Threshold Implementations of GIFT: A Trade-off Analysis

Naina Gupta, Arpan Jati, Anupam Chattopadhyay, Somitra Kumar Sanadhya, and Donghoon Chang

Abstract

Threshold Implementation (TI) is one of the most widely used countermeasure for side channel attacks. Over the years several TI techniques have been proposed for randomizing cipher execution using different variations of secret-sharing and implementation techniques. For instance, direct-shares is the most straightforward implementation of the threshold countermeasure. But, its usage is limited due to its high area requirements, whereas, the 3-shares countermeasure for cubic non-linear functions significantly reduces area and complexity compared to direct-shares.

Nowadays, security of ciphers using a side channel countermeasure is of utmost importance. This is due to the wide range of security critical applications from smart cards, battery operated IOT devices to accelerated crypto-processors. Such applications have different requirements (higher speed, energy efficiency, low latency, small area etc.) and hence need different implementation techniques. This paper presents an in-depth analysis of the various ways in which TI can be implemented for a lightweight cipher. We chose GIFT for our analysis as it is currently the most energy-efficient lightweight cipher. We present nine different profiles using different implementation techniques and show that no single technique is good for all scenarios. For example, the direct-shares technique is good for high throughputs whereas 3-shares is suitable for constrained environments with less area and moderate throughput requirements. The techniques presented in the paper are also applicable to other blockciphers. For security evaluation, we performed CPA on the 3-shares technique as it has good area versus speed trade-off. Experiments using 3 million traces show that it is protected against first-order attacks.

I. Introduction

Implementing secure embedded systems has been a cat-and-mouse game since the last two decades due to the constant development of side-channel attack techniques followed by new countermeasures. The security of even the smallest of embedded devices is of a major concern; as most of these devices have become such an important part of our daily lives. The seminal work by Kocher et al. [1], [2] in the late 90's showed the world that unprotected cryptographic algorithms are vulnerable against side-channel attacks.

Over the years, many countermeasure techniques have been proposed to prevent such attacks, for instance introducing noise in the signal [3], to randomize the intermediate values i.e. masking [3], to balance the power consumption in circuit's design [4], etc. Despite these countermeasures, the devices are still vulnerable to some form of the side-channel attacks or the other; for example, masking still leaks some form of information in the presence of glitches [5], [6]. In 2006, Nikova et al. proposed a new countermeasure known as Threshold Implementation (TI) [7]. TI is based on secret-sharing and is secure even in the presence of glitches. TI soon became one of the most widely used countermeasures. As a result, there has been a lot of work in the past years towards developing new methodologies for secret-sharing and efficient implementation of TI. For example, in [8] the authors show how to apply TI on the PRESENT cipher. Later, in 2013 Kutzner et al. [9] present the *one S-box for all* technique to efficiently implement 3-shares. Furthermore, the authors in [10] describe how to speed-up the search for decomposed S-box and also derive the results for TI on all 3×3 and 4×4 S-boxes. Efficient TI implementation of AES is presented in [11]. The design exploration using all these TI methodologies and implementation techniques have not yet been performed carefully. In this work, we focus on performing such a detailed design analysis of TI using GIFT [12], which was introduced by Banik et. al. in CHES 2017. It is currently the smallest block cipher reported in literature.

Our contribution. First, we present a Correlation Power Analysis (CPA) [13] attack for an unprotected FPGA implementation of the GIFT cipher in section IV-A. Since a single round of GIFT uses 64-bit keys at a time and each S-box operation uses only 2-bits of the key, we implemented the attack 4 S-boxes at a time. In our experiments using Xilinx Kintex-7 FPGA, we were able to recover the key in less than 10,000 traces. Second, we implemented an efficient TI countermeasure for GIFT. The implementation is protected against first-order CPA attacks. We support this claim by providing experimental results of CPA on 3,000,000 real power traces in section IV-B. Third, we implemented the known TI techniques and provide a trade-off analysis in terms of area, frequency, latency, power and energy. In particular, we focused on three TI techniques – 3-shares, combined 3-shares and direct-shares using various options as discussed in section III-B. For all of the experiments we considered a round-based implementation of the cipher. Sharing of the non-linear function (in our case S-Box) yields Boolean equations in Algebraic Normal Form (ANF). The implementation can directly be done using ANF, or it can be further minimized using a Boolean minimization tool like Espresso [14], [15], BOOM [16], ABC [17] etc. In our analysis, we found that logic minimization using Espresso and ABC leads to similar results in terms of overall area for GIFT. Whereas, a major difference

N. Gupta, A. Jati and D. Chang, Department of Computer Science and Engineering, Indraprastha Institute of Information Technology, New Delhi, India.
S. Sanadhya, Department of Computer Science and Engineering, Indian Institute of Technology Ropar, Punjab, India.
A. Chattopadhyay, School of Computer Science and Engineering, Nanyang Technological University, Singapore.

was found between an implementation using ANF compared to the Boolean minimization tools. As a result, we present detailed analysis contrasting these two implementation methods. In many implementations the key-update masking is skipped, but even for a very simple key-schedule the leakage of *hamming weight* for certain parts of the key is possible, hence we considered it in our analysis.

We implemented all the `TI` schemes and analyzed the synthesis results using the same library (TSMC 65nm Low Power). As discussed in section III-C, the 3-shares technique is 44.9% smaller but requires twice the number of clock cycles compared to the direct-shares technique. It is noteworthy to observe that both the designs have very similar overall energy requirements. Further, the 3-shares technique results in 13.8% larger area but is about five times faster than the one using combined 3-shares technique. The former is also 2.9 times energy efficient.

II. Preliminaries

A. *GIFT Specifications*

`GIFT` is a SPN (substitution-permutation network) based cipher. Its design is strongly influenced by the cipher `PRESENT` [18]. It has two versions `GIFT-64-128`: 28 rounds with a block size of 64-bits and `GIFT-128-128`: 40 rounds with 128-bit blocks. Both the versions have 128-bit keys. For this work, we focus only on `GIFT-128-128`.

Initialization. The cipher state S is first initialized from the 128-bit plaintext represented as 32 4-bit nibbles $w_{31}, \dots w_2, w_1, w_0$. The 128-bit key is divided into 16-bit words $k_7, k_6, \dots, k_0$ and is used to initialize the key register K.

The Round Function. Each round of the cipher comprises of a Substitution Layer (S-layer) followed by a Permutation Layer (P-layer) and a XOR with the round-key and predefined constants (AddRoundKey).

S-layer (S). Apply the same S-box to each of the 4-bit nibbles of the state S. The truth-table for the S-box is as follows:

TABLE I
GIFT S-box

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
S(x)	1	a	4	c	6	f	3	9	2	d	b	7	5	0	8	e

P-layer (P). This operation permutes the bits of the cipher state S from position i to $\mathsf{P}(i)$. Please refer to the design document [12] for the full permutation table.

AddRoundKey. XORs a 64-bit round key RK and a 7-bit round constant Rcon to a part of the cipher state S. The round key is extracted from the 128-bit key register K as $RK = U||V$ where $U \leftarrow k_5||k_4$ and $V \leftarrow k_1||k_0$. The round key $U||V$ can be represented as $= u_{31}, \dots, u_1, u_0||v_{31}, \dots, v_1, v_0$. U and V are XORed to the cipher state as follows: $b_{4i+2} \leftarrow b_{4i+2} \oplus u_i$ and $b_{4i+1} \leftarrow b_{4i+1} \oplus v_i$ $\forall i \in \{0, \dots, 31\}$. The round constant $(c_5c_4c_3c_2c_1c_0)$ and a single-bit '1' is XORed to the cipher state as defined below:
$b_{n-1} \leftarrow b_{n-1} \oplus 1$, $b_{23} \leftarrow b_{23} \oplus c_5$, $b_{19} \leftarrow b_{19} \oplus c_4$, $b_{15} \leftarrow b_{15} \oplus c_3$, $b_{11} \leftarrow b_{11} \oplus c_2$, $b_7 \leftarrow b_7 \oplus c_1$ and $b_3 \leftarrow b_3 \oplus c_0$, where $n - 1, 23, 19, 15, 11, 7$ *and* 3 denote bit positions in the cipher state respectively.

Key Expansion and Constants Generation. After AddRoundKey, the key register is updated as follows $k_7||k_6|| \dots ||k_1||k_0 \leftarrow k_1 \ggg 2||k_0 \ggg 12|| \dots ||k_3||k_2$. The 6-bit round constant is initialized to zero and is updated before each round as $(c_5, c_4, c_3, c_2, c_1, c_0) \leftarrow (c_4, c_3, c_2, c_1, c_0, c_5 \oplus c_4 \oplus 1)$.

`GIFT` Encryption. As shown in Fig. 1, a single block is processed by the application of a series of round functions. At each round, S-layer, P-layer and AddRoundKey operations are performed on the previous cipher state. After 40 such rounds, the current state is provided as the ciphertext.

B. *Threshold Implementation: Requirements*

As mentioned in section I, `(TI)` is based on *secret-sharing* and *multi-party* computations. Over time, `TI` has received widespread adoption as it works even in the presence of *glitches* where certain other countermeasure techniques fail [8], [19], [11], [10], [9]. Initially, the `TI` was proposed to prevent only first-order attacks only. But recently, `TI` has been successfully applied to prevent *Higher Order DPA* attacks as well [20]. `TI` needs the following three properties to be satisfied:

1) **Correctness:** The property states that the cumulative output of all the shares should be same as the output of the function without sharing.
2) **Non-completeness:** Every function should be independent of at-least d shares in order to prevent the d^{th} order attack. This is the most important property of `TI`. It is due to this property that `TI` works even with glitches.

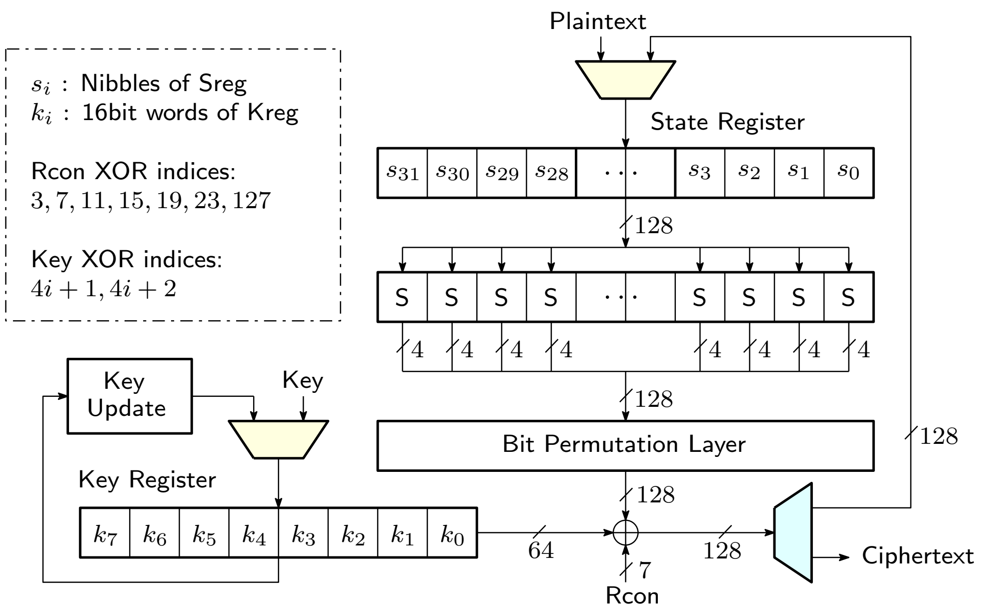


Fig. 1. `GIFT` Encryption

3) **Uniformity:** At every point of execution, the shares should be uniformly distributed. This property ensures that the mean leakages when the cipher is executing are independent of the state.

III. Implementations and Design Architecture

A. Different variants of `TI`

In this section, we discuss about the three known variants for Threshold Implementations in detail:

1) Sharing using Decomposition of S-box with cubic algebraic degree (3-shares)
2) Sharing using Decomposition with one S-Box for all (combined 3-shares)
3) Direct Sharing (direct-shares)

Sharing using Decomposition (3-shares). In 2011, Poschmann et. al. [8] proposed a technique to decompose a cubic S-box function into two quadratic functions G and F represented as $S(X) = F(G(X))$ where $S, G, F : GF(2)^4 \rightarrow GF(2)^4$. Fig. 2 shows this method graphically. As the `GIFT` S-Box is cubic, we can use this technique to decompose it into two quadratic functions. Considering the input and output of $G(X)$ as 4-bit vectors $X = (x, y, z, w)$ and $G(X) = (g_3(X), g_2(X), g_1(X), g_0(X))$. Each g_i, being a quadratic Boolean function, can be represented in ANF as shown below:

$$\begin{aligned} g_i(x, y, z, w) = a_{i,0} + a_{i,1}x + a_{i,2}y + a_{i,3}z + a_{i,4}w + a_{i,13}xz \\ + a_{i,14}xw + a_{i,23}yz + a_{i,24}yw + a_{i,34}zw \end{aligned}$$

where, $a_{i,j}$ are the binary coefficients of the Boolean function. Similar Boolean functions and equations hold for $F(X)$.

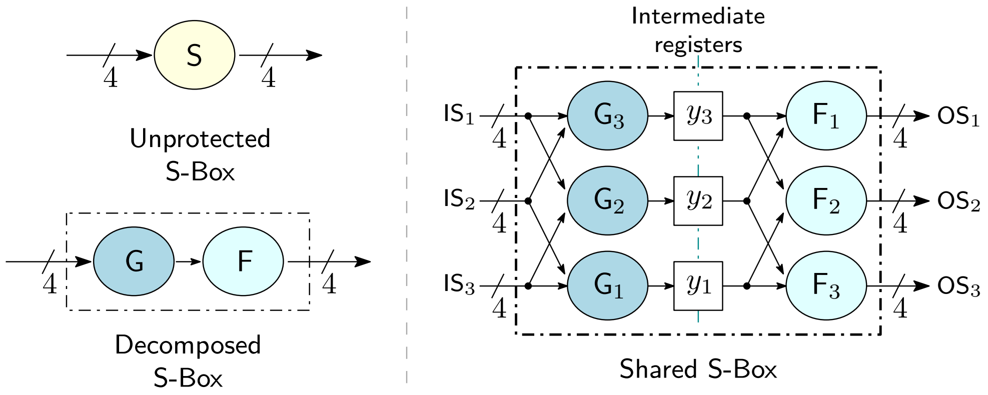


Fig. 2. Sharing using Decomposition (3-shares)

As discussed in [8], the following two facts were used to reduce the overall search space for the two decomposed functions G and F:

1) Rewriting $S(X) = F(G(X))$ as $S(G^{-1}(X)) = F(X)$, one needs to search only for all possible quadratic functions for $G(X)$. This is then used to compute the other quadratic function $F(X)$ as $S(G^{-1}(X))$.
2) Assuming $G(0) = 0,\ G'(x) = G(X) + G(0)$ and $F'(X) = F(X + G(0))$, the decomposed equation $S(X) = F(G(X))$ can be re-written as $S(X) = F'(G'(X))$. This step helps in considering only the variable coefficients in the ANF, thus reducing the overall search space for the decomposition.

Following steps were implemented in order to compute the desired optimized G and F quadratic Boolean functions:

1) For all possible combinations of the input to the functions g_i, f_i where $i \in \{0, 1, 2, 3\}$, compute its corresponding output from the ANF equations and check if its a vectorial boolean function [21] is balanced or not. If the combination is balanced, then add it to a set of possible coefficients for the ANF (say P), otherwise discard it.
2) For each balanced coefficient in the set P, compute the corresponding $G(X)$ iteratively.
3) Check whether this computed $G(X)$ is a permutation or not. If yes, compute $F(X)$ using $S(G^{-1}(X))$, otherwise discard this $G(X)$.
4) Check whether the computed $F(X)$ is a quadratic function or not. If yes, add both the $G(X)$ and $F(X)$ functions to a set of possible decompositions, otherwise discard both of them. We obtained 80641 possible decompositions after this step.
5) Now considering the 15 possibilities of the constant term in the ANF, we obtained 1290241 total possible decompositions for `GIFT` S-box after filtering.
6) Keep only the $G(X)$ and $F(X)$ combinations which are permutations, discard the rest.
7) In order to choose the decomposition with minimum area, we applied the following two metrics:
   - For each of the possible decomposition, calculate the total ANF weight of $G(X)$ and $F(X)$ using the formula provided in [8]. Sort this set based on the total weight in ascending order.
   - After the first metric, we used the LIGHTER tool [22] to generate a good estimate in GE [1] (*gate equivalents*) for an efficiently implemented hardware circuit for the decomposition.

   Finally, we choose the decomposition with a trade-off between minimum total ANF weight and minimum total GE according to the LIGHTER tool

The finally chosen $G(X)$ and $F(X)$ satisfying all the three `TI` requirements - Correctness, Non-Completeness and Uniformity are shown in Table II. The chosen $G(X)$ belongs to Q_{293} quadratic class and $F(X)$ belongs to Q_{294} class[23]. The ANFs for

TABLE II
GIFT S-BOX DECOMPOSITION

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
G(x)	4	d	f	7	1	a	2	8	5	c	e	6	0	b	3	9
F(x)	5	6	3	8	1	2	7	c	9	e	f	0	d	a	b	4

both the quadratic functions are as below:

$$\begin{aligned} G(d,c,b,a) &= (g_3, g_2, g_1, g_0) \\ g_0 &= a + b + ba + c + d \\ g_1 &= b + ca \\ g_2 &= 1 + c \\ g_3 &= a + b + cb \end{aligned}$$

$$\begin{aligned} F(d,c,b,a) &= (f_3, f_2, f_1, f_0) \\ f_0 &= 1 + a \\ f_1 &= a + b \\ f_2 &= 1 + b + c + d + da \\ f_3 &= ba + d \end{aligned}$$

The corresponding ANFs for eight output shares are provided in the Appendix A.

Sharing using Decomposition (combined 3-shares). In [9], Kutzner et al. proposed a new methodology to implement the threshold countermeasure presented in [8]. The technique is based on optimizing the area requirements for the protected

[1]GE: Total cell area divided by the cell area of a 2-input NAND gate.

implementation of a non-linear operation using multiplexers. Referring to ANF equations for the chosen $G(X)$ and $F(X)$ in Appendix A, one can clearly see that G_1, G_2 and G_3 comprise of similar polynomials and only the indices are different. Similarly, F_1, F_2 and F_3 share a similar template. The constant terms are handled in the respective $G(X)$ and $F(X)$ function. So, instead of using six different (8×4 bit) Boolean functions, we used only two functions – one for $G(X)$ and another for $F(X)$.

As shown in the Fig. 3, two multiplexers are used to choose the input for the $G(X)$ Boolean function depending on which part of the secret it is operating on. After that, a de-multiplexer is used to store the result of the $G(X)$ operation to the requisite register. $F(X)$ is implemented in a similar manner and the result is stored in the respective output registers OS_1, OS_2 and OS_3.

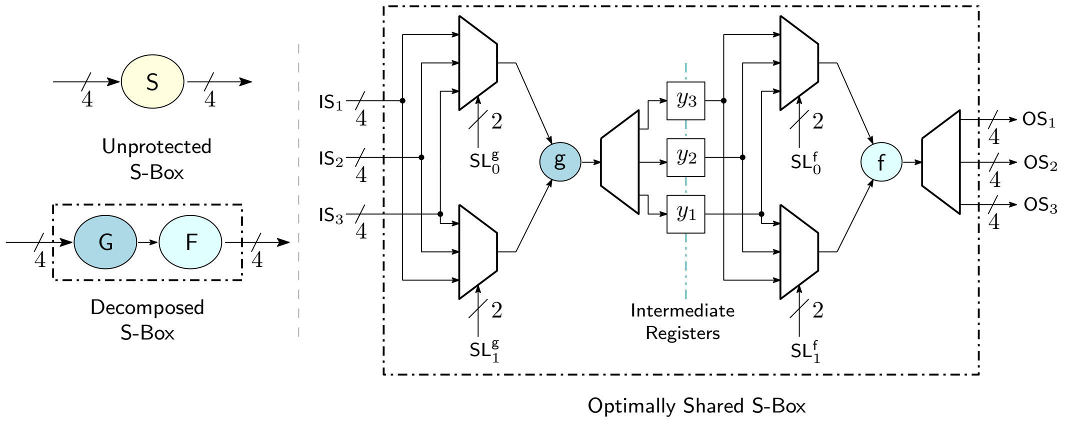


Fig. 3. Sharing using Decomposition (combined 3-shares)

Direct Sharing (direct-shares). For TI implementation using direct-shares, one use the minimum required number of shares to share the secret variables. The minimum number of shares, s required to protect a Boolean function from *first-order* DPA attack is given by $s \geq 1 + d$, where d is the algebraic degree of the function [24]. For example, the function $F(X, Y, Z) = XY + Z$ has an algebraic degree of two. Hence, it requires at-least three shares. The ANF equations for the function F are as stated below:

$$F_1 = Z_2 + X_2Y_2 + X_2Y_3 + X_3Y_2$$
$$F_2 = Z_3 + X_1Y_3 + X_3Y_1 + X_3Y_3$$
$$F_3 = Z_1 + X_1Y_1 + X_1Y_2 + X_2Y_1$$

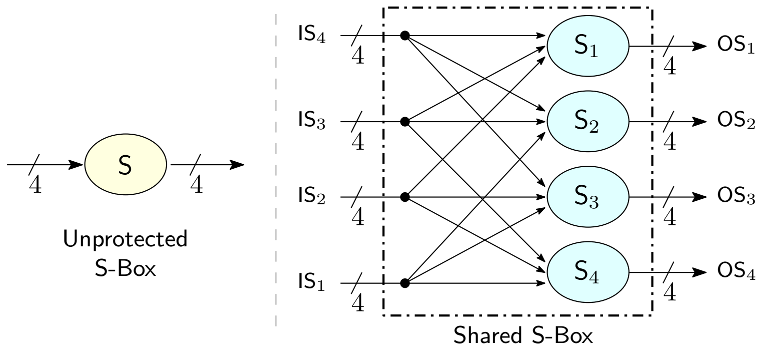


Fig. 4. Direct Sharing (direct-shares)

In case of GIFT, the only non-linear operation is its S-box. The S-box is a 4×4 Boolean function (represented as $S(d, c, b, a) \rightarrow (w, z, y, x)$) and has a cubic degree. Hence, we need a minimum of four shares. Fig. 4 shows the approach

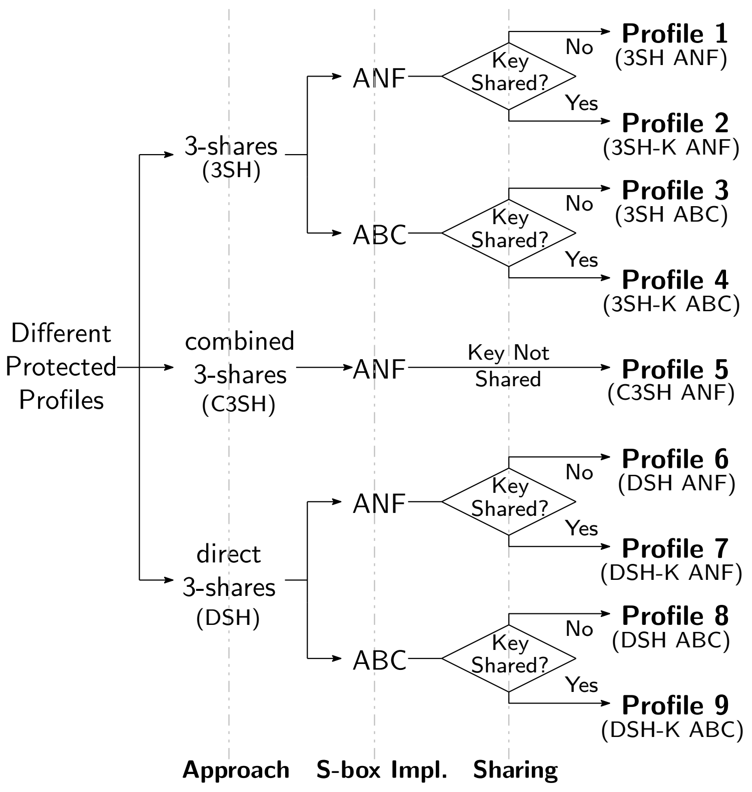


Fig. 5. Different Profiles for Threshold countermeasure

graphically. The truth table for `GIFT` S-box is as shown in Table I and its corresponding ANFs are given as:

$$\begin{aligned} S(d,c,b,a) &= (s_3,s_2,s_1,s_0) \\ s_0 &= 1+a+b+ba+c+d \\ s_1 &= a+ba+c+ca+d \\ s_2 &= b+c+da+db+dcb \\ s_3 &= a+db+dca \end{aligned}$$

The output shares (OS_1, OS_2, OS_3, OS_4) can be calculated from the above equations. The ANF for the four shares are listed in Appendix B.
An advantage of this technique is that there is no need for additional *registers* in the S-layer. As this approach does not attempt to reduce the degree of the Boolean function before implementation, it results in implementations with significantly large area compared to other techniques.

B. Implementation Profiles and Their Architecture

Here, we present nine different profiles for `TI-GIFT` implementation and discuss about various trade-offs. The profiles are a combination of an approach (described in section III-A) with an option. The different options which can be combined with an approach are described as below:

Option 1: Sharing of the *data-path*
Option 2: Sharing of the *key-register*
Option 3: S-box implemented using ANF
Option 4: S-box equations optimized using ABC

Since all the profiles are protected, data-path is shared for all. As shown in Fig. 5, *Profile 1* uses the 3-shares approach with data sharing and the S-box implemented using ANF representation. *Profile 2* is same as *Profile 1* with an extra shared key register. In *Profile 3*, ABC is used to optimize the S-box. It uses the 3-shares approach with data sharing. Compared to *Profile 3*, *Profile 4* adds sharing of the key register. *Profile 6...9* use same set of options as in *Profile 1...2*, but uses the direct-shares approach. *Profile 5* uses the combined 3-shares approach using multiplexers to switch between the input and output of $G(X)$ and $F(X)$. The data-path is shared in *Profile 5* with ANF representation being used for the S-box implementation. Fig. 6 presents an overall architecture for all the variants of threshold countermeasures we implemented. The solid lines depict

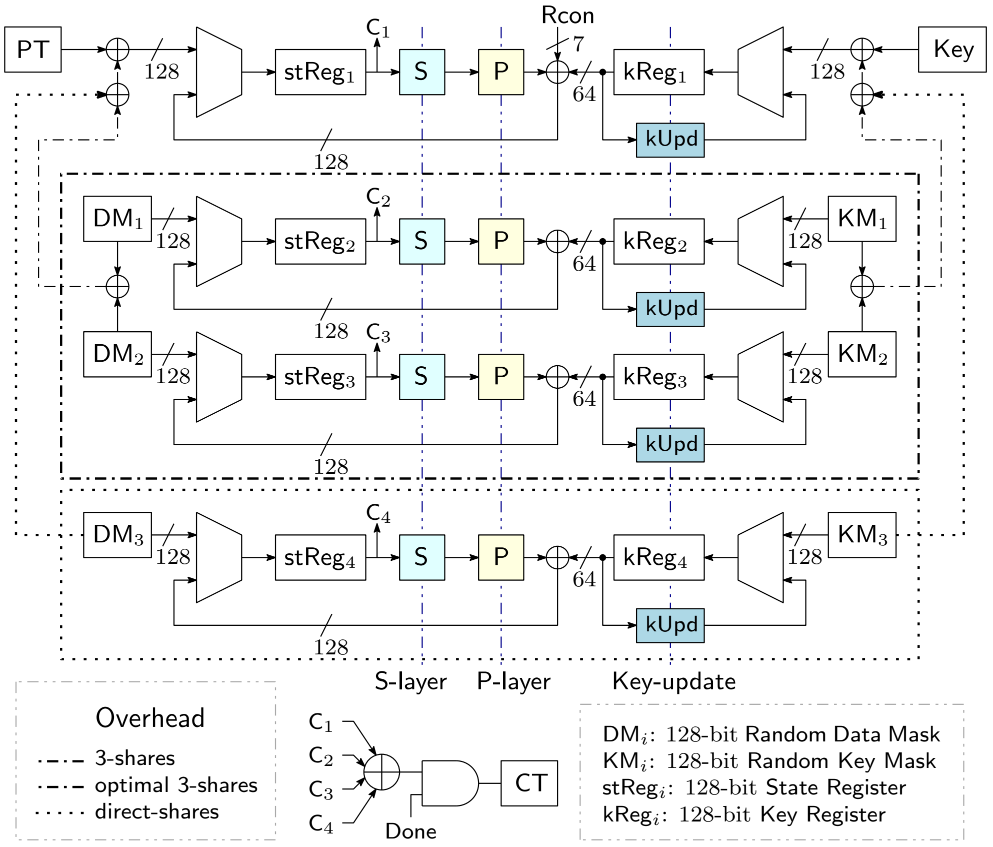


Fig. 6. Overall Architecture for `TI` techniques for `GIFT` S-box

the unprotected `GIFT` implementation. The unprotected implementation comprises of a state-register (stReg_1), a key-register (kReg_1), a bit-permutation layer and the S-box layer. stReg_1 is used to keep the current state. A multiplexer is used to select between the updated state and the input. The same holds for the kReg_1 key register. The state is updated after applying the S-box, bit-permutation, key and round constant (Rcon) addition steps. For a *parallelized* implementation, one round of unprotected `GIFT` takes one clock-cycle to update the state-register. So it takes 40 clock cycles to process one block of data.

Additional hardware required for *Profile 1…5* are marked by dashed-dotted regions in Fig. 6. *Profile 1…4* requires two random-mask values (DM_1 and DM_2 128-bit each), two additional state registers (stReg_2 and stReg_3), two additional multiplexers and some XORs. Furthermore, if the key is also shared as in the case for *Profile 2 and 4*, two random-masks (KM_1 and KM_2 128-bit each) for the key, two key registers (kReg_2 and kReg_3) and two multiplexers are also required. Implementation of the S-box layer for these profiles depends on whether it is using ANF or ABC, but the overall architecture presented in Fig. 2 remains the same. These profiles also require three additional registers to store the intermediate state in the S-box, hence they take 2 clock-cycles per round of the cipher. As a result, these profiles need 80 clock-cycles in all to process a block. In case of *Profile 5*, all the hardware overhead compared to *Profile 1…4* differs only in the architecture of the S-box. The S-box in this case is implemented using multiplexers and de-multiplexers as shown in Fig. 3. *Profile 5* requires seven times more clock-cycles compared to the unprotected implementation.

Profile 6…9 use the direct-shares technique for `TI`. In this case, in addition to the hardware overheads for 3-shares technique, a random-mask (DM_3), a state-register (stReg_4) and a multiplexer is required if only the data-path is shared as in the case of *Profile 6 and 8*. *Profile 7 and 9* share both the data-path and the key-register, thus they need an additional random-mask (KM_3), a key-register (kReg_4) and a multiplexer. The details of the corresponding S-box is shown in Fig. 4. In all of the profiles, the unmasking step is performed by XORing all the respective shares.

C. Synthesis Results

The HDL designs for all of the implementation profiles were written in VHDL[2]. Functional testing was done using the *Xilinx Vivado Simulator* version 2016.3. After functional testing, we used *Synopsys Design Compiler* version J-2014.09 for synthesis

[2]The HDL files will be available on *github* after the review process.

of the designs. *Synopsys IC Compiler* version L-2016.03-SP5-1 was used for placement and routing. We used TSMC 65nm Low Power Standard Cell Library (TCBN65LP) for all the ASIC implementations. We used compile_ultra during synthesis to get an optimized design. We also used flags to prevent optimization between hierarchal boundaries. *Synopsys PrimeTime* version J-2014.12-SP3-1 was then used on the post-layout design in conjunction with activity factors from simulations done using *Vivado* in order to get accurate power consumption estimates. For this analysis, we focused on getting a balanced design with good area vs. throughput trade-off and hence avoided any specific optimization; this is because aggressive optimization towards area leads to poor timing results and vice versa. It is also important to note that power estimates assume the design running at the highest possible clock speed. Running the designs at lower clock speeds leads to significantly reduced dynamic power consumptions; under such conditions leakage power can be the primary contributor to overall power consumption. The area and power overheads for the random source has not been considered and we assume that the randomness is provided externally.

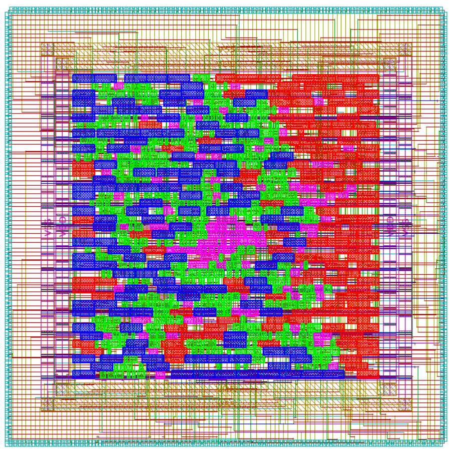
TABLE III
POST-LAYOUT RESULTS FOR DIFFERENT PROFILES OF THRESHOLD COUNTERMEASURE

Metric	**Unprotected**	**Protected Profiles**								
	GIFT	1	2	3	4	5	6	7	8	9
		3SH	3SH-K	3SH	3SH-K	C3SH	DSH	DSH-K	DSH	DSH-K
		ANF	ANF	ABC	ABC	ANF	ANF	ANF	ABC	ABC
S-Box Area (GE)	632	7286	7286	7661	7657	1705	18129	18110	84103	84198
State Register Area (GE)	800	3358	3358	3360	3360	4477	3200	3206	3258	3314
Key Register Area (GE)	801	1125	3359	1125	3360	808	801	3200	801	3202
Total Area (GE)	2478	13349	16595	13728	16964	**11729**	24233	27340	90426	93597
Ratio	1.000	5.387	6.697	5.540	6.846	4.733	9.779	11.033	36.492	37.771
Time (ns)	2.31	2.68	2.71	2.74	2.68	4.6	3.52	3.56	5.56	5.93
Frequency (MHz)	432	373	369	364	373	217	284	280	179	168
# Clocks	40	80	80	80	80	280	40	40	40	40
Throughput (Mbps)	1286	562	556	548	562	94	**845**	833	532	500
S-Box Power (mW)	0.51	3.01	2.95	3.1	3.13	0.42	5.61	5.52	19.6	18.7
State Register Power (mW)	0.7	2	2.01	2.05	2.13	2.66	3.04	3.1	2.54	2.36
Key Register Power (mW)	0.64	0.72	1.82	0.72	1.81	0.06	0.52	1.62	0.32	1
Total Power (mW)	2.396	7.578	9.217	7.75	9.687	3.716	10.3	11.9	23.8	23.6
Energy (pJ/bit)	1.777	12.859	15.809	13.487	16.438	37.701	**11.625**	13.624	42.664	45.013
Random bits	0	256	512	256	512	256	384	768	384	768

Fig. 7 shows the *placed* and *routed* physical design for the unprotected, and one of the protected designs. All the protected profiles were compiled using the same script (with different clock constraints). The script was written to accommodate some moderate variations in design complexity.

Table III shows the implementation results for all the profiles. As expected, the protected implementations require more resources than the unprotected one. The smallest protected implementation C3SH is 4.7 times larger. It is clear that most of the area is taken up by the S-Box. As direct-sharing leads to very large Boolean equations, the overall area becomes quite large. Depending on the number of shares, key-sharing can triple or quadruple the size of the key-register size. C3SH uses a sequential design as the decomposed S-Box share a similar template. Multiplexers and de-multiplexers are then used to update the state for all the 3 shares. This leads to a large number of clocks and an extra intermediate state register, but leads to an overall smaller design.

It is also interesting to contrast ABC based implementation results with ANF ones. For 3SH the difference is small, whereas for DSH the difference is quite significant (4.6 times). We believe the reason for this difference is the very large size of expressions in case of direct-sharing. In these experiments it is clear that any Boolean-minimization is not required as the synthesis tool, *Synopsys Design Compiler* was able to perform efficient minimization as it had access to a large library of logic primitives.


(a) Unprotected `GIFT`, width = 100 μm

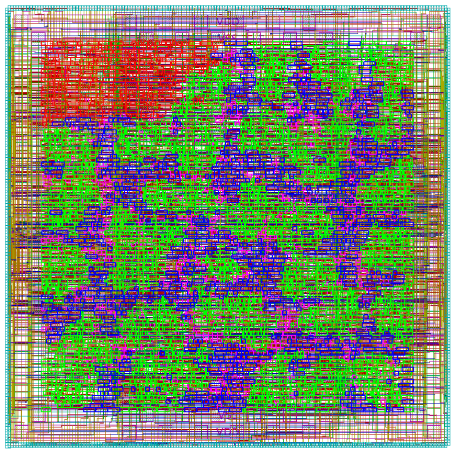

(b) Protected `GIFT` (3SH-ANF), width = 200 μm

Fig. 7. Two of the *placed* and *routed* designs using *Synopsys IC Compiler*. Colors: S-reg (blue), Slayer (green), Key-reg (red), Glue Logic (magenta)

Fig. 8 shows area vs. throughput for all the profiles. It is clear that 3SH approach leads to smaller area, but as it requires an intermediate register, it ends up taking twice the number of clocks. This leads to lower throughput compared to DSH. As can be seen from Fig. 9, DSH using ANF consumes less energy even though it has a significantly larger area than 3SH; this can be attributed to its higher throughput. As a result both the designs can be used depending on application requirements. One can also note that the performance and efficiency of C3SH is not good compared to the other designs, so even though it has the smallest area, using such a design is not recommended.

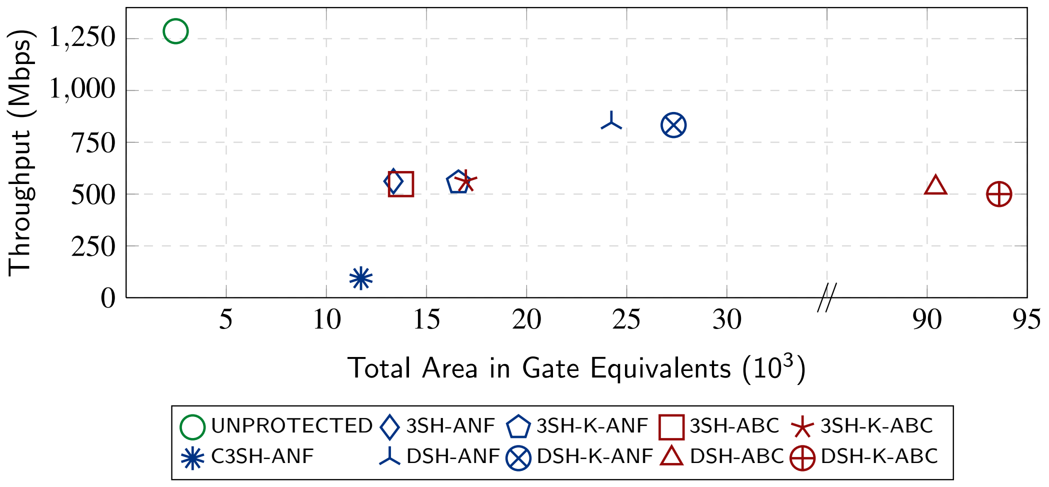


Fig. 8. Area vs. Throughput for all the selected profiles.

IV. Power Analysis

In order to evaluate the security of our design, we implemented the design using HDL and tested it on a SAKURA-X board with a Xilinx Kintex-7 XC7K160T FPGA. The power consumption was measured by probing the voltage drop across the 50 milliohms resistor on the 1V0 FPGA core power line. The output was sampled using a Tektronix MSO4034 at 2.5 Gs/s for unprotected implementation and a Teledyne LeCroy HDO6104A at 5.0 Gs/s @ 12 bits/sample. As the SAKURA-X board is lacking an on-board amplifier we had to use an external preamplifier (Langer 3 Ghz, 30 dB). As `GIFT` is a very small cipher having very small leakage signature, it was important to use a pre-amplifier, without it the leakage was below the noise floor and was hardly discernible. In all the experiments, we were running the cipher cores at 48 MHz. The random bits for the masks were generated using AES-128 in counter mode.

A. *CPA on the unprotected* `GIFT` *cipher*

As mentioned earlier, in this paper we only consider round based hardware implementations (FPGA / ASIC), i.e., for every clock the implementation executes one round or a portion of a round, but, all the plaintext bits and the requisite key bits are processed together. In such implementations, a register is used to store the state and is updated at specific clock events.

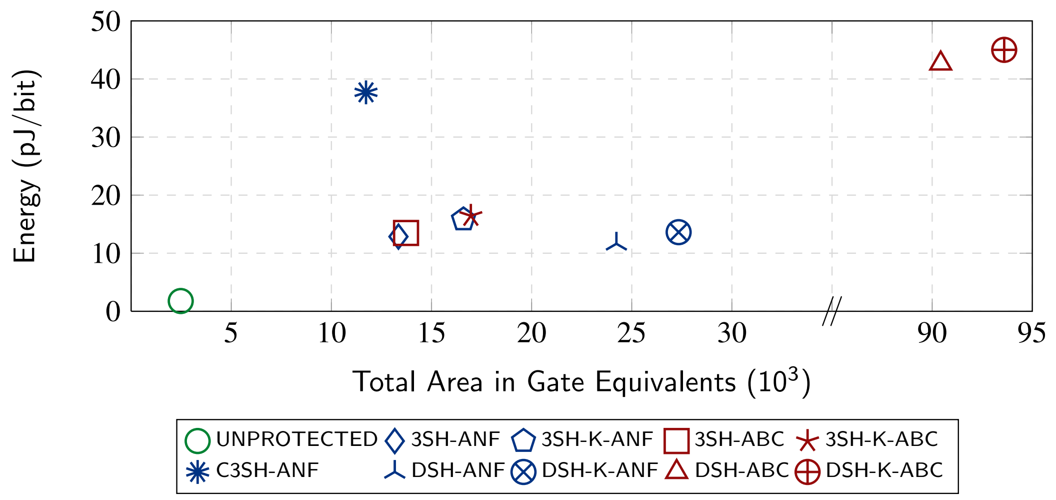


Fig. 9. Area vs. Energy for all the selected profiles.

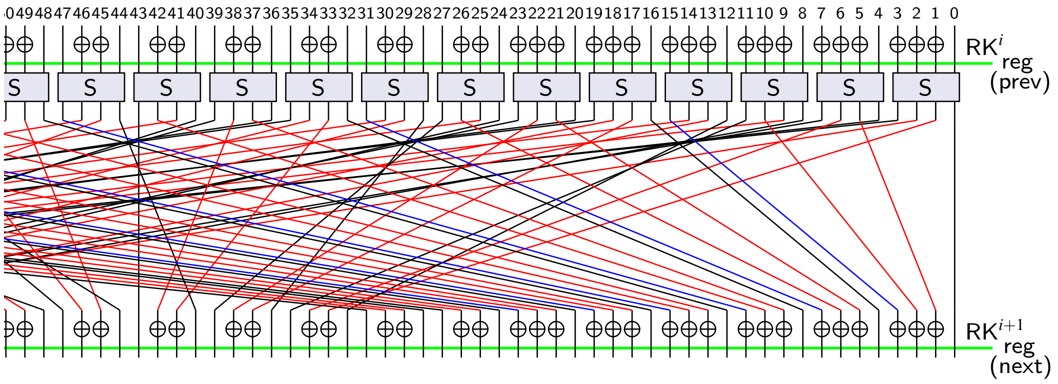


Fig. 10. A portion of the `GIFT-128` round function. S is the `GIFT` S-box and RK_i is the i^{th} round-key. The state registers store the value corresponding to the position represented by the green horizontal lines.

Fig. 10 shows the round function of `GIFT-128`. Assuming an unprotected implementation, the value of the state register is overwritten (updated) at every clock cycle. As a result, the complete cipher execution needs 40 clock cycles (one or two extra clocks may be needed for reading in and out the data, depending on implementation). In such implementations, the leakage follows the Hamming Distance (HD) model as the old data in the state register is overwritten by new data which is calculated by combinatorial circuits.

In Fig. 10, value of the register reg (prev) is over-written by reg (next). Given the bitwise nature of the permutation layer, for leakage modeling we have to consider one S-box at a time and track which bits are permuted to what locations. Unlike `PRESENT`, `GIFT` uses only 64 bits of the round key every round, as a result, for every S-box we can only guess 2 bits; this reduces effectiveness of the CPA attack. Fig. 11 shows two power traces for the reference unprotected implementation of `GIFT`. For this attack we try to focus on the last round and try to recover the key used in the last round. We also assume that the cipher-text is known to the attacker; and all the traces use random plain-texts. Considering the first S-box with input bits 0, 1, 2 and 3, according to the permutation, the output bits go to positions 0, 33, 66, and 99 respectively, and then they are XORed with the corresponding round key bits (33 and 66). Bits 0 and 99 pass through unchanged and are known as we know the ciphertext. Now, if we guess two bits of the key (bit 33 and 66 in this case) we can compute the input of the S-box by computing the *inverse* S-box operation. As reg (prev) is updated by reg (next), we can now have a valid four bit HD estimate based on a guess of two key bits. This can be used as a hypothetical power model. For the ease of implementation we decided to guess 8 bits of the round-key at a time, as a result we had to process 4 S-boxes at a time. In the rest of the paper, guessing a byte of the key refers to guessing 8 bits which can be in different positions at the last XOR, but arise from a set of 4 S-boxes. Fig. 12 shows the correlation values for three guessed key bytes vs *trace points*. Considering CPA for a successful attack, the correct key has the highest correlation value across the trace points. The peak in the figure for key `0x08` corresponds to the time instant at which maximum correlation with leaked key was found. This is the same location of the last round execution as per Fig. 11.

In order to extract the complete round key, we repeated the the above steps for the other 8 bytes and recovered 64 bits. Fig.

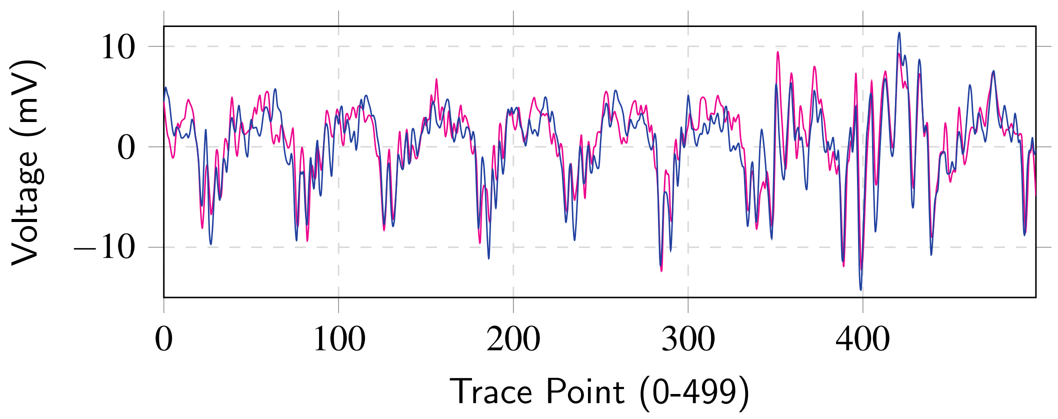


Fig. 11. **Power Trace:** Last 8 rounds of the unprotected `GIFT` implementation.

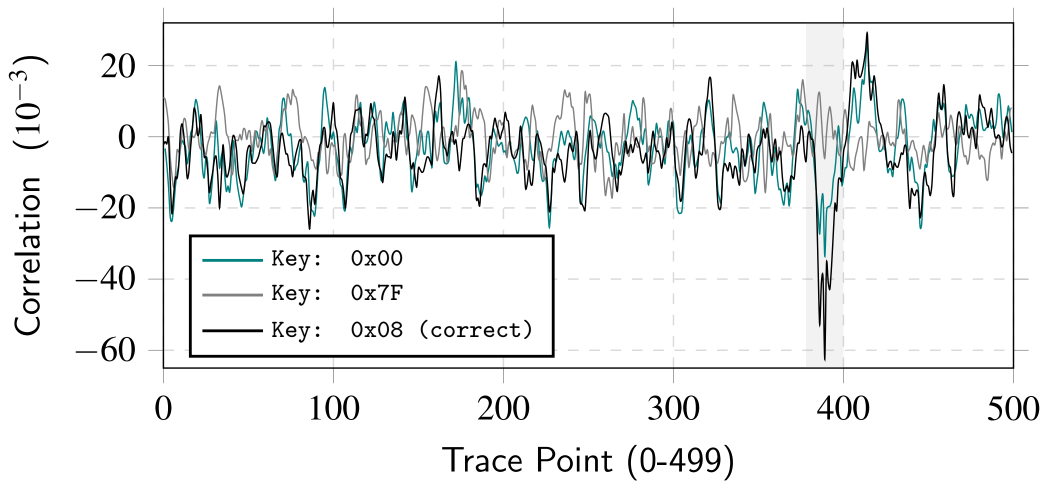


Fig. 12. **Correlation values vs. Trace Point:** For Key Byte 0. A large peak is visible only for the correct key `0x08`.

13 shows correlation values for all guesses for the first 2 bytes of the last round key. In order to recover the complete key, we have to use the fact that we know the last round-key and go one step back and recover rest of the words of the key. This is possible as the key-schedule uses only rotates and no other function.

B. CPA on the protected `GIFT` cipher

The protected implementation as mentioned in the previous section uses two registers and 80 clocks for 40 rounds. Within a round, the first clock is used to evaluate the G function and the second clock computes F, the *permutation*, *Rcon-update* and *key-update*. This causes the two clock to consume different amounts of power; this is quite clear in the *power trace* shown in Fig. 14.

As can be seen from Fig. 15, the protected implementation expectedly does not reveal anything about the key. The highlighted area in the referred figure shows the correct values for the bytes 0 and 5 whereas the peak in the Correlation values suggests wrong value for the same bytes.

It is also clear from Fig. 16 that the protected implementation is secure against first order CPA attacks.

Fig. 17 shows the correlation values over the number of traces for every value of the first key byte. From the figure, one can easily observe that it is not possible to distinguish the correct key byte value from the other hypotheses even after using 3 million traces.

V. Conclusion

In this work, we presented a Correlation Power Analysis attack on the cipher `GIFT`. We also showed that the same attack does not work on a protected implementation of the cipher. We support this claim by analysing 3 million traces collected from a protected FPGA implementation. Furthermore, we performed design analysis over nine different strategies and give trade-off results for area vs throughput (Fig. 8) and area vs energy (Fig. 9). All the required hardware implementation results are reported in Table III. It is interesting to note certain facts from the presented results:

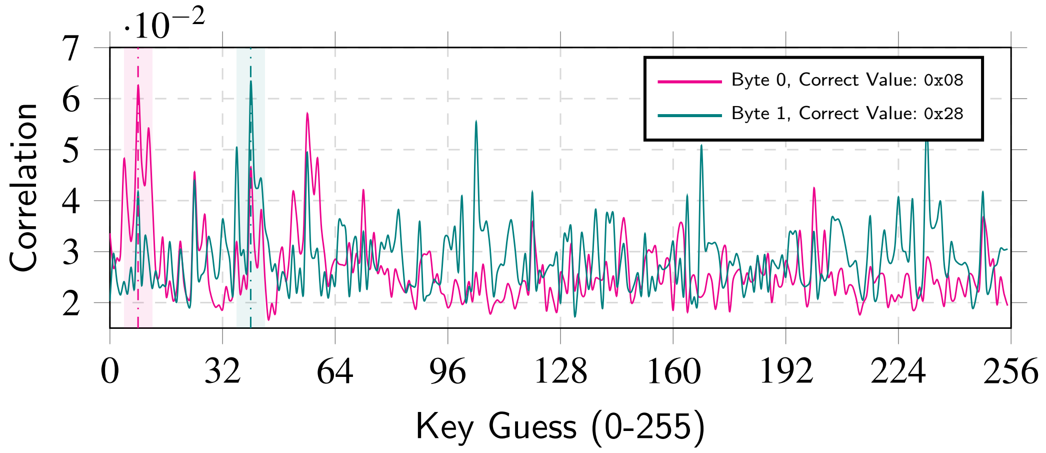


Fig. 13. Correlation values for the two bytes of the key (see text). Peaks at positions 8 and 40 correspond to the correct keys.

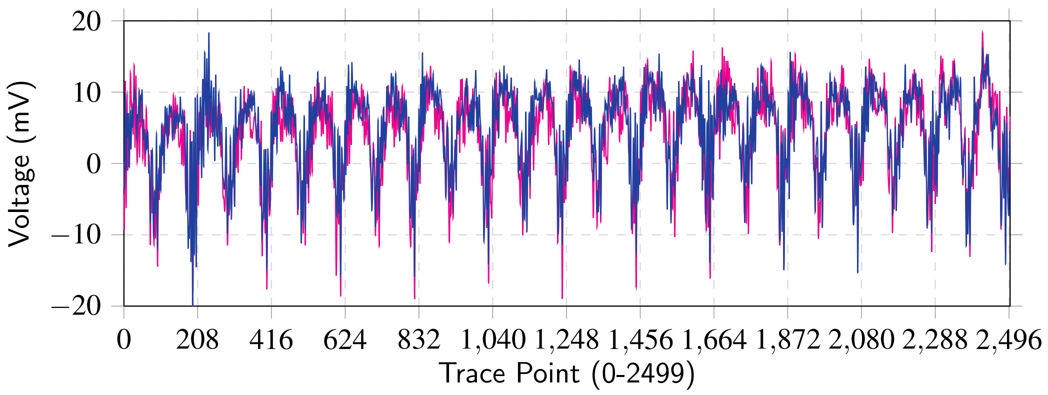


Fig. 14. **Power Trace:** Last 10 rounds of the protected GIFT implementation. The alternating big and small spikes correspond to the *state update* and the *intra S-box register update* respectively.

1) ANF based implementations takes less area, consumes less power and provides higher or similar throughput as compared to the ones using direct Boolean minimization.
2) Even though using combined 3-shares approach for implementation takes lesser area as compared to other approaches using ANF, its throughput as well as energy requirements are much higher. Hence, using such a design is not recommended for round-based implementations as most of the expected reduction in area is nullified by the large multiplexers (this is not a problem in serialized implementations).
3) The energy requirements of implementations using 3-shares and direct-shares is quite similar. The actual comparison between the two arises in terms of throughput and area. It is recommended to use direct-shares approach where higher throughput is required, whereas using 3-shares approach will be a good option in constrained environments with less area and moderate throughput.

In this work we targeted high performance round based implementations, but most of the previous TI implementations focus on serialized implementation to reduce the area. Analyzing such implementations can be a possible future extension.

Appendix A
ANF equations for 3-shares

$G_1(a_2, b_2, c_2, d_2, a_3, b_3, c_3, d_3) = (g_{13}, g_{12}, g_{11}, g_{10})$

$$\begin{aligned}
g_{10} &= a_2 + b_2 + c_2 + d_2 + a_2b_2 + a_2b_3 + a_3b_2\\
g_{11} &= b_2 + a_2c_2 + a_2c_3 + a_3c_2\\
g_{12} &= 1 + c_2\\
g_{13} &= a_2 + b_2 + b_2c_2 + b_2c_3 + b_3c_2
\end{aligned}$$

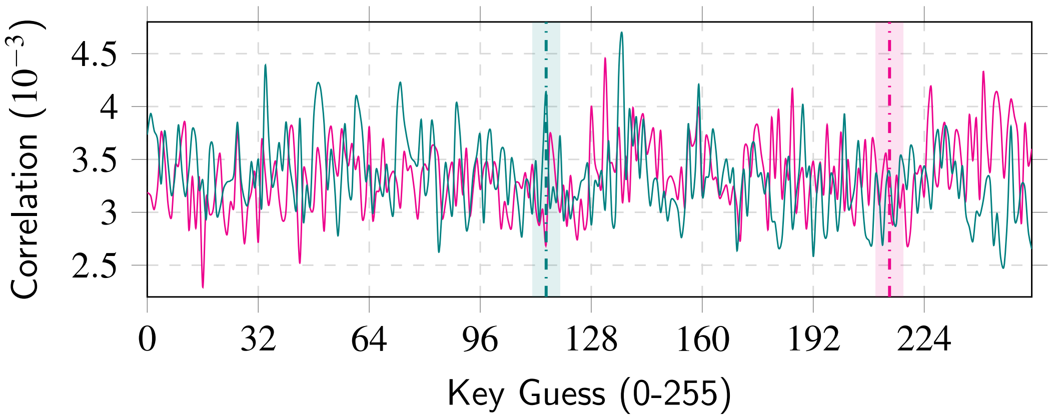


Fig. 15. Correlation values for bytes 0 (magenta) and 5 (plum) of the key. We don't see peaks in the expected locations, showing an unsuccessful attack.

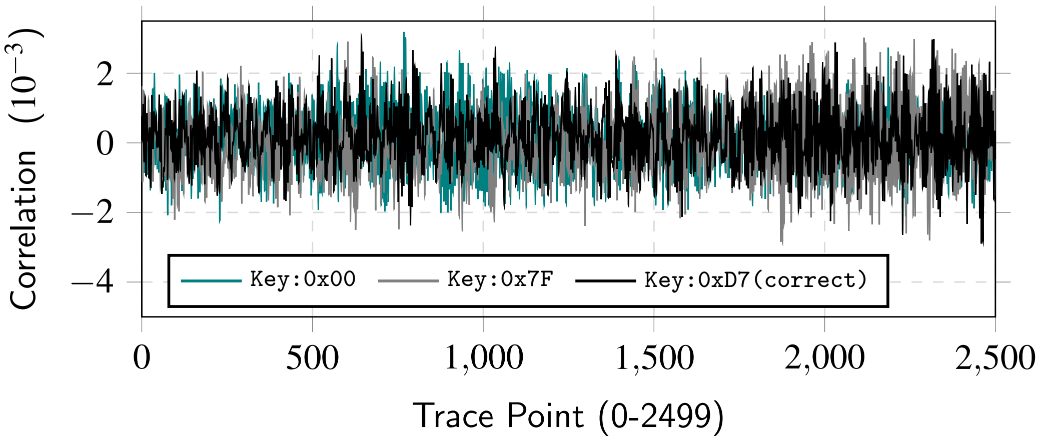


Fig. 16. **Correlation values vs. Trace Point:** The trace corresponding to correct key (`0xD7`) does not have any significant peaks, demonstrating the fact that CPA attack is unsuccessful.

$$G_2(a_1, b_1, c_1, d_1, a_3, b_3, c_3, d_3) = (g_{23}, g_{22}, g_{21}, g_{20})$$

$$\begin{aligned} g_{20} &= a_3 + b_3 + c_3 + d_3 + a_1b_3 + a_3b_1 + a_3b_3 \\ g_{21} &= b_3 + a_1c_3 + a_3c_1 + a_3c_3 \\ g_{22} &= c_3 \\ g_{23} &= a_3 + b_3 + b_1c_3 + b_3c_1 + b_3c_3 \end{aligned}$$

$$G_3(a_1, b_1, c_1, d_1, a_2, b_2, c_2, d_2) = (g_{33}, g_{32}, g_{31}, g_{30})$$

$$\begin{aligned} g_{30} &= a_1 + b_1 + c_1 + d_1 + a_1b_1 + a_1b_2 + a_2b_1 \\ g_{31} &= b_1 + a_1c_1 + a_1c_2 + a_2c_1 \\ g_{32} &= c_1 \\ g_{33} &= a_1 + b_1 + b_1c_1 + b_1c_2 + b_2c_1 \end{aligned}$$

$$F_1(a_2, b_2, c_2, d_2, a_3, b_3, c_3, d_3) = (f_{13}, f_{12}, f_{11}, f_{10})$$

$$\begin{aligned} f_1 &= 1 + a_2 \\ f_{11} &= a_2 + b_2 \\ f_{12} &= 1 + b_2 + c_2 + d_2 + a_2d_2 + a_2d_3 + a_3d_2 \\ f_{13} &= d_2 + a_2b_2 + a_2b_3 + a_3b_2 \end{aligned}$$

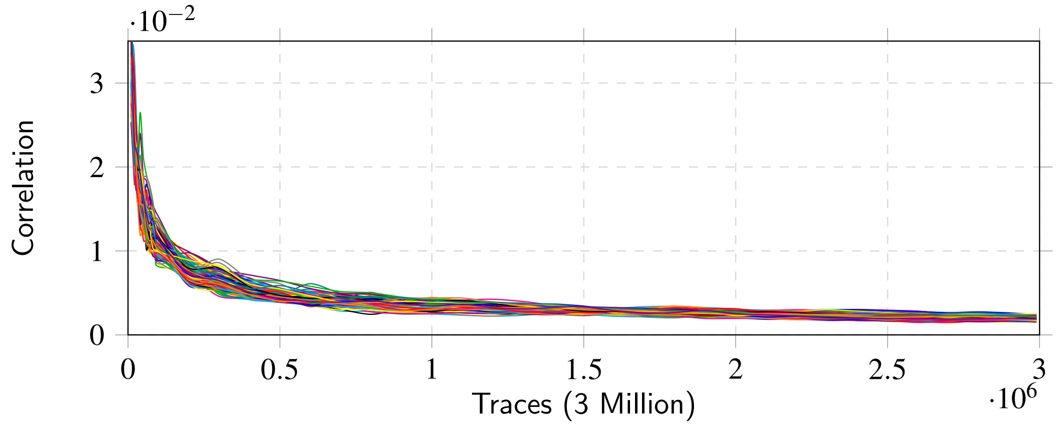


Fig. 17. **Correlation values vs. Traces:** This shows the correlation values for all possible values for the first key byte over the number of traces used.

$$F_2(a_1,b_1,c_1,d_1,a_3,b_3,c_3,d_3) = (f_{23},f_{22},f_{21},f_{20})$$

$$\begin{aligned}
f_{20} &= a_3\\
f_{21} &= a_3 + b_3\\
f_{22} &= b_3 + c_3 + d_3 + a_1d_3 + a_3d_1 + a_3d_3\\
f_{23} &= d_3 + a_1b_3 + a_3b_1 + a_3b_3
\end{aligned}$$

$$F_3(a_1,b_1,c_1,d_1,a_2,b_2,c_2,d_2) = (f_{33},f_{32},f_{31},f_{30})$$

$$\begin{aligned}
f_{30} &= a_1\\
f_{31} &= a_1 + b_1\\
f_{32} &= b_1 + c_1 + d_1 + a_1d_1 + a_1d_2 + a_2d_1\\
f_{33} &= d_1 + a_1b_1 + a_1b_2 + a_2b_1
\end{aligned}$$

Appendix B
ANF equations for direct-shares

$$S_1(a_2,b_2,c_2,d_2,a_3,b_3,c_3,d_3,a_4,b_4,c_4,d_4) = \\ (s_{13},s_{12},s_{11},s_{10})$$

$$\begin{aligned}
s_{10} &= 1 + a_2 + b_2 + c_2 + d_2 + a_2b_2 + a_2b_3 + a_2b_4 + a_4b_3\\
s_{11} &= a_2 + c_2 + d_2 + a_2b_2 + a_2b_3 + a_2b_4 + a_4b_3 + a_2c_2\\
&\quad + a_2c_3 + a_2c_4 + a_4c_3\\
s_{12} &= b_2 + c_2 + a_2d_2 + a_2d_3 + a_2d_4 + b_2d_2 + b_2d_3 + b_2d_4\\
&\quad + a_4d_3 + b_4d_3 + b_2c_2d_2 + b_2c_3d_2 + b_2c_4d_2 + b_3c_4d_2\\
&\quad + b_4c_3d_2 + b_2c_2d_3 + b_2c_3d_3 + b_2c_4d_3 + b_4c_2d_3\\
&\quad + b_4c_3d_3 + b_4c_4d_3 + b_2c_2d_4 + b_2c_3d_4 + b_2c_4d_4\\
&\quad + b_3c_2d_4 + b_4c_3d_4\\
s_{13} &= a_2 + b_2d_2 + b_2d_3 + b_2d_4 + b_4d_3 + a_2c_2d_2 + a_2c_3d_2\\
&\quad + a_2c_4d_2 + a_3c_4d_2 + a_4c_3d_2 + a_2c_2d_3 + a_2c_3d_3\\
&\quad + a_2c_4d_3 + a_4c_2d_3 + a_4c_3d_3 + a_4c_4d_3 + a_2c_2d_4\\
&\quad + a_2c_3d_4 + a_2c_4d_4 + a_3c_2d_4 + a_4c_3d_4
\end{aligned}$$

$$S_2(a_1,b_1,c_1,d_1,a_3,b_3,c_3,d_3,d_3,a_4,b_4,c_4,d_4) = (s_{23},s_{22},s_{21},s_{20})$$

$$\begin{aligned}
s_{20} &= a_3+b_3+c_3+d_3+a_3b_3+a_3b_4+a_3b_1+a_1b_4\\
s_{21} &= a_3+c_3+d_3+a_3b_3+a_3b_4+a_3b_1+a_1b_4+a_3c_3\\
&\quad +a_3c_4+a_3c_1+a_1c_4\\
s_{22} &= b_3+c_3+a_3d_3+a_3d_4+a_3d_1+b_3d_3+b_3d_4+b_3d_1\\
&\quad +a_1d_4+b_1d_4+b_3c_3d_3+b_3c_4d_3+b_3c_1d_3+b_4c_1d_3\\
&\quad +b_1c_4d_3+b_3c_3d_4+b_3c_4d_4+b_3c_1d_4+b_1c_3d_4\\
&\quad +b_1c_4d_4+b_1c_1d_4+b_3c_3d_1+b_3c_4d_1+b_3c_1d_1\\
&\quad +b_4c_3d_1+b_1c_4d_1\\
s_{23} &= a_3+b_3d_3+b_3d_4+b_3d_1+b_1d_4+a_3c_3d_3+a_3c_4d_3\\
&\quad +a_3c_1d_3+a_4c_1d_3+a_1c_4d_3+a_3c_3d_4+a_3c_4d_4\\
&\quad +a_3c_1d_4+a_1c_3d_4+a_1c_4d_4+a_1c_1d_4+a_3c_3d_1\\
&\quad +a_3c_4d_1+a_3c_1d_1+a_4c_3d_1+a_1c_4d_1
\end{aligned}$$

$$S_3(a_1,b_1,c_1,d_1,a_2,b_2,c_2,d_2,a_4,b_4,c_4,d_4) = (s_{33},s_{32},s_{31},s_{30})$$

$$\begin{aligned}
s_{30} &= a_4+b_4+c_4+d_4+a_4b_4+a_4b_1+a_4b_2+a_2b_1\\
s_{31} &= a_4+c_4+d_4+a_4b_4+a_4b_1+a_4b_2+a_2b_1+a_4c_4\\
&\quad +a_4c_1+a_4c_2+a_2c_1\\
s_{32} &= b_4+c_4+a_4d_4+a_4d_1+a_4d_2+b_4d_4+b_4d_1+b_4d_2\\
&\quad +a_2d_1+b_2d_1+b_4c_4d_4+b_4c_1d_4+b_4c_2d_4+b_1c_2d_4\\
&\quad +b_2c_1d_4+b_4c_4d_1+b_4c_1d_1+b_4c_2d_1+b_2c_4d_1\\
&\quad +b_2c_1d_1+b_2c_2d_1+b_4c_4d_2+b_4c_1d_2+b_4c_2d_2\\
&\quad +b_1c_4d_2+b_2c_1d_2\\
s_{33} &= a_4+b_4d_4+b_4d_1+b_4d_2+b_2d_1+a_4c_4d_4+a_4c_1d_4\\
&\quad +a_4c_2d_4+a_1c_2d_4+a_2c_1d_4+a_4c_4d_1+a_4c_1d_1\\
&\quad +a_4c_2d_1+a_2c_4d_1+a_2c_1d_1+a_2c_2d_1+a_4c_4d_2\\
&\quad +a_4c_1d_2+a_4c_2d_2+a_1c_4d_2+a_2c_1d_2
\end{aligned}$$

$$S_4(a_1,b_1,c_1,d_1,a_2,b_2,c_2,d_2,a_3,b_3,c_3,d_3) = (s_{43},s_{42},s_{41},s_{40})$$

$$\begin{aligned}
s_{40} &= a_1+b_1+c_1+d_1+a_1b_1+a_1b_2+a_1b_3+a_3b_2\\
s_{41} &= a_1+c_1+d_1+a_1b_1+a_1b_2+a_1b_3+a_3b_2+a_1c_1\\
&\quad +a_1c_2+a_1c_3+a_3c_2\\
s_{42} &= b_1+c_1+a_1d_1+a_1d_2+a_1d_3+b_1d_1+b_1d_2+b_1d_3\\
&\quad +a_3d_2+b_3d_2+b_1c_1d_1+b_1c_2d_1+b_1c_3d_1+b_2c_3d_1\\
&\quad +b_3c_2d_1+b_1c_1d_2+b_1c_2d_2+b_1c_3d_2+b_3c_1d_2\\
&\quad +b_3c_2d_2+b_3c_3d_2+b_1c_1d_3+b_1c_2d_3+b_1c_3d_3\\
&\quad +b_2c_1d_3+b_3c_2d_3\\
s_{43} &= a_1+b_1d_1+b_1d_2+b_1d_3+b_3d_2+a_1c_1d_1+a_1c_2d_1\\
&\quad +a_1c_3d_1+a_2c_3d_1+a_3c_2d_1+a_1c_1d_2+a_1c_2d_2\\
&\quad +a_1c_3d_2+a_3c_1d_2+a_3c_2d_2+a_3c_3d_2+a_1c_1d_3\\
&\quad +a_1c_2d_3+a_1c_3d_3+a_2c_1d_3+a_3c_2d_3
\end{aligned}$$

References

[1] P. C. Kocher, “Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems,” in *Annual International Cryptology Conference*. Springer, 1996, pp. 104–113.

[2] P. Kocher, J. Jaffe, and B. Jun, “Differential Power Analysis,” in *Advances in cryptology-CRYPTO’99*. Springer, 1999, pp. 789–789.

[3] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer Science & Business Media, 2008, vol. 31.

[4] K. Tiri and I. Verbauwhede, “A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation,” in *Proceedings of the conference on Design, automation and test in Europe-Volume 1*. IEEE Computer Society, 2004, p. 10246.

[5] S. Mangard, T. Popp, and B. M. Gammel, “Side-Channel Leakage of Masked CMOS Gates,” in *CT-RSA*, vol. 3376. Springer, 2005, pp. 351–365.

[6] S. Mangard, N. Pramstaller, and E. Oswald, “Successfully attacking masked AES hardware implementations,” in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2005, pp. 157–171.

[7] S. Nikova, C. Rechberger, and V. Rijmen, “Threshold implementations against side-channel attacks and glitches,” in *International Conference on Information and Communications Security*. Springer, 2006, pp. 529–545.

[8] A. Poschmann, A. Moradi, K. Khoo, C.-W. Lim, H. Wang, and S. Ling, “Side-channel resistant crypto for less than 2,300 GE,” *Journal of Cryptology*, vol. 24, no. 2, pp. 322–345, 2011.

[9] S. Kutzner, P. H. Nguyen, A. Poschmann, and H. Wang, “On 3-share Threshold Implementations for 4-bit S-boxes,” in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2013, pp. 99–113.

[10] B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, and G. Stütz, “Threshold implementations of all 3×3 and 4×4 S-boxes,” in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2012, pp. 76–91.

[11] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov and V. Rijmen, “A more efficient AES threshold implementation,” in *International Conference on Cryptology in Africa*. Springer, 2014, pp. 267–284.

[12] S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, “GIFT: A Small Present,” *Cryptographic Hardware and Embedded Systems–CHES*, pp. 25–28, 2017.

[13] E. Brier, C. Clavier, and F. Olivier, “Correlation power analysis with a leakage model,” in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2004, pp. 16–29.

[14] R. K. Brayton, G. D. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic minimization algorithms for VLSI synthesis*. Springer Science & Business Media, 1984, vol. 2.

[15] G. D. Hachtel and F. Somenzi, *Logic synthesis and verification algorithms*. Springer Science & Business Media, 2006.

[16] J. Hlavička and P. Fišer, “BOOM: a heuristic boolean minimizer,” in *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*. IEEE Press, 2001, pp. 439–442.

[17] R. Brayton and A. Mishchenko, “ABC: An academic industrial-strength verification tool,” in *Computer Aided Verification*. Springer, 2010, pp. 24–40.

[18] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, “PRESENT: An ultra-lightweight block cipher,” in *CHES*, vol. 4727. Springer, 2007, pp. 450–466.

[19] A. Shahverdi, M. Taha, and T. Eisenbarth, “Silent Simon: A threshold implementation under 100 slices,” in *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 1–6.

[20] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen, “Higher-order threshold implementations,” in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2014, pp. 326–343.

[21] C. Carlet, “Vectorial Boolean functions for cryptography,” *Boolean models and methods in mathematics, computer science, and engineering*, vol. 134, pp. 398–469, 2010.

[22] J. Jean, T. Peyrin, and S. M. Sim, “Optimizing Implementations of Lightweight Building Blocks.” *IACR Cryptology ePrint Archive*, vol. 2017, p. 101, 2017.

[23] C. De Canniere, “Analysis and design of symmetric encryption algorithms,” *Doctoral Dissertaion, KULeuven*, 2007.

[24] S. Nikova, V. Rijmen, and M. Schläffer, “Secure hardware implementation of nonlinear functions in the presence of glitches,” *Journal of Cryptology*, vol. 24, no. 2, pp. 292–321, 2011.