischer and N. Homma (Eds.): CHES 2017, LNCS 10529, pp. 137–153, 2017.
DOI: 10.1007/978-3-319-66787-4_7

consumption or electromagnetic radiation. Adversaries can use this side channel information to retrieve cryptographic keys. A particularly powerful attack against implementations of cryptographic algorithms is differential power analysis (DPA) introduced by Kocher et al. [20]. This attack can exploit even the weakest dependence of the power consumption (or electromagnetic radiation) on the value of the manipulated data by combining the measurements of many computations to improve the signal-to-noise ratio. The simplest form of DPA is first-order DPA, that exploits the correlation between the data and the power consumption. To make side channel attacks impractical, system builders implement countermeasures, often multiple at the same time.

In threshold schemes, as proposed by Rijmen et al. [23–25] one represents each sensitive variable by a number of shares (typically denoted by $d + 1$) such that their (usually) bitwise sum equals that variable. These shares are initially generated in such a way that any subset of d shares gives no information about the sensitive variable. Functions (S-boxes, mixing layers, round functions ...) are computed on the shares of the inputs resulting in the output as a number of shares. Threshold schemes must be *correct*: the sum of the output shares equals the result of applying the implemented function on the sum of the input shares. Another essential property of a threshold implementation of a function is *incompleteness*: each output share shall be computed from at most d input shares, or equivalently, in the computation of each output share at least one input share is not used. Incompleteness guarantees that each individual output share computation cannot leak information about sensitive variables. The resulting output is then typically subject to some further computation, again in the form of separate and incomplete computation on shares. For these subsequent computations to not leak information about the sensitive variables, the output of the previous stage must still be uniform. Therefore, in an iterative cryptographic primitive such as a block cipher, we need a threshold implementation of the round function that yields a uniformly shared output if its input is uniformly shared. This property of the threshold implementation is called *uniformity*.

Threshold schemes form a good protection mechanism against DPA. In particular, using it allows building cryptographic hardware that is guaranteed to be unattackable with first-order DPA, assuming certain leakage models of the cryptographic hardware at hand and for a plausible definition of "first order". De Cnudde et al. have an interesting work [13] on such assumptions and their validity in the real world. Still, threshold schemes remain a very attractive technique for building cipher implementations that offer a high level of resistance against DPA and differential electromagnetic analysis (DEMA).

Constructing an incomplete threshold implementation of a non-linear function is rather straightforward and can be done in the following way. One can express the function algebraically as the sum of monomials. Then one replaces each shared variable by the sum of its shares. Subsequently, one can work out the expressions resulting in a larger number of monomials, where the factors are bits (or in general, components) of the shares. A monomial of degree d can have factors from at most d shares. So if there are $d + 1$ shares, such a monomial

is incomplete: there is at least one share missing. It follows that to build an incomplete sharing of a function of algebraic degree d, it suffices to take $d+1$ shares. Clearly, the implementation cost of a function increases exponentially with its degree: a monomial of degree d requires $d+1$ shares and explodes into the sum of $(d+1)^d$ monomials. To reduce the implementation cost, Stoffelen applies techniques for representing S-boxes with minimum number of nonlinear operations [31]. Kutzner et al. on the other hand factor S-boxes of some degree as the composition of functions of lower algebraic degree [21]. Such techniques, combined with tower field representation, are also applied in the sharing of the AES S-box, that natively has algebraic degree 7. We refer again to De Cnudde et al. for an example [14]. These publications demonstrate that these techniques are quite powerful, but serial composition comes at a prize. It requires the insertion of registers (or latches) between the combinatorial circuits that increase latency.

Constructing a correct, incomplete and uniform sharing is widely perceived as a challenge and an important research problem. Several publications have been devoted to the classification of 3, 4 and 5-bit S-boxes with respect to cryptographic properties, and the minimum number of shares for which a uniform sharing is known is an important criterion. Examples include the study of Bilgin et al. [8] and that of Božilov et al. [10]. Other papers propose solutions, sometimes only partial, for large classes of S-boxes. We refer again to Bilgin et al. [9], Kutzner et al. [21], and Beyne et al. [5]. A well-known example of an S-box that is problematic in this context is the Keccak S-box, known as χ. It has algebraic degree 2 and no uniform incomplete 3-share threshold implementations is known. We proposed a number of different solutions with varying degrees of efficiency in [6]. One solution is the transition from 3 to 4 or even 5 shares. Another is the compensation of loss of uniformity by injecting fresh randomness. As argued by Reparaz et al. [29], this technique brings the threshold scheme in the realm of private circuits as proposed by Ishai et al. [19].

Given a non-uniform threshold implementation, it is not immediate how to exploit its non-uniformity in an attack. We made a start in explorations in that direction in [16,17]. However, uniformity of a threshold implementation is essential in its information-theoretical proof of resistance against first-order DPA. In short, if one has a uniform sharing, one does not have to give additional arguments why the threshold scheme would be secure against first-order DPA.

In this paper we present a simple and efficient technique for building a threshold implementation with $d+1$ shares of any invertible S-box layer of degree d that is correct, incomplete and uniform. When applied to the nonlinear layer in Keccak, χ, it can be seen as the next logical step of the methods discussed in Sect. 3 of our paper [6]. In that method 4 fresh random bits must be introduced every round to restore uniformity. The added value of the technique in this paper is that it no longer needs any fresh randomness and that it can convert a correct and incomplete sharing of any S-box into a correct, incomplete and uniform sharing of a layer of such S-boxes.

1.1 The "Changing of the Guards" Idea in a Nutshell

The basic method can be summarized as follows:

- The shared S-boxes are arranged in a linear array. These sharings must be correct and incomplete.
- Each share at the output of S-box i is made uniform by bitwise adding to it one or two shares from the input of S-box $i-1$.
- The state is augmented with d dummy components, called *guards*, to be added to the output of the first S-box in the array.
- The new value of the guards are taken from the input of the last S-box in the array.
- Uniformity is proven by giving an algorithm that computes the shared input from the shared output of this mapping.

For threshold sharings that have a so-called multi-transformation property, the guards can be reduced in size and so does the amount of bits fed forward.

1.2 Notation

Assume we have a nonlinear mapping that consists of a layer of invertible S-boxes. We denote the width of the S-boxes by n and their total number by m. So the layer operates on an array of $n \times m$ bits. We denote the input as $x = (x_1, x_2, x_3, \ldots x_m)$ and the output as $X = (X_1, X_2, X_3, \ldots X_m)$, with each of the x_i and X_i an n-bit array.

In general the S-boxes can differ per position. We denote the S-box at position i by S_i, so $X_i = S_i(x_i)$.

We denote addition in GF(2) by $+$.

1.3 Overview of the Paper

In Sect. 2 we explain and prove the soundness of the method applied to the simplest possible case. In Sect. 3 we formulate the method for a more general case and in Sect. 4 we apply it to the nonlinear layer used in Keccak, Keyak and Ketje. Finally in Sect. 5 we discuss some implementation aspects.

2 The Basic Method Applied to 3-Share Threshold Schemes

Assume the same S-box is used for all positions and its algebraic degree is 2 over GF(2), that we denote by S. In that case it is trivial to find a correct and incomplete threshold scheme with 3 shares S by substituting the terms in the algebraic expression of the S-box by their sum as components and appropriately distributing the monomials over the three shares of the S-box sharing. We denote the three shares that represent x_i by a_i, b_i and c_i, with $x_i = a_i + b_i + c_i$. Likewise, we denote the three shares that represent X_i by A_i, B_i and C_i, with

$X_i = A_i + B_i + C_i$. The sharing of S consists of three functions from $2n$ to n bits, that we denote as (S_a, S_b, S_c). Correctness is satisfied if:

$$S_a(b_i, c_i) + S_b(a_i, c_i) + S_c(a_i, b_i) = S(a_i + b_i + c_i)\,.$$

Incompleteness is implied by the fact that each of the three elements of (S_a, S_b, S_c) take only two shares as inputs. In this scheme our m-component input x is represented by triplet (a, b, c) with three shares.

At the basis of our "Changing of the Guards" technique for achieving uniformity is the expansion of the shared representation. In particular, for the input we expand share b with an additional dummy component that we denote as b_0 and do the same for c. In this sharing x is represented by (a, b, c) where a has m components and both b and c have $m + 1$ components. A triplet (a, b, c) is a uniform sharing of x if all possible values (a, b, c) compliant with x are equiprobable. As there are $2^{(3m+2)n}$ possible triplets (a, b, c) and being compliant with x requires the satisfaction of mn independent linear binary equations, there are exactly $2^{(3m+2)n-mn} = 2^{2(m+1)n}$ encodings (a, b, c) of any particular value x. The same holds for the sharing (A, B, C) of the output X.

Definition 1. *The* Changing of the Guards *sharing of an S-box layer where* (S_a, S_b, S_c) *is a sharing of* S*, mapping* (a, b, c) *to* (A, B, C)*, is given by:*

$$\begin{aligned}
A_i &= S_a(b_i, c_i) + b_{i-1} + c_{i-1} && for\ i > 0\\
B_i &= S_b(a_i, c_i) + c_{i-1} && for\ i > 0\\
C_i &= S_c(a_i, b_i) + b_{i-1} && for\ i > 0\\
B_0 &= c_m\\
C_0 &= b_m
\end{aligned}$$

The sharing is depicted in Fig. 1.

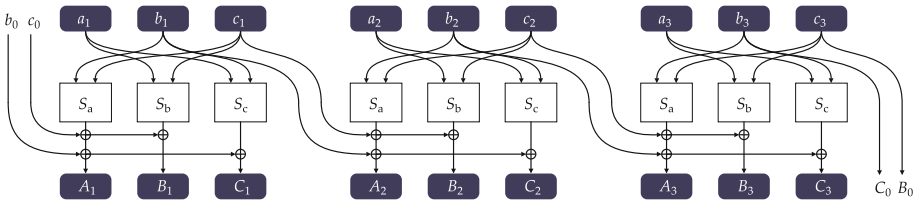

Fig. 1. Changing of the Guards sharing applied to simple S-box layer.

We can now prove the following theorem.

Theorem 1. *If* S *is an invertible S-box and* (S_a, S_b, S_c) *is a correct and incomplete sharing of* S*, the sharing of Definition 1 is a correct, incomplete and uniform sharing of an S-box layer with* S *as component.*

Proof. Correctness follows from the correctness of the S-box sharing and the fact that each input components that is fed forward to the output of its neigboring components is added twice. We have for all $i > 0$:

$$\begin{aligned} A_i + B_i + C_i &= S_a(b_i, c_i) + b_{i-1} + c_{i-1} + S_b(a_i, c_i) + c_{i-1} + S_c(a_i, b_i) + b_{i-1} \\ &= S_a(b_i, c_i) + S_b(a_i, c_i) + S_c(a_i, b_i) \\ &= S(a_i + b_i + c_i)\,. \end{aligned}$$

For incompleteness, we see in Definition 1 that the computation of A_i does not involve components of a, the one of B_i does not involve components of b and the one of C_i does not involve components of c. Note that making this statement valid for component $i = 0$ necessitates the *swap* when expressing the output guards from the input shares of the last S-box: $(C_0, B_0) = (b_m, c_m)$.

For uniformity, we observe that for each input x or each output X there are exactly $2^{2(m+1)n}$ valid sharings. If the mapping of Definition 1 is an invertible mapping from (a, b, c) to (A, B, C), it implies that if (a, b, c) is a uniform sharing of x, then (A, B, C) is a uniform sharing of X. It is therefore sufficient to show that the mapping of Definition 1 is invertible. We will do that by giving a method to compute (a, b, c) from (A, B, C).

We compute the components of (a, b, c) starting from index m down to 0. First we compute the shares b_m and c_m from the output guards. We have $b_m = C_0$ and $c_m = B_0$. Then we use the correctness property to compute the component x_m from $X_m = A_m + B_m + C_m$ by applying the inverse S-box yielding $a_m = x_m + b_m + c_m$. From this we compute the output components of the S-box. This allows us again to compute b_{m-1} and c_{m-1}. Concretely, we can iterate the following loop for i going from m down to 1:

$$\begin{aligned} a_i &= S^{-1}(A_i + B_i + C_i) + b_i + c_i \\ b_{i-1} &= S_c(a_i, b_i) + C_i \\ c_{i-1} &= S_b(a_i, c_i) + B_i\,. \end{aligned}$$

□

The term "guards" refers to the dummy components b_0 and c_0 that are there to guard uniformity and that are "changed" to B_0 and C_0 by the shared implementation of the S-box layer.

The cost of this method is the addition of 4 XOR gates per bit of x and the expansion of the representation by $2n$ bits. The cost of additional XOR gates is typically not negligible but still relatively modest compared to the gates in the S-box sharing. For a typical S-box layer the expansion of the state is very small.

When applying this method to an iterated cipher that has a round function consisting of an S-box layer and a linear layer, one can do the following. The sharing of the S-box layer maps (a, b, c) to (A, B, C) and the linear layer is applied to the shares separately. In the linear mapping the guard components B_0 and C_0 are simply mapped to the components b_0 and c_0 of the next round by the identity.

It is likely that the *swapping* that takes place between the guards is not necessary, but it does simplify the proof for the incompleteness aspect.

3 Generalization to Any Invertible S-box Layer

Here we give a method for an S-box layer with only restriction that the component S-boxes have the same width and are all invertible. So this includes the case that the S-boxes are different and even the case that they have different algebraic degrees. We assume the maximum degree over all S-boxes of the layer is d and so we can produce a correct and incomplete threshold scheme with $d+1$ shares. We denote the shares by x^0 to x^d and component j of share i by x_i^j.

In the generalization there are d guard components instead of two. Similarly to the three-share implementation, there is no guard for the first share (a or x^0). The schedule for adding shares from the neighboring S-box is somewhat more complicated. There are four cases, depending on the index j of the output share considered:

$j > 2$: add input shares $j-1$ and $j-2$ of its neighboring S-box;
$j = 2$: add input share 1 of its neighboring S-box;
$j = 1$: add input share d of its neighboring S-box;
$j = 0$: add input shares d and $d-1$ of its neighboring S-box.

Clearly, input shares with index 0 are not added to the neighboring S-box output share. All other input shares are added to exactly two output shares of the neighboring S-box. We depict the treatment of the output of shared S-box of index i for a threshold scheme with 6 shares in Fig. 2. The S-box inputs have been omitted for not crowding the picture. We now provide the more formal definition.

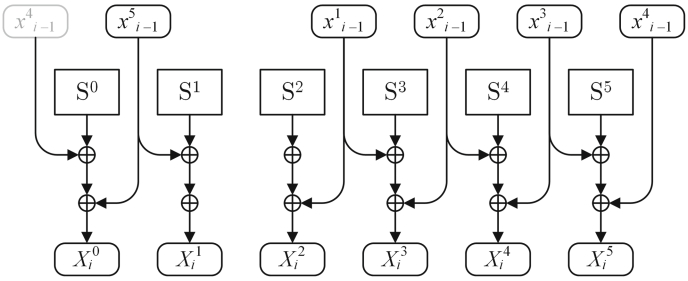

Fig. 2. Example of the generic method, depicting treatment of output of shared S-box i.

Definition 2. *The* Changing of the Guards *sharing of an S-box layer with* $(S_i^0, S_i^1, \ldots, S_i^d)$ *a sharing of* S_i*, mapping* $(x^0, x^1, x^2, \ldots, x^d)$ *to* $(X^0, X^1, X^2, \ldots, X^d)$*, is given by (with* $x_i \setminus x_i^j$ *denoting the vector of* d *elements* $x_i^{j'}$ *for* $j' \neq j$*):*

$$
\begin{aligned}
X_i^0 &= S_i^0(x_i \setminus x_i^0) + x_{i-1}^{d-1} + x_{i-1}^d && \text{for } i > 0 \\
X_i^1 &= S_i^1(x_i \setminus x_i^1) + x_{i-1}^d && \text{for } i > 0 \\
X_i^2 &= S_i^2(x_i \setminus x_i^2) + x_{i-1}^1 && \text{for } i > 0 \\
&\ldots
\end{aligned}
$$

$$
\begin{aligned}
X_i^j &= S_i^j(x_i \setminus x_i^j) + x_{i-1}^{j-2} + x_{i-1}^{j-1} && for\ i > 0 \\
&\cdots \\
X_i^d &= S_i^d(x_i \setminus x_i^d) + x_{i-1}^{d-2} + x_{i-1}^{d-1} && for\ i > 0 \\
X_0^j &= x_m^{j+1} && for\ j > 0 \\
X_0^d &= x_m^1
\end{aligned}
$$

We can now prove the following theorem.

Theorem 2. *Let $\mathcal{S}$ be an S-box layer consisting of invertible n-bit S-boxes S_i with $1 \leq i \leq m$, where the S-boxes S_i may be different and where d is the maximum degree over all these S-boxes. Let $(S_i^0, S_i^1, \ldots S_i^d)$ with $1 \leq i \leq m$ be correct and incomplete sharing of S_i with $d+1$ shares. Then the sharing of Definition 2 is a correct, incomplete and uniform sharing of the S-box layer with S_i as components.*

Proof. Correctness follows from the correctness of the individual S-box sharings and the fact that each input component that is fed forward to the output of its neigboring components is added twice. We have for all $i > 0$:

$$
\begin{aligned}
\sum_j X_i^j &= \left(\sum_j S_i^j(x_i \setminus x_i^j)\right) + \left(\sum_{j>2} x_{i-1}^{j-2} + x_{i-1}^{j-1}\right) + \left(x_{i-1}^{d-1} + x_{i-1}^{d}\right) + \left(x_{i-1}^{d}\right) + \left(x_{i-1}^{1}\right) \\
&= S_i(x_i) + \left(\sum_{0<j<d-1} x_{i-1}^j\right) + \left(\sum_{1<j<d} x_{i-1}^j\right) + x_{i-1}^{d-1} + x_{i-1}^1 \\
&= S_i(x_i)\,.
\end{aligned}
$$

For incompleteness, we see in Definition 2 that the computation of X_i^j does not involve components of x^j as in the S-box input x_i^j is excluded and the inputs of the neighboring S-box that are added are taken from shares $j-1$ and $j-2$ (modulo $d+1$ for $j < 2$). Moreover, there is a (cyclic) swap taking place in the mapping from the input shares of the last S-box to the output guards to ensure this.

For uniformity, it is again sufficient to show that the mapping of Definition 2 is invertible. We give a method to compute $(x^0, x^1, x^2, \ldots, x^d)$ from $(X^0, X^1, X^2, \ldots, X^d)$.

We compute the components of $(x^0, x^1, x^2, \ldots, x^d)$ starting from index m down to 0. From Definition 2 it is immediate that $x_m^1 = X_0^d$ and then $x_m^2 = X_0^1, x_m^3 = X_0^2, x_m^4 = X_0^3, \ldots x_m^d = X_0^{d-1}$. Then we can iterate the following loop for i going from m down to 1:

$$
\begin{aligned}
x_i^0 &= S_i^{-1}(\sum_j X_i^j) + \sum_{j>0} x_i^j \\
x_{i-1}^d &= S_i^1(x_i \setminus x_i^j) + X_i^1 \\
x_{i-1}^1 &= S_i^2(x_i \setminus x_i^2) + X_i^2
\end{aligned}
$$

$$
\begin{aligned}
x_{i-1}^2 &= S_i^3(x_i \setminus x_i^3) + X_i^3 + x_{i-1}^1 \\
x_{i-1}^3 &= S_i^4(x_i \setminus x_i^4) + X_i^4 + x_{i-1}^2 \\
&\dots \\
x_{i-1}^{d-1} &= S_i^d(x_i \setminus x_i^j) + X_i^d + x_{i-1}^{d-2}\,.
\end{aligned}
$$

□

As said, our method applies also to heterogeneous S-box layers, i.e., S-box layers with different S-boxes. Such layers are quite rare in modern cryptography, especially after the benefits of symmetry became clear. The block cipher DES [26] is a notable exception to this, but one may argue whether that is a modern cipher. In any case, one may ask how the method applies to S-box layer in the DES F-function as it consists of non-invertible S-boxes. Remarkably, as was stated by Boss et al. [11] and mathematically explained by the same team [12], in Feistel networks where the S-box layer is embedded in a function whose output is (bitwise) added to part of the state, uniformity is achieved automatically. Basically, thanks to the Feistel construction the shared round function is a permutation and hence uniform. So, if the algebraic degree of the S-box layer is d, it is sufficient to represent the state by $d+1$ shares and have a threshold implementation for the S-boxes that is correct and incomplete.

4 Application to the Sharing χ' for Keccak

Keccak-p is the permutation underlying our hash function Keccak [2,28], our authenticated encryption schemes Keyak [4] and Ketje [3] and is defined in the Keccak reference [2] and NIST standard [28].

4.1 The Sharing χ' of the Nonlinear Layer in Keccak

The nonlinear layer in Keccak-p is called χ. It has algebraic degree 2 over GF(2) and operates independently on 5-bit rows. If we denote the elements of a row by x^0 to x^4, the mapping χ applied to a single row is defined as (with addition and multiplication over GF(2) and indices $\ell \in \mathbb{Z}_5$ taken modulo 5):

$$X^\ell = x^\ell + (x^{\ell+1} + 1)x^{\ell+2}\,.$$

Note that the state of Keccak-p is a three-dimensional array and we only represent the intra-row index here by ℓ for clarity as we look here at a single row.

In [1] we proposed a correct and incomplete sharing of χ with 3 shares and called it χ'. As she mapping χ operates independently on 5-bit rows and consequently χ' operates in parallel on 15-bit units. If we denote the three shares by a, b and c, χ' is defined as:

$$A^\ell = b^\ell + (b^{\ell+1} + 1)b^{\ell+2} + b^{\ell+1}c^{\ell+2} + b^{\ell+2}c^{\ell+1}, \tag{1}$$

$$B^\ell = c^\ell + (c^{\ell+1} + 1)c^{\ell+2} + c^{\ell+1}a^{\ell+2} + c^{\ell+2}a^{\ell+1}, \tag{2}$$

$$C^\ell = a^\ell + (a^{\ell+1} + 1)a^{\ell+2} + a^{\ell+1}b^{\ell+2} + a^{\ell+2}b^{\ell+1}, \tag{3}$$

4.2 The Multi-transformation Property

The mapping χ' has a remarkable property that we can exploit to reduce the overhead due to the "Changing of the Guards" method. We call this a *multi-transformation property*, inspired by the concept of multi-permutations proposed by Schnorr and Vaudenay [30]. Loosely speaking, an n-bit transformation has a multi-transformation property of order r if for any input, the bits in r specific positions in the input and the bits in $n-r$ specific positions in the output, with $r < n$, together fully determine the remaining $n-r$ bits of the input. We now give a more rigorous definition.

Definition 3 (Transformation property with respect to an index subset). *Let f be a transformation operating on vectors of n bits $(x_0, x_1, \ldots, x_{n-1})$ and let us denote the bits of $f(x_0, x_1, \ldots, x_{n-1})$ by $(x_n, x_{n+1}, \ldots, x_{2n-1})$. We can now represent f by a set $\mathcal{F}$ of 2^n vectors of the form $(x_0, x_1, \ldots, x_{2n-1})$. Let S be a subset with n elements of the set of indices of these vectors, i.e., $S \subset \mathbb{Z}_{2n}$. Then we say f* has the transformation property with respect to *S if the set $\mathcal{F}$ has no two elements that are equal in all components in S, or equivalently:*

$$\forall x \in \mathcal{F} : \#\{y \mid \forall i \in S : x_i = y_i\} = 1.$$

We call $r = \#(S \cap \mathbb{Z}_n)$ the order of the transformation property.

Clearly, any n-bit transformation f has a transformation property of order n with respect to $S = \mathbb{Z}_n$. So if it has the transformation property with respect to an additional set S, we call it a multi-transformation. Note that any permutation f has a transformation property of order 0 with respect $S = \mathbb{Z}_{2n} \setminus \mathbb{Z}_n$. In the context of this paper we are interested in finding a multi-transformation property in S-box threshold implementations that are not uniform and hence are not permutations.

4.3 Using the Multi-transformation Property of χ'

The mapping χ' restricted to a single row is a transformation operating on 15 bits. We can show it has a transformation property of order 6. The consequence of this is that we can reduce the size of the guards from 10 bits to 4 bits and the number of bitwise addition operations per row to 8.

We first need to introduce some notation. For a 5-bit vector s, let $\mathrm{L}(s) \triangleq (s^0, s^1, s^2)$ and $\mathrm{R}(s) \triangleq (s^3, s^4)$. Similarly, we define $\mathrm{L}(a,b,c) \triangleq (a^0, b^0, c^0, a^1, b^1, c^1, a^2, b^2, c^2)$ and $\mathrm{R}(a,b,c) \triangleq (a^3, b^3, c^3, a^4, b^4, c^4)$.

Lemma 1. *For any of the 2^{15} choices of $\mathrm{L}(A,B,C), \mathrm{R}(a,b,c)$, there is exactly one solution $\mathrm{L}(a,b,c), \mathrm{R}(A,B,C)$ such that $(A,B,C) = \chi'(a,b,c)$.*

Proof. We describe how to compute $\mathrm{L}(a,b,c), \mathrm{R}(A,B,C)$ from $\mathrm{L}(A,B,C)$, $\mathrm{R}(a,b,c)$. We rewrite each of the Eq. (1) by switching lefthand term and first terms on the righthand from side:

$$\begin{aligned} b^\ell &= A^\ell + (b^{\ell+1}+1)b^{\ell+2} + b^{\ell+1}c^{\ell+2} + b^{\ell+2}c^{\ell+1}, \\ c^\ell &= B^\ell + (c^{\ell+1}+1)c^{\ell+2} + c^{\ell+1}a^{\ell+2} + c^{\ell+2}a^{\ell+1}, \\ a^\ell &= C^\ell + (a^{\ell+1}+1)a^{\ell+2} + a^{\ell+1}b^{\ell+2} + a^{\ell+2}b^{\ell+1}, \end{aligned}$$

We can use these equations for computing (a^2, b^2, c^2) by taking $\ell = 2$. Clearly the first term on the righthand side is part of $\mathrm{L}(A,B,C)$ and the remaining terms are expressed in terms of bits in $\mathrm{R}(a,b,c)$. We can now use this equation with $\ell = 1$ to compute (a^1, b^1, c^1) using the acquired value of (a^2, b^2, c^2). This can be repeated for $\ell = 0$ giving us the full knowledge of (a,b,c). From (a,b,c) we can compute (A,B,C) using Eq. (1) and hence we also know $\mathrm{R}(A,B,C)$. □

We can use Lemma 1 to apply a variant of the "Changing of the Guards" method to χ' that requires less state expansion and XOR gates due to the feedforward. We call it χ''.

Definition 4. *The χ'' sharing of χ is given by:*

$$\begin{aligned} \mathrm{R}(A_i) &= \mathrm{R}(\chi'_a(b_i, c_i)) + \mathrm{R}(b_{i-1}) + \mathrm{R}(c_{i-1}) && \text{for } i > 0 \\ \mathrm{R}(B_i) &= \mathrm{R}(\chi'_b(a_i, c_i)) + \mathrm{R}(c_{i-1}) && \text{for } i > 0 \\ \mathrm{R}(C_i) &= \mathrm{R}(\chi'_c(a_i, b_i)) + \mathrm{R}(b_{i-1}) && \text{for } i > 0 \\ \mathrm{L}(A_i) &= \mathrm{L}(\chi'_a(b_i, c_i)) && \text{for } i > 0 \\ \mathrm{L}(B_i) &= \mathrm{L}(\chi'_b(a_i, c_i)) && \text{for } i > 0 \\ \mathrm{L}(C_i) &= \mathrm{L}(\chi'_c(a_i, b_i)) && \text{for } i > 0 \\ \mathrm{R}(B_0) &= \mathrm{R}(c_m) \\ \mathrm{R}(C_0) &= \mathrm{R}(b_m)\,. \end{aligned}$$

Here the indexing i assumes rows arranged in a one-dimensional array. In Keccak*-p this is a two-dimensional array indexed by y and z. It is however simple to adopt a convention for converting this to a single-dimensional one, e.g. $i = y + 5z$.*

Note that $\mathrm{L}(b_0)$, $\mathrm{L}(c_0)$, $\mathrm{L}(B_0)$ and $\mathrm{L}(C_0)$ do not occur in the computations. We can therefore reduce the guards to their 2-bit right parts: $\mathrm{R}(b_0)$, $\mathrm{R}(c_0)$, $\mathrm{R}(B_0)$ and $\mathrm{R}(C_0)$.

The total expansion of the state reduces from 2 times the S-box width (totalling to 10 bits) to 4 bits. Moreover, there are only 8 XOR gates per S-box, i.e. 1.6 per native bit instead of 4 additional XOR gates per native bit. In the context of the χ' sharing the computational overhead is very small, as implementing Eq. (1) requires 9 XOR gates and 9 (N)AND gates per native bit. Note that the multi-transformation technique can be applied to other primitives that use a variant of χ as nonlinear layer.

We can now prove the following theorem.

Theorem 3. *χ'' as defined in Definition 4 is a correct, incomplete and uniform sharing of χ.*

Proof. Correctness and incompleteness is immediate. For proving uniformity we describe how to compute (a, b, c) from (A, B, C). We compute the components of (a, b, c) starting from index m down to 0. First we have $\mathrm{R}(b_m) = \mathrm{R}(C_0)$ and $\mathrm{R}(c_m) = \mathrm{R}(B_0)$. Then we can iterate the following loop going from m down to 1, computing $a_i, \mathrm{L}(b_i), \mathrm{L}(c_i)$ and $\mathrm{R}(b_{i-1}), \mathrm{R}(c_{i-1})$:

- $\mathrm{R}(a_i) = \mathrm{R}(S^{-1}(A_i + B_i + C_i)) + \mathrm{R}(b_i) + \mathrm{R}(c_i)$
- compute $\mathrm{L}(a_i, b_i, c_i)$ from $\mathrm{L}(A_i, B_i, C_i), \mathrm{R}(a_i, b_i, c_i)$ using Lemma 1
- $\mathrm{R}(b_{i-1}) = \mathrm{R}(S_c(a_i, b_i)) + \mathrm{R}(C_i)$
- $\mathrm{R}(c_{i-1}) = \mathrm{R}(S_b(a_i, c_i)) + \mathrm{R}(B_i)$. □

5 Implementation Aspects

In this section we discuss suitability of our method for decomposed S-boxes, parallel and serial architectures.

5.1 Compatibility with Serial Decomposition of S-boxes

To reduce the number of shares, one has proposed the serial decomposition of S-boxes in S-boxes of lower degree. Notably, Kutzner et al. decomposed all 4-bit S-boxes of algebraic degree 3 into component degree-2 mappings [21] in such a way that for each of the components a correct, incomplete and uniform 3-share threshold scheme can be found. One may wonder whether our method can be combined with such decomposition.

As a matter of fact, when "Changing of the Guards" is applied, the requirements on the decomposition due to sharing vanish: it suffices to find a decomposition of an invertible S-box as the series of two degree-2 S-boxes. If such a decomposition exists, but if no uniform sharing for one or both component S-boxes can be found, our method comes to the rescue. In Fig. 3 we illustrate it for the case that the "Changing of the Guards" is applied to both layers. Note that the uniformity of the composed mapping follows directly from the uniformity of the component mappings.

One can see that in between the two layers, there is a register or latch. This results in an increase of latency. The output guards of the first step are used as input guards for the second step. If one of the two layers would not require "Changing of the Guards", the guards would just skip that step. For example, if the first layer would not use the method, the incoming guards b_0, c_0 would not be used the first layer but directly in the second layer and the outcoming guards A_0, B_0 would be produced by the second layer.

In the case of more complex decompositions that combine serial and parallel composition, our "Changing of the Guards" cannot be readily applied. Especially if the decomposition contains building blocks that are not permutations. A well-known example of such decompositions are the ones applied to

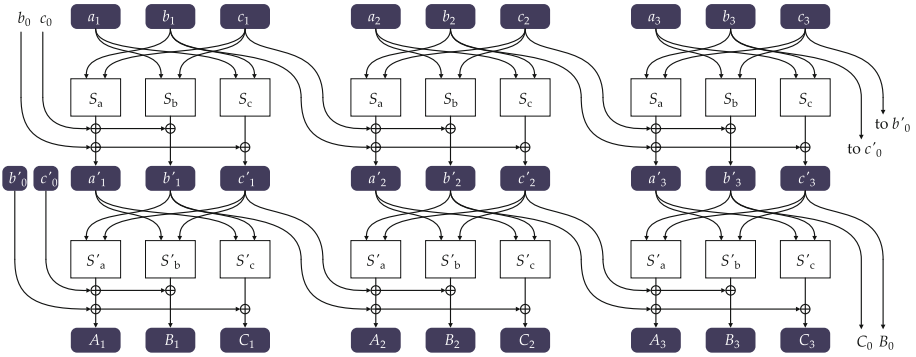

Fig. 3. Changing of the Guards applied to a layer of serially decomposed S-boxes.

the S-box of our cipher Rijndael [15] by Moradi et al. [22] and Bilgin et al. [7]. As the Rijndael S-box has algebraic degree 7 in GF(2) and hence would require 8 shares, a straightforward implementation of our proposed method would be very expensive. Due to the status of Rijndael as worldwide block cipher standard [27], it would be interesting further work to find a decomposition of the Rijndael S-box in terms of components that are all permutations of low algebraic degree.

5.2 Implementation Cost in Parallel Architectures

In a parallel architecture where the full round function is implemented in a block of combinatorial logic, the cost of the basic method is d XOR gates per bit plus d/m additional registers per bit. Introducing an extra share costs a single additional register per bit, plus possibly additional combinatorial logic. It follows that in parallel architectures the method becomes less and less interesting as d grows. It is at its best for protecting degree-2 functions, especially when a multi-transformation property can be exploited as in Keccak. As a matter of fact, Bilgin et al. [6] compare Keccak 4-share circuits with ones protected with a method that only differs from our method by the fact that the 4 bits of additional state are generated randomly every round, and hence the numbers reported there are expected to be very close to what we would achieve. A 4-share fully parallel implementation of Keccak-f[1600] turns out to be about 20% more expensive than a 3-share *guards-like* one.

5.3 Implementation Cost in Serial Architectures

In serial architectures, the combinatorial logic can be limited to a fraction of the round function and one round takes multiple cycles. In the extreme case, this logic would only contain a single implementation of the (shared) S-box. In Fig. 4 we illustrate two cases to show that our method is compatible with such an architecture, both zooming in on the circuit implementing the shared S-box.

It can be seen that the combinatorial logic is extended with two registers (called *guard*) for keeping the inputs of the previous S-box computation. Figure 4 should give a good idea of how these circuits operate, but are not fully self-explanatory as we omit some details to not overload the pictures. The single-stage circuit operates as follows:

- The S-box input arrives in the boxes indicated by *in*. Depending on the architecture these can be registers, the output of another combinatorial block or a multiplexer.
- The operation of the guard registers:
  - At the beginning of the computation, they are initialized to random values (not depicted).
  - While processing an S-box layer, they get their input from the *in* boxes.
  - After processing the last S-box of a layer, they keep their value but swap contents (not depicted).
- The S-box output is presented in the boxes indicated by *out* for further processing or storage. The guards never leave the *guard* registers.

The two-stage circuit is a pipeline and operates similarly to the single-stage one, with the following refinements:

- During operation the first stage will always be one S-box ahead of the second stage. This implies that the processing of a layer of m S-boxes will take $m+1$ cycles.
- The guard register of the first stage operate similarly to the single-stage case. The only difference is that after the last S-box of a layer has been processed,

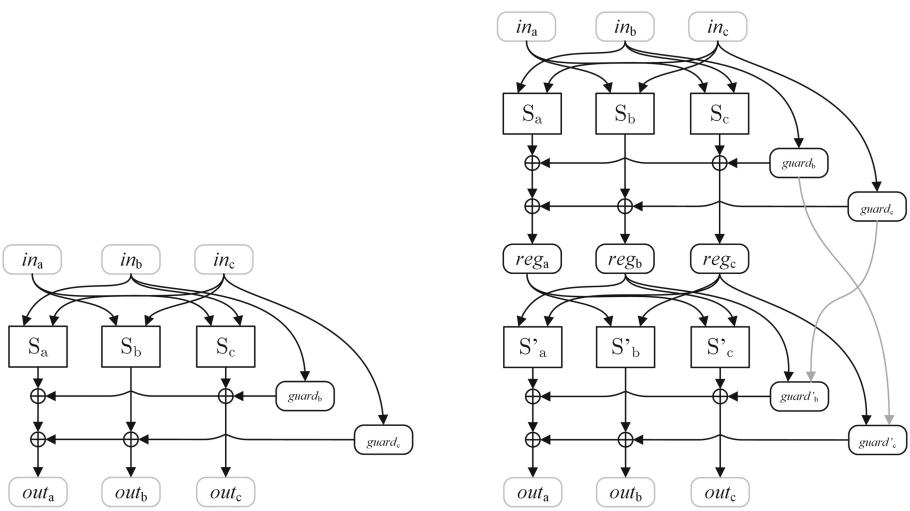

Fig. 4. Circuit for shared S-box computation in serial architecture, single-stage (left), two-stage (right)

they get their values from the guard registers of the second stage (not depicted to not overload the figure).
– The operation of the guard registers of the second stage:
  • At the beginning of the computation, they are initialized to random values (not depicted).
  • While processing an S-box layer, they get their input from the from the registers or latches, indicated by *reg*, in between the two stages.
  • After processing the last S-box of a stage, they get their value from the guard registers of the first stage.

In a serial implementation the guard registers have a higher relative overhead when comparing to the combinatorial circuit alone. However, when the real estate for keeping the state is also counted, an additional share is much more expensive than some additional XOR gates and guard registers. The exercise by Bilgin et al. [6] reports on 4-share serialized architectures that are 30% more expensive than a 3-share guards-like one.

6 Conclusions

In this paper we introduce a simple and low cost technique for achieving a 3-share correct, incomplete and uniform threshold implementation of the nonlinear layer in KECCAK. We have generalized this to a generic technique for achieving a $d+1$-share correct, incomplete and uniform threshold implementation of any S-box layer of invertible S-boxes that have degree at most d. Looking for S-boxes with uniform threshold implementations with the minimum $(d+1)$ number of shares has therefore lost relevance. On the other hand, it becomes now interesting to look for S-boxes that have $d+1$-share implementations with a suitable multi-transformation property, such as observed in the nonlinear layer of KECCAK.

Acknowledgements. I thank Gilles Van Assche, Vincent Rijmen, Begül Bilgin, Svetla Nikova and Ventzi Nikov for working with me on the paper [6], that already contained an idea very close to the "Changing of the Guards" technique, Guido Bertoni for inspiring discussions and finally Lejla Batina and Amir Moradi for useful feedback on earlier versions of this text.

References

1. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Building power analysis resistant implementations of KECCAK. In: Second SHA-3 Candidate Conference, August 2010
2. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: The KECCAK reference, January 2011. http://keccak.noekeon.org/
3. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Van Keer, R.: CAESAR submission: KETJE v2, September 2016. http://ketje.noekeon.org/
4. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Van Keer, R.: CAESAR submission: KEYAK v2, document version 2.2, September 2016. http://keyak.noekeon.org/

5. Beyne, T., Bilgin, B.: Uniform first-order threshold implementations. IACR Cryptology ePrint Archive 2016:715 (2016)
6. Bilgin, B., Daemen, J., Nikov, V., Nikova, S., Rijmen, V., Assche, G.: Efficient and First-Order DPA resistant implementations of KECCAK. In: Francillon, A., Rohatgi, P. (eds.) CARDIS 2013. LNCS, vol. 8419, pp. 187–199. Springer, Cham (2014). doi:10.1007/978-3-319-08302-5_13
7. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: A more efficient AES threshold implementation. In: Pointcheval, D., Vergnaud, D. (eds.) AFRICACRYPT 2014. LNCS, vol. 8469, pp. 267–284. Springer, Cham (2014). doi:10.1007/978-3-319-06734-6_17
8. Bilgin, B., Nikova, S., Nikov, V., Rijmen, V., Stütz, G.: Threshold implementations of all 3 × 3 and 4 × 4 S-boxes. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 76–91. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33027-8_5
9. Bilgin, B., Nikova, S., Nikov, V., Rijmen, V., Tokareva, N.N., Vitkup, V.: Threshold implementations of small S-boxes. Cryptogr. Commun. **7**(1), 3–33 (2015)
10. Božlov, D., Bilgin, B., Sahin, H.: A note on 5-bit quadratic permutations' classification. IACR Trans. Symmetric Cryptol. **2017**(1), 398–404 (2017)
11. Boss, E., Grosso, V., Güneysu, T., Leander, G., Moradi, A., Schneider, T.: Strong 8-bit S-boxes with efficient masking in hardware. In Gierlichs, B., Poschmann, A.Y. (eds.) [18], pp. 171–193 (2016)
12. Boss, E., Grosso, V., Güneysu, T., Leander, G., Moradi, A., Schneider, T.: Strong 8-bit sboxes with efficient masking in hardware extended version. J. Cryptogr. Eng. **7**(2), 149–165 (2017)
13. De Cnudde, T., Bilgin, B., Gierlichs, B., Nikov, V., Nikova, S., Rijmen, V.: Does coupling affect the security of masked implementations? IACR Cryptology ePrint Archive 2016:1080 (2016)
14. 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.) [18], pp. 194–212 (2016)
15. Daemen, J., Rijmen, V.: The Design of Rijndael — AES, the Advanced Encryption Standard. Springer, Heidelberg (2002)
16. Daemen, J.: Spectral characterization of iterating lossy mappings. IACR Cryptology ePrint Archive 2016:90 (2016)
17. Daemen, J.: Spectral characterization of iterating lossy mappings. In: Carlet, C., Hasan, M.A., Saraswat, V. (eds.) SPACE 2016. LNCS, vol. 10076, pp. 159–178. Springer, Cham (2016). doi:10.1007/978-3-319-49445-6_9
18. Gierlichs, B., Poschmann, A.Y. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2016–Proceedings of the 18th International Conference, Santa Barbara, CA, USA, 17–19 August 2016. LNCS, vol. 9813. Springer (2016)
19. 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 (2003). doi:10.1007/978-3-540-45146-4_27
20. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). doi:10.1007/3-540-48405-1_25
21. Kutzner, S., Nguyen, P.H., Poschmann, A.: Enabling 3-share threshold implementations for all 4-Bit S-boxes. In: Lee, H.-S., Han, D.-G. (eds.) ICISC 2013. LNCS, vol. 8565, pp. 91–108. Springer, Cham (2014). doi:10.1007/978-3-319-12160-4_6

22. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the limits: a very compact and a threshold implementation of AES. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 69–88. Springer, Heidelberg (2011). doi:10.1007/978-3-642-20465-4_6
23. Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 529–545. Springer, Heidelberg (2006). doi:10.1007/11935308_38
24. Nikova, S., Rijmen, V., Schläffer, M.: Secure hardware implementation of non-linear functions in the presence of glitches. In: Lee, P.J., Cheon, J.H. (eds.) ICISC 2008. LNCS, vol. 5461, pp. 218–234. Springer, Heidelberg (2009). doi:10.1007/978-3-642-00730-9_14
25. Nikova, S., Rijmen, V., Schläffer, M.: Secure hardware implementation of nonlinear functions in the presence of glitches. J. Cryptol. **24**(2), 292–321 (2011)
26. NIST: Federal information processing standard 46, data encryption standard (DES), October 1999
27. NIST: Federal information processing standard 197, advanced encryption standard (AES), November 2001
28. NIST: Federal information processing standard 202, SHA-3 standard: permutation-based hash and extendable-output functions, August 2015. doi:10.6028/NIST.FIPS.202
29. Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating masking schemes. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 764–783. Springer, Heidelberg (2015). doi:10.1007/978-3-662-47989-6_37
30. Schnorr, C.P., Vaudenay, S.: Parallel FFT-hashing. In: Anderson, R.J. (ed.) FSE 1993. LNCS, vol. 809, pp. 149–156. Springer, Heidelberg (1994). doi:10.1007/3-540-58108-1_18
31. Stoffelen, K.: Optimizing S-box implementations for several criteria using SAT solvers. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 140–160. Springer, Heidelberg (2016). doi:10.1007/978-3-662-52993-5_8