Enabling 3-share threshold implementations for all 4-bit S-boxes

Kutzner, Sebastian; Nguyen, Phuong Ha; Poschmann, Axel

2012

Kutzner, S., Nguyen, P. H., & Poschmann, A. (2012). Enabling 3-share Threshold Implementations for all 4-bit S-boxes. Cryptology ePrint Archive, 2012/510.

https://hdl.handle.net/10356/102051

Enabling 3-share Threshold Implementations for all 4-bit S-boxes

Sebastian Kutzner[1,2], Phuong Ha Nguyen[1,2] and Axel Poschmann[1,2]

[1] PACE Temasek Laboratories,
[2] Division of Mathematical Sciences, SPMS,
Nanyang Technological University, Singapore
{skutzner, aposchmann}@ntu.edu.sg, phuongha.ntu@gmail.com

Abstract. Threshold Implementation (TI) is an elegant and promising lightweight countermeasure for hardware implementations to resist first order Differential Power Analysis (DPA) in the presence of glitches. Unfortunately, in its most efficient version with only three shares, it can only be applied to 50% of all 4-bit S-boxes so far. In this paper, we introduce a new approach, called *factorization*, that enables us to protect *all* 4-bit S-boxes with a 3-share TI. This allows—for the first time—to protect numerous important ciphers to which the 3-share TI countermeasure was previously not applicable, such as CLEFIA, DES, DESL, GOST, HUMMINGBIRD1, HUMMINGBIRD2, LUCIFER, mCrypton, SERPENT, TWINE, TWOFISH among others. We verify the security and correctness with experiments on simulations and real world power traces and finally provide exemplary decompositions of all those S-boxes.

1 Introduction

In 1996, Paul Kocher [13] showed that although a cryptographic algorithm is theoretically secure, when implemented on ordinary digital circuits, the physical side-effect observed during the processing of the algorithm, such as the timing, power [14], or electromagnetic emanation [10], could potentially leak information if properly analyzed. Though the existence of these so-called *side-channels* have been known since 1943 [23], Kocher's work marks the beginning of the (public) research in the field of side-channel analysis, and powerful attacks, such as Simple Power Analysis (SPA) [14], Differential Power Analysis (DPA) [14], Correlation Power Analysis (CPA) [6], and Mutual Information Analysis (MIA) [11] have been developed since. At the same time ever more sophisticated countermeasures have been proposed. Most countermeasures aim at decreasing the *signal-to-noise ratio* (SNR) [17] by balancing the leakage, that is *hiding* the information processed [27] and/or breaking the link between the processed data and the secret, which is called *masking* [7]. However, in [18,19] it was shown that masking is still vulnerable to DPA due to the presence of glitches in hardware implementations. For that reason, a secret-sharing based countermeasure called *Threshold Implementation* (TI) [24] was proposed in 2006, that is provably secure against first-order DPA even in the presence of glitches. A few follow-up papers have discussed mostly applications to 4-bit S-boxes [24,26,25,5] and implementations of TI have been reported for PRESENT [28], AES [20] and KECCAK [3,2].

In its most resource-efficient form the TI countermeasure needs only 3 shares, which implies the function that is to be shared can have at most an algebraic degree of 2. In order to apply a 3-share TI to a function with a larger degree (4-bit S-boxes typically have a degree of 3), this function, for minimal area requirements,[1] should be represented as a composition of quadratic functions [28]. According to [28,25] there are two stages in applying the 3-share TI: the *decomposition stage*, during which a given S-box is decomposed into quadratic permutations, and the *sharing stage*, during which all those quadratic permutations are shared into 3 shares in a way that we obtain 12-bit permutations. According to [5], all 4-bit S-boxes that can be protected by a 3-share TI using the *sequential structure*, belong to the alternating group A_{16} of the symmetric group S_{16}. This result implies that we cannot apply a 3-share TI to those 50% of all 4-bit S-boxes which do not belong to A_{16}.

Our main contribution is the introduction of the *factorization structure* which is an extension of the sequential structure. This idea allows to decompose any 4-bit S-box into quadratic vectorial Boolean functions and, hence, enables to *protect any given 4-bit S-box with the TI countermeasure using only 3 shares.*

To support our claims we show how to apply the 3-share TI to SERPENT and many other 4-bit S-boxes, what, up to now, was believed to be not possible.

The remainder of this article is organized as follows. In Section 2, we recall the basics of the Threshold Implementation countermeasure. Then we will discuss how to decompose any 4-bit S-box into quadratic decompositions, i.e., studying the decomposition stage in Section 3. Subsequently, we show how to share each quadratic decomposition in a way that the uniformity property is fulfilled, i.e., being a 12-bit permutation, in Section 4. We verify our claims by successfully applying the 3-share TI countermeasure to the S-box S_5 of SERPENT, which does not belong to A_{16}. Our experimental results, provided in Section 5, verify that the protected S_5 implementation is secure against first-order DPA attacks. The paper is concluded in Section 6 and in the appendix we list 3-share TIs for S-boxes and important permutations which are not in A_{16}, and thus, previously could not have been protected by 3-share TIs.

2 Threshold Implementation

In this section we recall the preliminaries of the Threshold Implementation countermeasure and the results of [28] describing a 3-share TI of PRESENT.

2.1 Threshold Implementation Countermeasure

In [24], the Threshold Implementation (TI) was introduced as a side-channel analysis countermeasure. It is based on secret sharing and multi-party computation and provably secure against first order DPA, even in the presence of glitches. Let denote by small characters x, y, ... stochastic variables and by capitals X, Y, ... samples of these variables. The probability that x takes the value X is denoted by $Pr(x = X)$. The variable x is divided into s shares x_i, $1 \leq i \leq s$, such that $x = \bigoplus_{i=1}^{s} x_i$. Denote $\bar{x}$=(x_1, ..., x_{i-1}, x_i, x_{i+1}, ..., x_s), $\bar{x_i}$=(x_1, ...,

[1] See Section 5 for our detailed line of argumentation.

x_{i-1}, x_{i+1}, ..., x_s) (or the vector $\bar{x_i}$ does not contain the share x_i) and denote by $Pr(\bar{x} = \bar{X}|x = X)$ the conditional probability of an event that $\bar{x} = \bar{X}$ under condition $x = X$. The method can be described as follows. Let *F(x, y, z, ...)* be a vectorial Boolean function which needs to be shared. A sharing of F is a set of s functions F_i which it must fulfill the following properties:

1. **Non-completeness:** All functions F_i must be independent of at least one share of the input variables x, y, z, ... This can be translated to F_i should be independent of x_i, y_i, z_i, ..., i.e., the inputs of F_i does not have x_i, y_i, z_i, ...or F_i= $F_i(\bar{x_i}, \bar{y_i}, \bar{z_i}, ...)$.
2. **Correctness:** $F(x, y, z, ...) = \bigoplus_{i=1}^{s} F_i(\bar{x_i}, \bar{y_i}, \bar{z_i}, ...)$.

According to Theorems 2 and 3 of [24,26,25], if the inputs satisfy the following condition

$$Pr(\bar{x} = \bar{X}, \bar{y} = \bar{Y}, ...) = q \times Pr(x = \bigoplus_{i}^{s} X_i, y = \bigoplus_{i}^{s} Y_i, ...), \tag{1}$$

where q is a constant or $Pr(\bar{x} = \bar{X}, \bar{y} = \bar{Y}, ...|x = \bigoplus_i^s X_i, y = \bigoplus_i^s Y_i, ...)$ is a constant, then the sharing of F can resist first order DPA even in the presence of glitches.

In general, F is a round function (or a nonlinear function) and its output is the input of next round (or of next nonlinear function). Hence, the following property for the output of F is required in order to make the cipher resistant against first order DPA in the presence of glitches. Assume that $(u, v, ..., w) = F(x, y, ..., z)$ and $u = \bigoplus_{i=1}^{s} u_i$, $\bar{u} = (u_1, u_2, ..., u_s)$, ..., $w = \bigoplus_{i=1}^{s} w_i$, $\bar{w} = (w_1, w_2, ..., w_s)$, then the third property is defined as follows:

3. **Uniformity:** A shared version of $(u, v, ..., w) = F(x, y, ..., z)$ is uniform, if $Pr(\bar{u} = \bar{U}, ..., \bar{w} = \bar{W})$=$q \times Pr(u = \bigoplus_i^s U_i, ..., w = \bigoplus_i^s W_i)$ where q is a constant or $Pr(\bar{u} = \bar{U}, ..., \bar{w} = \bar{W}|u = \bigoplus_i^s U_i, ..., w = \bigoplus_i^s W_i)$ is a constant.

If the function $u = F(x)$ is invertible, then every vector $\bar{u}$ is reached for exactly one input vector $\bar{x}$. In this paper, the function F is a 4-bit S-box which is a 4-bit permutation. Hence, its 3-share TI is required to be 12-bit permutation.

All 4-bit permutations constitute the *symmetric group* S_{16} [12]. The identity permutation is an even permutation. An *even permutation* can be obtained as the composition of an even number and only an even number of exchanges (called transpositions) of two elements, while an *odd permutation* can be obtained by (only) an odd number of transpositions [12]. All 4-bit even permutations in S_{16} constitute a subgroup which is called the *alternating group* A_{16}. Let B_{16} be the set of all 4-bit odd permutations or B_{16}=$S_{16} \setminus A_{16}$.

Assume that the degree of F is d, then the number of shares s required is computed as follows:

Theorem 1. *[25] The minimum number of shares required to implement a product of d variables satisfying Property 1 and 2 is given by*

$$s \geq 1 + d.$$

Since the minimum degree of a nonlinear vectorial Boolean function is 2, the number of shares s is at least 3. The more shares are needed, the bigger the hardware implementation. Therefore, a 3-share TI is the most efficient—and thus, most desirably–case.

2.2 3-share TI for cubic 4-bit S-boxes

In this section we revisit the results of [28] describing a 3-share TI of PRESENT. Since the PRESENT S-box $S(\cdot)$ is a cubic 4-bit permutation, the minimum number of shares is 4 [25]. To apply 3-share TI, the S-box is decomposed into two quadratic permutations $S(\cdot) = F(G(\cdot))$ as shown in Figure 1, i.e., transforming it into a *sequential structure*.

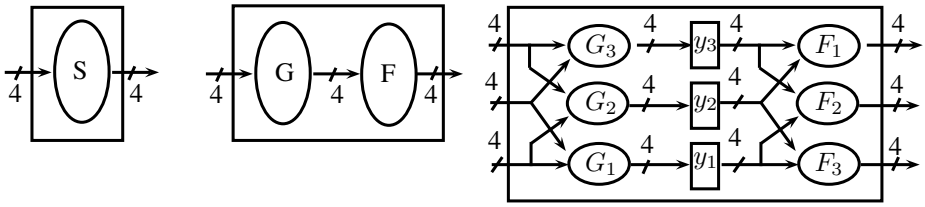


Fig. 1. Composition of the PRESENT S-box [28].

According to [28,25] there are two stages in applying 3-share TI to a 4-bit S-box when using a sequential structure:

1. **Decomposition:** Finding the decompositions of a given S-box, which are required to be quadratic permutations.
2. **Sharing:** Constructing the 3-share TIs for those quadratic permutations. Their shared versions should fulfill all three requirements, most importantly uniformity, i.e., the shared versions must be 12-bit permutations.

Note 1. In the sharing stage, constructing a 3-share TI satisfying non-completeness property and correctness property to any 4-bit at most quadratic permutation is not difficult. Unfortunately it does not guarantee that its 3-share TI is a 12-bit permutation, i.e., it is not guaranteed that the uniformity property is satisfied. For that purpose, the so-called *remasking* technique [25] may be applied, which remasks the input(s) of next round or the input(s) of the next function with fresh (and uniformly distributed) random bits.

In this paper, we discuss how to decompose an arbitrary 4-bit S-box first (decomposition stage), before we show how to obtain its 3-share TIs of decompositions which are 12-bit permutations. It means that these 3-share TIs satisfy the uniformity without using the remasking method, which is a significant advantage, as the generation of random bits suitable for cryptographic masking can be very expensive on embedded devices.

Note 2. In order to apply 3-share TI to any arbitrary 4-bit S-box, we have to extend the decomposition and the sharing stages. Those extensions yield the *factorization structure*, our main contribution in this article. We will detail these extensions in Section 3.2.

3 The Decomposition Stage

In this section we investigate the decomposability of 4-bit S-boxes. In Section 3.1, we will recall the results from [5]. If a 4-bit S-box can be decomposed in a sequential structure then it must belong to the alternating group A_{16}. In order to apply a 3-share TI to the remaining 50% of 4-bit S-boxes which are in B_{16}, we extend the sequential structure by using our new idea: *Factorization*, which yields the *factorization structure*. Our contribution allows to decompose any 4-bit S-box.

3.1 Decomposition of 4-bit S-boxes using a Sequential Structure

Assume that $S(\cdot)= F\ (\ldots\ G(\ H(\cdot)))$ and if S is a permutation then all its decompositions H, G, ..., F have to be permutations as well. Hence, if S is a 4-bit permutation then H, G, ..., F are also 4-bit permutations. We recall the following important result about permutations in S_{16}.

Theorem 2. *[5] If a permutation $F(\cdot)$ is a composition of quadratic permutations, then $F(\cdot)$ is in A_{16}.*

However, 50% of all 4-bit S-boxes are not decomposable using a sequential structure, i.e., all those S-boxes belong to B_{16}. Hence, there is no method known so far on how to apply 3-share TIs to those S-boxes. We now introduce a new methodology to solve this open problem.

3.2 Decomposition of 4-bit S-boxes using a Factorization Structure

We start with a very simple example, i.e., decomposing a cubic term. The *Algebraic Normal Form* (ANF) of a cubic 4-bit S-box contains at least one cubic term. Without loss of generality, we first assume that the ANF contains only one cubic term $T(w, z, y, x)= (d, c, b, a)=(xyz, 0, 0, 0)$. The input bits of T are x, y, z, w and the output bits of T are a, b, c, d. The left most bit represents the most significant bit and the right most bit represents the least significant bit, respectively. The ANF of T is:

$$\begin{aligned} d &= xyz \\ c &= 0 \\ b &= 0 \\ a &= 0. \end{aligned}$$

We can also write T as follows:

$$T(\cdot) = F(G(\cdot)) \oplus V(\cdot),$$

where F, G and V are the following quadratic vectorial Boolean functions:

$$G: \begin{array}{l} d = xy \oplus w \\ c = z \\ b = y \\ a = x. \end{array} \quad F: \begin{array}{l} d = zw \\ c = 0 \\ b = 0 \\ a = 0. \end{array} \quad V: \begin{array}{l} d = zw \\ c = 0 \\ b = 0 \\ a = 0. \end{array}$$

As one can see, using this approach, it is possible to represent a cubic term of an ANF by a set of quadratic vectorial Boolean functions. By applying this approach term-by-term, it is possible to decompose any cubic vectorial Boolean function, *including odd permutations*. This observation results in the following theorem

Theorem 3. *For any given 4-bit S-box S we can always find a set of quadratic vectorial Boolean functions F_i, G_i, $1 \leq i \leq n$, and V such that:*

$$S(\cdot) = \bigoplus_{i=1}^{n} F_i(G_i(\cdot)) \oplus V(\cdot).$$

We call the format above *factorization structure* and we summarize the idea for the decomposition stage in Figure 2.

In order to make our idea clear, we provide another example, the 4-bit odd permutation M=[0, 1, 2, 3, 4, 5, 6, 15, 8, 9, 10, 11, 12, 13, 14, 7]. Its ANF is:

$$\begin{array}{l} d = w \oplus xyz \\ c = z \\ b = y \\ a = x. \end{array}$$

Please note that the example above is only *one* out of many choices for M. We chose M for its simplicity and implementation efficiency. The permutation M can be factorized as follows:

$$M(\cdot) = M_2(M_1(\cdot)) \oplus M_3(\cdot).$$

Where the ANFs of M_1, M_2, and M_3 are:

$$M_1: \begin{array}{l} d = xy \oplus w \\ c = z \\ b = y \\ a = x. \end{array} \quad M_2: \begin{array}{l} d = zw \\ c = 0 \\ b = 0 \\ a = 0. \end{array} \quad M_3: \begin{array}{l} d = zw \oplus w \\ c = z \\ b = y \\ a = x. \end{array}$$

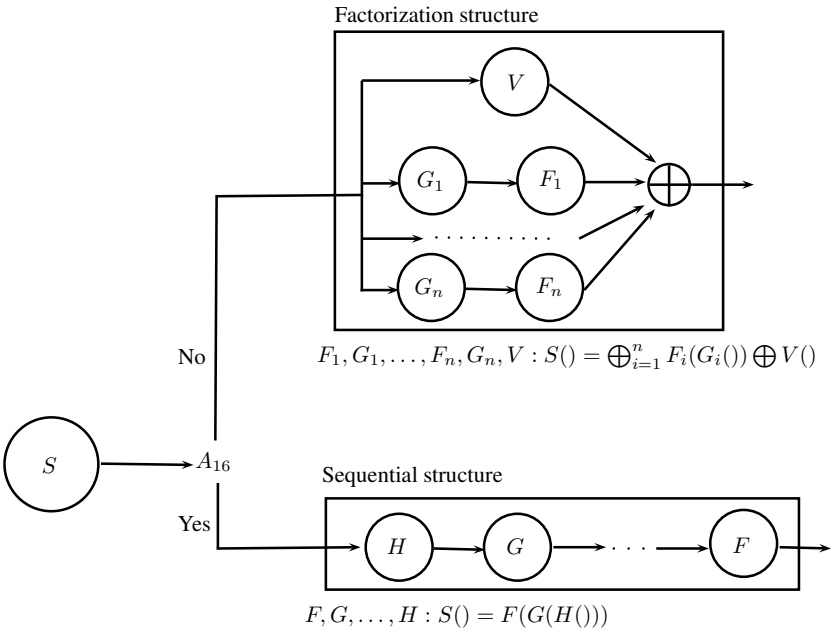


Fig. 2. Flowchart of the decomposition stage for a given 4-bit S-box.

3.3 Decomposition of 4-bit S-boxes using a Hybrid Structure

We will discuss the sharing stage in Section 4 where we ensure that the 3-share TI of a 4-bit S-box is a 12-bit permutation, i.e., the 3-share TI fulfills the uniformity property. Please note that the sharing stage for sequential structures is not complicated, and we will show how to solve this problem in Section 4.1.

Sharing a factorization structure, however, without using the remasking method to fulfill the uniformity is a challenge, and it is extremely difficult for 4-bit S-boxes which have many cubic terms. In order to make the workload in the sharing stage easier, we propose the following approach:

1. If a given S-box S is in A_{16} (S is an even permutation) then we use the method in Section 4.1.
2. If a given S-box S is in B_{16} (S is an odd permutation) then:
   (a) Construct a 4-bit odd permutation M that can be shared into a 12-bit permutation, for example M=[0, 1, 2, 3, 4, 5, 6, 15, 8, 9, 10, 11, 12, 13, 14, 7]. This permutation is decomposed by using a factorization structure and it can be ensured that its 3-share TI is a 12-bit permutation, i.e., permutation M satisfies uniformity. This will be shown in Section 4.2.
   (b) Since M and S are odd permutations, the permutation $S^{'}$ such that $S(\cdot) = M(S^{'}(\cdot))$ is an even permutation, i.e., $S^{'}$ is in A_{16} [12]. Then we apply the result in Section 4.1 to share the decompositions of $S^{'}$.

Actually, the method above is a *hybrid structure* between a sequential structure and a factorization structure. This hybrid structure is very useful, because

it can help us to fulfill the uniformity property without using remasking, thus we will work with the hybrid structure instead of a plain factorization structure. Figure 3 depicts the hybrid structure.

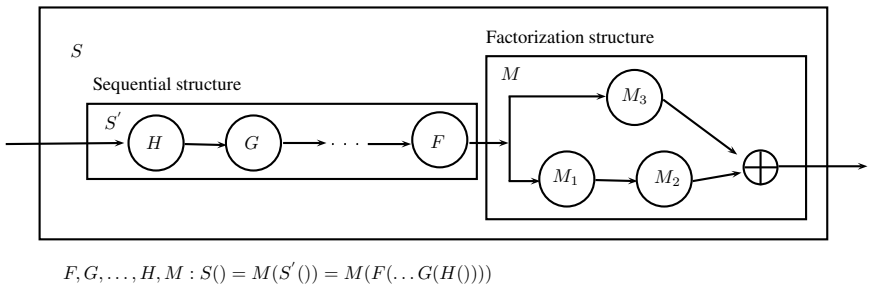


Fig. 3. Hybrid Structure.

So far, we already presented how to decompose a given 4-bit S-box. If the given S-box is in A_{16} we use a sequential structure, otherwise we use a hybrid structure which is a mixture of a sequential and a factorization structure.

Now that the decomposition stage is done, we can move on to the sharing stage, which will be treated in the next section. It is interesting to see how to construct the 3-share TIs of these decompositions such that they are 12-bit permutations, i.e., they fulfill the uniformity property without using the remasking method.

3.4 Application to important S-boxes in B_{16}

In the appendix we list decompositions of S-boxes in B_{16}, that are used by the following algorithms: CLEFIA [32], DES [22], DESL [15], GOST [35], HUMMINGBIRD1 [9], HUMMINGBIRD2 [8], LUCIFER [33], mCrypton [16], SERPENT [4], TWINE [34], TWOFISH [31], and the Inversion (x^{-1}) function in $GF(2^4)$ which is used in mCrypton [16] or in [20]. Previously, all of these algorithms could not have been protected by a 3-share TI.

4 The Sharing Stage

In this section, we discuss how to make 3-share TIs of decompositions of a given 4-bit S-box being 12-bit permutations. Since the given S-box can belong to A_{16} or B_{16}, we have two structures: a sequential structure, which is treated in Section 4.1, and a hybrid structure, which is composed of a sequential structure and a factorization structure. Consequently, in Section 4.2 we will treat the factorization structure, and we will also discuss its security, i.e., why it resists first order DPA in the presence of glitches and why it satisfies uniformity.

4.1 Sharing Stage using a Sequential Structure

In this subsection, we discuss how to share the decompositions of a given S-box S in A_{16}. We adopt the sharing in [24,25,28]. For a given function $F(w, z, y, x)$ and inputs x, y, z, w, they are split into 3 shares F_1, F_2, F_3, $x_1, \ldots, w_3$. For monomials involving two indices, it is obvious which F_i to place them in. For example, we must place monomials y_1w_2 and z_2w_1 in F_3. For monomials involving just one index, e.g., x_1 or y_2w_2, we adopt the convention that terms with index 1 (resp. 2, 3) are placed in F_3 (resp. F_1, F_2). The constant term is placed in F_1. In [5], this approach is called *direct sharing*. For example a given Boolean function $f{=}xy \oplus z \oplus 1$ then the 3-share TI by using the direct sharing is as follows:

$$
\begin{aligned}
f_1 &= z_2 \oplus x_2y_2 \oplus x_2y_3 \oplus x_3y_2 \oplus 1 \\
f_2 &= z_3 \oplus x_3y_3 \oplus x_1y_3 \oplus x_3y_1 \\
f_3 &= z_1 \oplus x_1y_1 \oplus x_1y_2 \oplus x_2y_1
\end{aligned}
$$

According to [5], all 4-bit permutations in A_{16} can be decomposed by using a sequential structure and their 3-share TIs satisfy uniformity by simply using direct sharing.

4.2 Sharing Stage using a Factorization Structure

The sharing stage for S-boxes in B_{16} is detailed here. As we have already pointed out, it is better to use a hybrid structure to decompose an S-box in B_{16}. In the previous section, based on [5], we already saw that all 3-share TIs of 4-bit permutations in A_{16} can be made being 12-bit permutations by using direct sharing. Hence, we only focus on the 3-share TI of M, i.e., the 3-share TIs of the quadratic vectorial Boolean functions M_1, M_2, M_3. Let denote M_s, M_{s_1}, M_{s_2}, M_{s_3} as 3-share TIs of M, M_1, M_2, M_3, respectively.

The ANF of the 12-bit M_{s_1} of M_1 is:

$$
\begin{aligned}
d_1 &= w_2 \oplus y_2x_2 \oplus y_2x_3 \oplus y_3x_2 \\
d_2 &= w_3 \oplus y_3x_3 \oplus y_1x_3 \oplus y_3x_1 \\
d_3 &= w_1 \oplus y_1x_1 \oplus y_1x_2 \oplus y_2x_1 \\
c_1 &= z_2 \\
c_2 &= z_3 \\
c_3 &= z_1 \\
b_1 &= y_2 \\
b_2 &= y_3 \\
b_3 &= y_1 \\
a_1 &= x_2 \\
a_2 &= x_3 \\
a_3 &= x_1.
\end{aligned}
$$

The ANF of the 12-bit M_{s_2} of M_2 is:

$$\begin{aligned}
d_1 &= z_2w_2 \oplus z_2w_3 \oplus z_3w_2\\
d_2 &= z_3w_3 \oplus z_1w_3 \oplus z_3w_1\\
d_3 &= z_1w_1 \oplus z_1w_2 \oplus z_2w_1\\
c_1 &= 0\\
c_2 &= 0\\
c_3 &= 0\\
b_1 &= 0\\
b_2 &= 0\\
b_3 &= 0\\
a_1 &= 0\\
a_2 &= 0\\
a_3 &= 0.
\end{aligned}$$

The ANF of the 12-bit M_{s_3} of M_3 is:

$$\begin{aligned}
d_1 &= w_2 \oplus z_3w_3 \oplus z_2w_3 \oplus z_3w_2\\
d_2 &= w_3 \oplus z_1w_1 \oplus z_1w_3 \oplus z_3w_1\\
d_3 &= w_1 \oplus z_2w_2 \oplus z_1w_2 \oplus z_2w_1\\
c_1 &= z_2\\
c_2 &= z_3\\
c_3 &= z_1\\
b_1 &= y_2\\
b_2 &= y_3\\
b_3 &= y_1\\
a_1 &= x_2\\
a_2 &= x_3\\
a_3 &= x_1.
\end{aligned}$$

Then $M_s = M_{s_2}(M_{s_1}(\cdot)) \oplus M_{s_3}(\cdot)$ is a 12-bit permutation and thus M fulfills uniformity. Note that all the sharings of all decompositions of M are found by hand due to the simplicity of their ANF.

Note 3. Studying Figure 3 allows to observe the following:

1. Among three quadratic vectorial Boolean functions M_1, M_2, M_3, only M_1 is a 4-bit permutation and M_{s_1} is a 12-bit permutation. Hence, M_1 satisfies the uniformity property.
2. It is obvious that M_{s_2} and M_{s_3} do not fulfill the uniformity property. However, both functions are *not required to satisfy uniformity*, only their XOR sum has to (as this will potentially be the input to a subsequently shared function). Instead both functions M_{s_2} and M_{s_3} only need to satisfy non-completeness and correctness, and their inputs need to fulfill Equation 1, in order to resist first order DPA in the presence of glitches (Theorems 2 and 3 in [24,26,25]).

3. The output of M_s is the result of XORing the outputs of M_{s_2} and M_{s_3}. Since an XOR is a linear operation (i.e., having degree 1), only outputs from the same share are combined together. This means, this operation is first order DPA resistant in the presence of glitches, as potential leakage will only depend on a single share, but information of all three shares are required for a successful DPA.
4. Since M is in B_{16} and M_s is a 12-bit permutation, the 3-share TI of M satisfies uniformity. It means our proposed hybrid structure is secure.
5. We will present experimental results for supporting our theoretical arguments about the security of our proposed structure in the next section.

5 Experiments

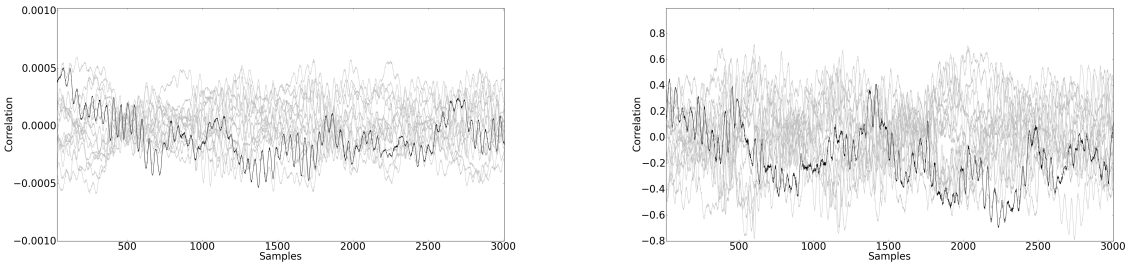
To verify the correctness and security of our new scheme we decomposed and shared the SERPENT S-box S_5 as described in the previous sections. The decomposition formulas for all stages can be found in the Appendix. We ensured that every stage of the shared S-box fulfills all requirements given in [24], especially the uniformity property. It should also be noted that a register has to be inserted in between every decomposition stage. Figure 5 shows a schematic of the hardware implementation.

First, several attacks were mounted on noise-free simulated power traces (assuming a HW leakage), i.e., CPA attacks on all (intermediate) registers. Neither of the attacks revealed the correct key hypothesis, hence supporting our claims. For testing purposes we also attacked a non-uniform implementation of S_5, i.e., M_3 was varied such that the XOR-output is not uniform. Here, a CPA attack on the output register was successful, proving (again) the importance of the uniformity property.

Next, we implemented the (uniform) shared S-box as shown in Figure 5 on an FPGA, i.e., a SASEBO-GII. To synthesize the design we used Xilinx ISE Webpack 13.3. The FPGA hosting the S-box ran at 2 MHz derived from the 24 MHz on-board oscillator. 20,000,000 measurements were taken at 1.25 GS/s (625 samples per clock cycle) and a CPA was performed, using the Hamming distance between the outputs of two consecutive S-box lookups as the attack model. Furthermore, we mounted a correlation-enhanced collision attack [21] to test the resistance against glitches, as failing attacks on registers do not necessarily prove the security of a scheme [19]. Figure 4 shows the results of both attacks. The first clock edge is at sample 125. One S-box computation takes four clock cycles, thus the computation is finished at sample 2625. As we can see, in neither of both attacks does the correct hypothesis yield the highest correlation.

Efficiency of 3-share TI for 4-bit Sboxes based on hybrid structure:

One of important factors in masking countermeasure is randomness. Generating a random number used for cryptographic purpose is very expensive in terms of time and power consumption because at least one full encryption of a cipher should be processed (hash function, block cipher, ...)[1,30]. Therefore, the number of random numbers used should be as small as possible in crypto system. In lightweight crypto designs the serialized implementation and 4-bit S-boxes are preferred due to their hardware compactness. [5] has provided 3-share TIs of all 4-bit S-boxes in A_{16} and 4- and 5-share TIs for the other in B_{16}. Some



(a) CPA results attacking the SERPENT S-box S_5.

(b) Correlation-enhanced collision attack

Fig. 4. Attack Results

of the 4- and 5-share TIs require less area than the corresponding 3-share TI with hybrid structure. Thus, the authors concluded that a 3-share realization may not be the optimal case in terms of hardware. However, it should be noted that all shares need to be maintained (i.e. stored) throughout the whole encryption process. In a lightweight setting, i.e. serialized implementation, the S-box layer contributes only 15% to the whole area [28], while the registers take the lion's share. Thus any reduction of the number of shares will reduce the overall gate count significantly at the potential cost of a slightly larger S-box. Hence, in all cases (with or without using remasking method) a 3-share TI is much more efficient than 4- and 5-share TI in terms of hardware and randomness (or time and power consumption).

It is well-known that the number of shares can be reduced to 2 for the linear layer to reduce the storage overhead for the shares. However, this approach requires extensive use of fresh randomness which is an expensive resource especially in embedded systems for the remasking step. The more shares are used in the non-linear part, the more randomness is required. Thus this approach nullifies the elegance of TI as a lightweight DPA countermeasure (only needing randomness once in the beginning), and consequently it is convinced the 3-share case is the most optimal in all aspects.

6 Conclusion

Threshold Implementation (TI) [24] is an elegant and promising lightweight countermeasure for hardware implementations to resist first order Differential Power Analysis (DPA) in the presence of glitches. The most challenging part in applying TI to ciphers are the non-linear functions, e.g., the S-boxes for block ciphers. To implement TI in its most efficient version, namely with only three shares, the functions to be shared have to have a degree smaller than three. For 50% of all S-boxes this requirement can be fulfilled by decomposing an S-box of degree three to several functions with a smaller degree [28,5]. After decomposition, the quadratic and linear functions can now be split into three shares.

Unfortunately, for the other 50% of the S-boxes, said method can not be applied. Therefore, we introduced the *factorization structure* which enables us to decompose those functions of degree three, which previously were deemed

to be not decomposable, to several quadratic and linear functions. It should be noted that the shared version of a function has to fulfill certain properties to be secure, namely correctness, non-completeness and uniformity. Using the *factorization structure* exacerbates fulfilling those requirements; especially the uniformity property is a very challenging task. Therefore, we introduced the *hybrid structure* combining the previous method and our *factorization structure*, enabling us to decompose cubic functions into quadratic and linear functions which subsequently can be easily shared and simultaneously fulfill all requirements of TI.

References

1. NIST Special Publication 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators, 2012. Technical report. Available via `http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf`.
2. G. Bertoni, J. Daemen, N. Debande, T.-H. Le, M. Peeters, and G. Van Assche. Power analysis of hardware implementations protected with secret sharing. Cryptology ePrint Archive, Report 2013/067, 2013. `http://eprint.iacr.org/`.
3. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Building power analysis resistant implementations of keccak. In *Second SHA-3 Candidate Conference*, 2010.
4. E. Biham, R. Anderson, and L.R. Knudsen. Serpent: A New Block Cipher Proposal. In S. Vaudenay, editor, *FSE1998*, volume 1372 of *Lecture Notes in Computer Science*, pages 222-238. Springer-Verlag, 1998.
5. B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, and G. Stütz. Threshold Implementations of All 3×3 and 4×4 S-Boxes. In E. Prouff and P. Schaumont, editors, *CHES2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 76-91. Springer-Verlag, 2012.
6. E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In *CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer-Verlag, 2004.
7. J.S. Coron and L. Goubin. On Boolean and Arithmetic Masking against Differential Power Analysis. In C.K. Koc and C. Paar, editors, *CHES2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 231-237. Springer-Verlag, 2000.
8. D. Engels, M.J.O. Saarinen, P. Schweitzer, and E.M. Smith. The Hummingbird-2 Lightweight Authenticated Encryption Algorithm. In A. Juels and C. Paar, editors, *RFIDSec2011*, volume 7055 of *Lecture Notes in Computer Science*, pages 19-31. Springer-Verlag, 2011.
9. X. Fan, H. Hu, G. Gong, E. M. Smith, and D. Engels. Lightweight Implementation of Hummingbird Cryptographic Algorithm on 4-Bit Microcontroller. In *ICITST 2009*, 2009.
10. K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic Analysis: Concrete Results. In *CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer-Verlag, 2001.
11. B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. Mutual Information Analysis. In E. Oswald and P. Rohatgi, editors, *CHES2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 487-496. Springer-Verlag, 2008.
12. N. Jacobson. *Basic Algebra.* 1 (2nd ed.). Dover. ISBN 978-0-486-47189-1.
13. P.C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *CRYPTO1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 104-113. Springer-Verlag, 1996.
14. P.C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M.J. Weiner, editor, *CRYPTO1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 388-397. Springer-Verlag, 1999.
15. G. Leander, C. Paar, A. Poschmann, and K. Schramm. New Lighweight DES Variants. In A. Biryukov, editor, *FSE2007*, volume 4593 of *Lecture Notes in Computer Science*, pages 196-210. Springer-Verlag, 2007.
16. C.H Lim and T. Korishko. mCrypton - A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors. In J. Song, T. Kwon, and M. Yung, editors, *WISA2005*, volume 3786 of *Lecture Notes in Computer Science*, pages 243-258. Springer-Verlag, 2005.
17. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag, 2007.

18. S. Mangard, T. Popp, and B.M. Gammel. Side-Channel Leakage of Masked CMOS Gates. In A. Menezes, editor, *CT-RSA2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 351-365. Springer-Verlag, 2005.
19. S. Mangard, N. Pramstaller, and E. Oswald. Successfully attacking masked AES hardware implementations. In J.R. Rao and B. Sunar, editors, *CHES2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 157-171. Springer-Verlag, 2005.
20. A. Moradi, A. Poschmann, C. Paar, and H. Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In K.G. Paterson, editor, *EUROCRYPT2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 69-88. Springer-Verlag, 2011.
21. Amir Moradi, Oliver Mischke, Christof Paar, Yang Li, Kazuo Ohta, and Kazuo Sakiyama. On the power of fault sensitivity analysis and collision side-channel attacks in a combined setting. In *Cryptographic Hardware and Embedded Systems–CHES 2011*, pages 292–311. Springer, 2011.
22. U.S Department of Commerce National Bureau of Standards. Data Encryption Standard. Technical report, 1977. Available via `http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf`.
23. National Security Agency. TEMPEST: A Signal Problem. *Cryptologic Spectrum*, 2(3), 1972 (declassified 2007).
24. S. Nikova, C. Rechberger, and V. Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In P. Ning, S. Quing, and N. Li, editors, *ICICS2006*, volume 4307 of *Lecture Notes in Computer Science*, pages 529-545. Springer-Verlag, 2006.
25. S. Nikova, V. Rijmen, and M. Schläffer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. *Journal of Cryptology*, 24(2):292-321, 2011.
26. S. Nikova, V. Rijmen, and M. Schläffer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. In P. Lee and J. Cheon, editors, *ICISC2008*, volume 5461 of *Lecture Notes in Computer Science*, pages 218-234. Springer-Verlag, 2008.
27. T. Popp and S. Mangard. Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints. In J.R. Rao and B. Sunar, editors, *CHES2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 172-186. Springer-Verlag, 2005.
28. A. Poschmann, A. Moradi, K. Khoo, C. Lim, C. Wee, H. Wang, and S. Ling. Side-Channel Resistant Crypto for Less than 2,300 GE. *Journal of Cryptology*, 24(2):322-345, 2011.
29. M.J.O. Saarinen. Cryptographic Analysis of All 4×4 S-boxes. In A. Miri and S. Vaudenay, editors, *SAC2011*, volume 7118 of *Lecture Notes in Computer Science*, pages 118-133. Springer-Verlag, 2011.
30. W. Schindler. *Random Number Generators for Cryptographic Applications*. Cryptographic Engineering, Springer, 2009.
31. B. Schneier, J. Kelsey, D. Whiting, D Wagner, C. Hall, and N. Ferguson. The Twofish Encryption Algorithm. Technical report, 1998.
32. Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit blockcipher CLEFIA. In *Fast Software Encryption*, pages 181–195. Springer-Verlag, 2007.
33. A. Sorkin. Lucifer, a cryptographic algorithm. *Cryptologia* , 8(1):22-41, 1984.
34. T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi. TWINE: A Lightweight Block Cipher for Multiple Platforms. In L.R. Knudsen and H. Wu, editors, *SAC2012*, volume 7707 of *Lecture Notes in Computer Science*, pages 339-354. Springer-Verlag, 2013.

35. I.A. Zabotin, G.P. Glazkov, and V.B. Isaeva. Cryptographic Protection for Information Processing Systems, Government Standard of the USSR, GOST 28147-89, Government Committee of the USSR for Standards. Technical report, 1989.

A Appendix: 3-share TIs of S-boxes in B_{16}

In this section, we present the 3-share TIs of some S-boxes or important permutations which are in B_{16} by using a hybrid structure. All examples use the odd permutation M=[0, 1, 2, 3, 4, 5, 6, 15, 8, 9, 10, 11, 12, 13, 14, 7] which is also used in previous sections. Recall, that M can be any odd permutation of which the shared version is a 12-bit permutation, i.e., satisfying the uniformity property without using remasking.

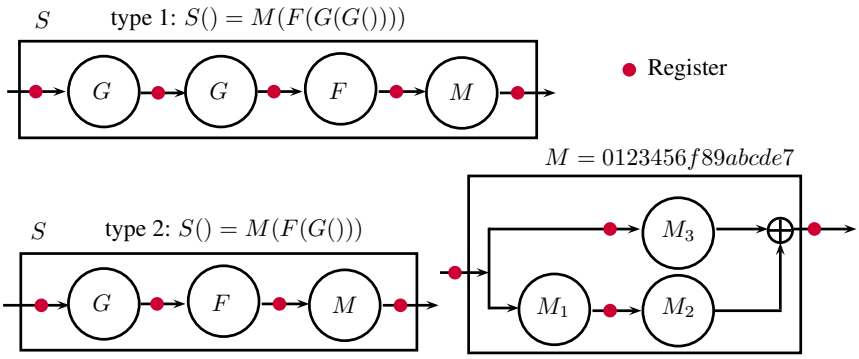


Fig. 5. Decompositions of S-boxes.

For the sake of convenience, a given permutation is described in hexadecimal representation. For example, if a permutation F=[15, 5, 6, 14, 13, 7, 2, 10, 8, 0, 11, 1, 12, 4, 9, b], then F is written as follows: F=f56ed72a80b1c493. All S-boxes in this section can be found in [29,5] or from their respective specifications.

All S-boxes below belong to B_{16} and they can be decomposed in two different ways (see Figure 5):

- **type 1:** $S(\cdot)$=$M(F(G(G(\cdot))))$
- **type 2:** $S(\cdot)$=$M(F(G(\cdot)))$

In fact nearly all S-boxes belong to type 1 and only two S-boxes (iS_4 of HUMMINGBIRD2 and S_5 of SERPENT) belong to type 2. The 3-share TIs of all F and G by using direct sharing are 12-bit permutations.

CLEFIA [32]

1. SS_0: F=e6ca89d24b10537f, G=021346fda89bce57;
2. SS_1: F=6f29a5e3781cd4b0, G=053f8db72694ae1c;
3. SS_2: F=b56e7302da981cf4, G=094c187f2b6e3a5d;
4. SS_3: F=a6d295c37bf048e1, G=02319b8a57ec46df;

DES [22] Actually, the $i-th$ DES S-box ($DESi$) contains a set of four 4-bit S-boxes. Notation $DESi_j$ means the $j-th$ row (i.e., 4-bit S-box) of the $i-th$ DES S-box.

1. $DES2_0$: F=f986bda42c710e35, G=4c28a0f7d539b16e;
2. $DES2_1$: F=acd1265b97403fe8, G=1c593a7f0d482b6e;
3. $DES2_2$: F=d3f0c481b596a2e7, G=9d26a78503cf4e1b;
4. $DES2_3$: F=dc8b37421f6a5e09, G=0b1a46579382decf;
5. $DES3_0$: F=d69e75a410f2b3c8, G=168c079d24be35af;
6. $DES3_1$: F=803a46ed952f17bc, G=17069a8b5243dfce;
7. $DES3_2$: F=df47ae50b921c836, G=0d861c972ea53fb4;
8. $DES3_3$: F=9716fac0b8e4d532, G=fb647ec318a59d02;
9. $DES4_0$: F=fd3402cb75168ae9, G=419b03c8de62fa57;
10. $DES4_1$: F=3edf68ba70c41592, G=125ac68e9bd34f07;
11. $DES4_2$: F=abc4fd928375e610, G=094b6a285d1f3e7c;
12. $DES4_3$: F=36dea581b2f047c9, G=02d64f135e8a9bc7;
13. $DES5_0$: F=28fc1b569a7d304e, G=0e1f869725bcad34;
14. $DES6_0$: F=792bd3c54a81e06f, G=4e396f18a0d7c5b2;
15. $DES6_3$: F=48ac537b2e9f601d, G=0a7c1e68295f3d4b;
16. $DES7_0$: F=6b3d719c2e5a8f40, G=21e74da903c56f8b;
17. $DES7_1$: F=68f143bc970ead52, G=be364f290c1d57a8;
18. $DES7_2$: F=abd4c93e671805f2, G=1a084e5c293b7d6f;
19. $DES8_0$: F=d572c908143be6af, G=0eb4962c1da7853f;
20. $DES8_1$: F=fd963b2745c01ae8, G=1c0d3a2b59487f6e;
21. $DES8_2$: F=fa41e5830b6d72c9, G=048c9d152f6b3e7a;

DESL [15]

1. Row_0: F=e6a3d4197f2b5c80, G=091d7f6b5c482a3e;
2. Row_1: F=51ebc9378d6204af, G=02cf1b5e93d68a47;
3. Row_2: F=15dbef74c2a63809, G=17ad358f269c04be;
4. Row_3: F=dae51379f80b64c2, G=af53269e8d7104bc;

GOST [35]

1. k_3: F=52840cadb79e613f, G=063d1f24acb5978e;
2. k_4: F=f93457dec1a62b08, G=0e7d1b4a2c5f3968;
3. k_7: F=d7954f6b2c08e1a3, G=0a6f384c1b7e295d;
4. k_8: F=5b79d3f104ae62c8, G=179fda52e46cb038;

HUMMINGBIRD1 [9]

1. S_0: F=82f7e639c40ab1d5, G=0f1e9687bd24ac35;
2. S_1: F=063b7f42d1eca895, G=0f861e97ad24bc35;
3. S_2: F=21430895dbeca76f, G=0ad7b16c92e54f38;
4. S_3: F=0f2e7d5c4a6b3819, G=0a7f295c6e1b4d38;

HUMMINGBIRD2 [8]

1. S_1: F=f56ed72a80b1c493, G=0a5bd38217ce469f;
2. S_2: F=a8034ce7b61d52f9, G=14860d9fae3cb725;
3. S_3: F=2f6e5d1c4a380b79, G=0f5bc78293d64a1e;
4. S_4: F=0819ae37c4d562fb, G=853b29a47ed1f06c;

The inverse S-boxes of HUMMINGBIRD2:

1. iS_1: F=0d42ca8597eb631f, G=3c4b21de56a9780f;
2. iS_2: F=de8c94b162305f7a, G=14a69d2fcbe05378;
3. iS_3: F=c36740b18e5d2fa9, G=0c6f2a583b491d7e;
4. iS_4: F=f5ac403b16927ed8, G=209a8b3164fced75; (type 2)

Inversion (x^{-1}) in $GF(2^4)$ The function x^{-1}=019edb76f2c5a438 which is defined over $GF(2)/(x^4 \oplus x \oplus 1)$.
F=843dae67f25bc91 and G=059dbf278e3416ac.

LUCIFER [33]

1. S_0: F=a2fde8b7906534c1, G=1e482c6b0f593d7a;
2. S_1: F=f21deb047c93658a, G=068f9e174bd3c25a;

mCrypton [16]

1. S_0: F=4af0827c3516b9de, G=0a7c5e28396d4f1b;
2. S_1: F=19df3b647580cea2, G=06d71fce8a5b9342;
3. S_2: F=31078f46ec25ad9b, G=2b5f097d4e186c3a;
4. S_3: F=b420af918c7e3d65, G=041d3f26ac97b58e;

SERPENT [4]

1. S_3: F=072e351c9db4af86, G=0c792f5a3e4b1d68;
2. S_4: F=53bd19f708e6ca24, G=ea5d69cf0873214b;
3. S_5: F=7c4b259a3e6f01d8, G=05432761c89feabd; (type 2)
4. S_7: F=18679d3f5acb024e, G=0d87961c3fa4b52e;

The inverse S-boxes of S_3, S_4, S_5, S_7:

1. iS_3: F=09dacef3b1624578, G=0c483e7a6f2b195d;
2. iS_4: F=98b7406fac5e21d3, G=1a0bc2d34e5f8796;
3. iS_5: F=87f6dc43b915e2a0, G=0eb63c95842da71f;
4. iS_7: F=35f921edc60a874b, G=0c489d5173bfa6e2;

TWINE [34]

1. S: F=d2305ebc7a98f614, G=bda5e92c0687431f;

TWOFISH [31]

1. $q1, t1$: F=a0f2d785c139b64e, G=0c483e7a6f2b195d;
2. $q1, t0$: F=2847ba6e1c9d350f, G=069c8d1734aebf25;
3. $q0, t0$: F=50d87b3fa6e29c14, G=b4ace16058732f9d;
4. $q0, t2$: F=456f09ba23e781dc, G=0d841cb73e952fa6;