Threshold Implementations of GIFT: A Trade-off Analysis

Arpan Jati, Naina Gupta, 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 Sharing (4-shares) is the most straightforward implementation of the threshold countermeasure. However, its usage is limited due to its high area requirements. On the other hand, sharing using decomposition (3-shares) countermeasure for cubic non-linear functions significantly reduces area and complexity in comparison to 4-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. Although, many TI strategies and implementation techniques are known for different ciphers, there is no single study comparing these on a single cipher. Such a study would allow a fair comparison of the various methodologies. In this work, we present 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 one of the most energy-efficient lightweight ciphers. The experimental results show that different implementation techniques have distinct applications. For example, the 4-shares technique is good for applications demanding high throughput 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 TVLA (test vector leakage assessment) on all the design strategies. Experiments using up to 50 million traces show that the designs are protected against first-order attacks.

Index Terms

Side-channel Attack, Threshold Implementation, DPA, CPA, GIFT, TI, Lightweight Cryptography, TVLA

I. Introduction

Implementing secure embedded systems has been a cat-and-mouse game since last few 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 many of these devices have become an important part of our daily lives. The seminal work by Kocher et al. [1], [2] in the late 90's showed 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 intermediate values during computations 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 showed how to apply TI on the PRESENT cipher. Later, in 2013 Kutzner et al. [9] presented the *one S-box for all* technique to efficiently implement 3-shares. Furthermore, [10] describes how to speed-up search for the 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]. However, the design exploration using all these TI methodologies and implementation techniques have not yet been performed for a single cipher on a common platform. 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.

Our contributions. First, we present a Correlation Power Analysis (CPA) [13] attack for an unprotected FPGA implementation of the GIFT cipher in § 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 implement the attack using 4 S-boxes at a time. In our experiments using Xilinx Kintex-7 FPGA, we are able to recover the secret key in less than 10,000 traces. Second, we implement multiple efficient TI countermeasures for GIFT. The implementations are protected against first-order power attacks. We support this claim by performing Test Vector Leakage Assessment methodology (TVLA) [14] using up to 50 Million real power traces on three of the protected

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

implementations. These experiments are described in § IV-B. Third, we implement nine different profiles using known TI techniques and provide a trade-off analysis in terms of area, frequency, latency, power and energy. In particular, we focus on three TI techniques – 3-shares, combined 3-shares and 4-shares using various options. This analysis is presented in § III-B.

There are two common types of implementations for iterated block ciphers, namely the *serialized* and the *round-based*. The *serialized* implementations typically require significantly smaller area and have much reduced throughput; whereas the *round-based* implementations are much larger and have very high throughputs. In this work, we focus on high throughput implementations and hence select only the *round-based* implementations for our analysis.

The Boolean equations corresponding to the individual bits of any non-linear function (in our case S-box) are typically represented using 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 [15], [16], BOOM [17], ABC [18] 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 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. Further, many implementations skip the key-update masking, but it is possible that the *hamming weight* of certain parts of the key is leaked even for very simple key-schedules. Therefore, we also consider key-update masking in our analysis.

Implementation and analysis of the synthesis results for all the TI schemes was done using the same library (TSMC 65nm Low Power). As discussed in § III-C, the 3-shares technique is 45.5% smaller but requires twice the number of clock cycles compared to the 4-shares technique. It is noteworthy to observe that both the designs have very similar overall energy requirements. Further, the combined 3-shares technique can operate at a very high frequency, but the design requires a large number of clock cycles leading to a very low energy efficiency.

II. Preliminaries

A. GIFT Specifications

GIFT is an SPN (substitution-permutation network) based cipher. Its design is strongly influenced by the cipher PRESENT [19]. 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 nibbles of 4-bit represented as $w_{31}, \dots w_1, w_0$. 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).

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

S-layer (S): An S-box is applied to each of the 4-bit nibbles of the state S. The S-box is shown in Table I.

P-layer (P): This operation permutes the bits of the cipher state S from position i to $\mathsf{P}(i)$. The permutation table can be referred from the design document [12].

AddRoundKey: A 64-bit round key RK and a 7-bit round constant Rcon is XORed to a part of the cipher state S in this operation. 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 = $u_{31}, \dots, u_1, u_0||v_{31}, \dots, v_1, v_0$. The two halves of RK, namely U and V, are XORed to the cipher state as follows: $b_{4i+2} \leftarrow b_{4i+2} \oplus u_i$ and $b_{4i+1} \leftarrow b_{4i+1} \oplus v_i$ $\forall i \in \{0, \dots, 31\}$, where b_j denotes the j^{th} bit of the state. The 6-bit round constant c (i.e. $c_5c_4c_3c_2c_1c_0$) 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 is 64/128 depending on the cipher.

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 produced as the ciphertext.

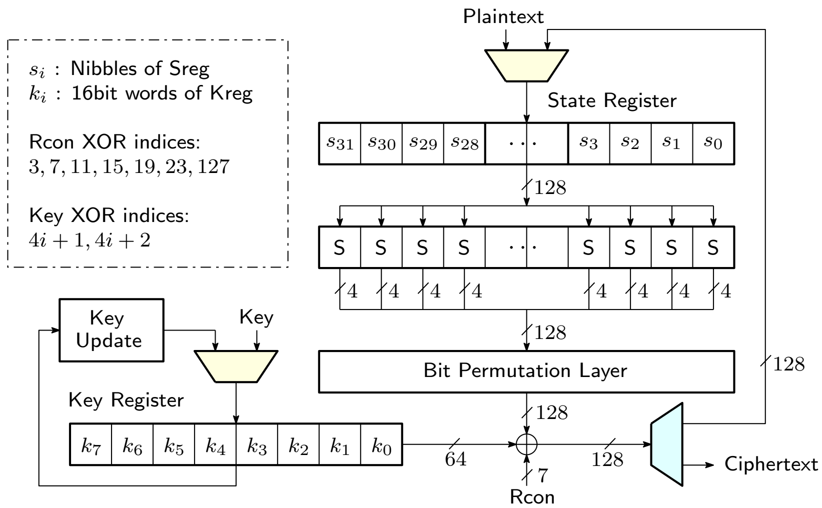


Fig. 1. GIFT Encryption

B. *Threshold Implementation: Requirements*

As mentioned in § I, TI is based on *secret-sharing* and *multi-party* computations. Over the years, TI has received widespread adoption since the technique works even in the presence of *glitches* while certain other countermeasure techniques fail [8], [20], [11], [10], [9]. Initially, TI was proposed to prevent first-order attacks only. But recently, TI has been successfully applied to prevent *Higher Order DPA* attacks as well [21]. TI needs the following three properties to be satisfied:

1) **Correctness:** 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.
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 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 (4-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 use this technique for decomposition. We also use the LIGHTER tool [22] for estimating GE [1] (*gate equivalents*) and use the result as a metric to select the final decomposition. A good GE estimate allows for an efficiently implementable hardware circuit.

Consider 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 can be written for $F(X)$.

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.

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

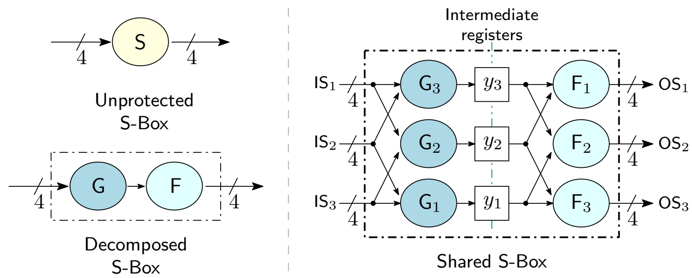


Fig. 2. Sharing using Decomposition (3-shares)

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

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 vectorial Boolean function [23] 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 possible decompositions for `GIFT` S-box after the above mentioned filtering steps.
6) Keep only the $G(X)$ and $F(X)$ combinations which are permutations while discarding 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, use the LIGHTER tool to generate a good estimate in GE.

   Finally, we choose the decomposition with a trade-off between minimum total ANF weight and minimum total GE.

The selected $G(X)$ and $F(X)$ satisfying all the three `TI` requirements, namely Correctness, Non-Completeness and Uniformity, are shown in Table II. The chosen $G(X)$ belongs to quadratic class Q_{293} and the chosen $F(X)$ belongs to the class Q_{294} [24].

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

The ANFs for 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 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 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. This allows us to use only two functions – one for $G(X)$ and another for $F(X)$, instead of using six different (8×4) Boolean functions.

As shown in Fig. 3, two multiplexers are used to choose the input for the Boolean function $G(X)$ 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 in 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. One must note that the intermediate registers g_1, g_2, f_1, and f_2 are required to avoid attacks using glitches.

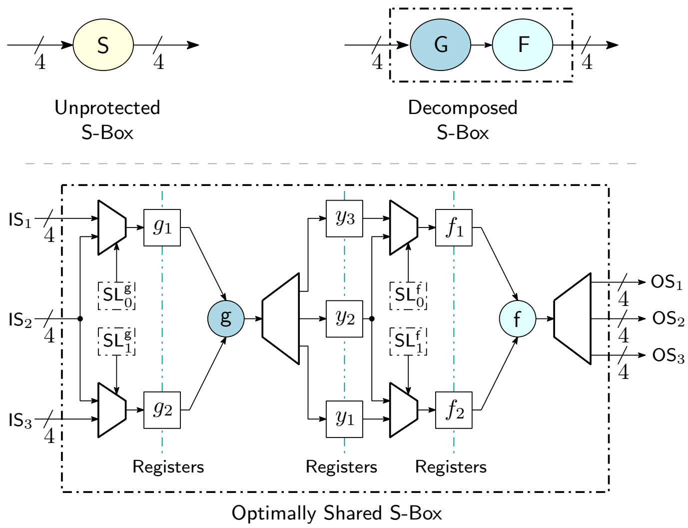


Fig. 3. Sharing using Decomposition (combined 3-shares)

Direct Sharing (4-shares). For TI implementation using 4-shares, one uses 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 [25]. 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:

$$
\begin{aligned}
F_1 &= Z_2 + X_2Y_2 + X_2Y_3 + X_3Y_2 \\
F_2 &= Z_3 + X_1Y_3 + X_3Y_1 + X_3Y_3 \\
F_3 &= Z_1 + X_1Y_1 + X_1Y_2 + X_2Y_1
\end{aligned}
$$

In the case of GIFT, the only non-linear operation is its S-box. The S-box is a 4 × 4 Boolean function (represented as

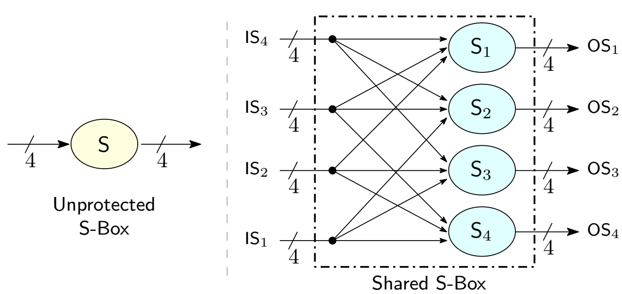


Fig. 4. Direct Sharing (4-shares)

$S(d, c, b, a) \rightarrow (w, z, y, x)$) and has a cubic degree. Hence, we need a minimum of four shares. Fig. 4 shows the approach

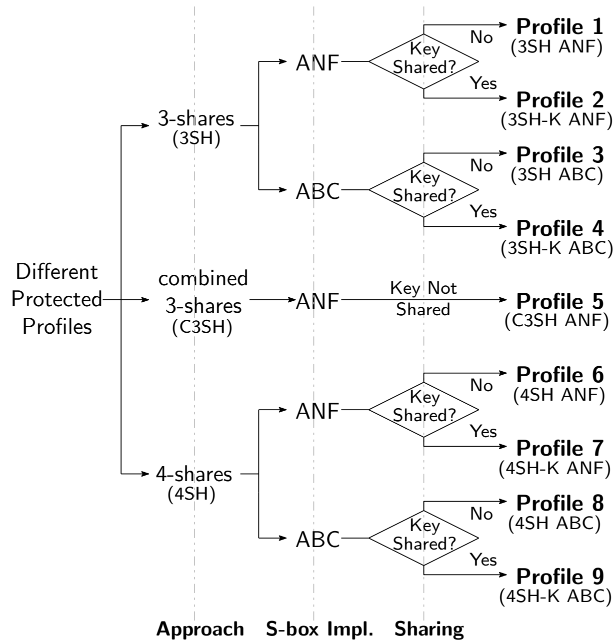


Fig. 5. Different Profiles for Threshold countermeasure

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 ANFs 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

Next we present nine different profiles for threshold implementation of GIFT and discuss the 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, the 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 use the 4-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 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

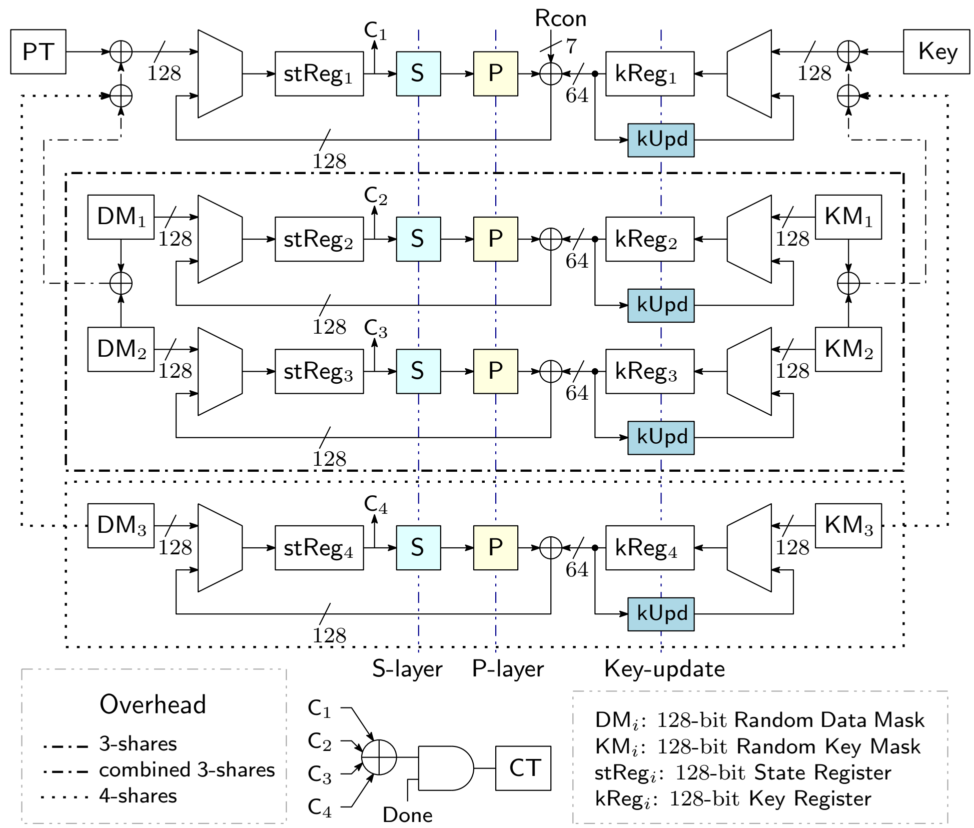


Fig. 6. Overall Architecture for TI techniques for GIFT S-box

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. Therefore 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. *Profiles 1…4* require 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*, the hardware overhead compared to *Profile 1…4* is 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 eight times more clock-cycles compared to the unprotected implementation.

Profile 6…9 use the 4-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

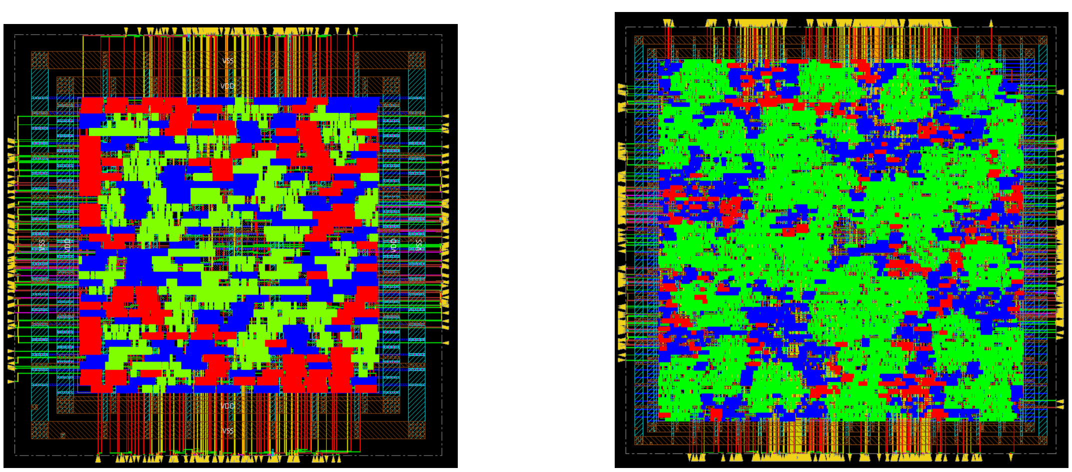
The HDL designs for all of the implementation profiles were written in VHDL. Functional testing was done using the *Xilinx Vivado Simulator* version 2018.1. After functional testing, we used *Synopsys Design Compiler* version J-2014.09 for synthesis of the designs. *Cadence Innovus* version 19.10 was used for placement, routing and power estimates. We used TSMC 65nm Low Power Standard Cell Library (TCBN65LP) for all the ASIC implementations. During synthesis, the compile_ultra command was used to generate an optimized design. Flags to prevent optimization between hierarchical boundaries were used. *Cadence Xcelium Simulator* version 19.3 was used to generate the activity factors from the testbenches. These were used 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

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 ANF	3SH-K ANF	3SH ABC	3SH-K ABC	C3SH ANF	4SH ANF	4SH-K ANF	4SH ABC	4SH-K ABC
S-Box Area (GE)	1020	8716	8931	9686	9872	4317	21731	22283	98074	98029
State Register Area (GE)	916	3847	3918	3732	3723	8129	3807	3794	3700	3717
Key Register Area (GE)	800	1190	3423	1169	3520	809	800	3202	803	3211
Total Area (GE)	3079	**15634**	19312	16388	20119	19912	28709	32722	105530	109035
Ratio	1.000	5.078	6.272	5.323	6.534	6.467	9.324	10.627	34.274	35.412
Frequency (MHz)	857	1215	1194	1148	1210	**2557**	624	641	532	535
# Clock-cycles	40	80	80	80	80	320	40	40	40	40
Output Latency (ns)	46.64	65.84	66.96	69.68	66.08	125.12	64.08	62.40	75.16	74.72
Throughput (Mbps)	2617	1854	1823	1751	1847	975	**1904**	**1956**	1624	1633
Throughput / Area (Kbps / GE)	870.38	**121.44**	96.66	109.46	94.02	50.17	67.94	61.22	15.76	15.34
Total Power (mW)	4.440	22.426	20.351	22.123	29.363	54.538	20.310	25.273	62.302	62.170
Energy (pJ/bit)	1.618	11.535	10.646	12.043	15.159	53.311	10.168	12.321	36.583	36.292
Random bits	0	256	512	256	512	256	384	768	384	768

also important to note that power estimates assume the design running at the highest possible frequency. Running the designs at lower frequency leads to significantly reduced dynamic power consumption. Under such conditions, leakage power can be the primary contributor to overall power consumption. The area and power overheads for the randomness source has not been considered and we assume that the randomness is provided externally. Fig. 7 shows the *placed* and *routed* physical design



(a) Unprotected GIFT, width = 100 μm (b) Protected GIFT (3SH-ANF), width = 200 μm

Fig. 7. Two of the *placed* and *routed* designs using *Cadence Innovus 19.10*. Colors: State register (blue), SBOX (green), Key register (red)

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 3SH is 5.08 times larger. One can see 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

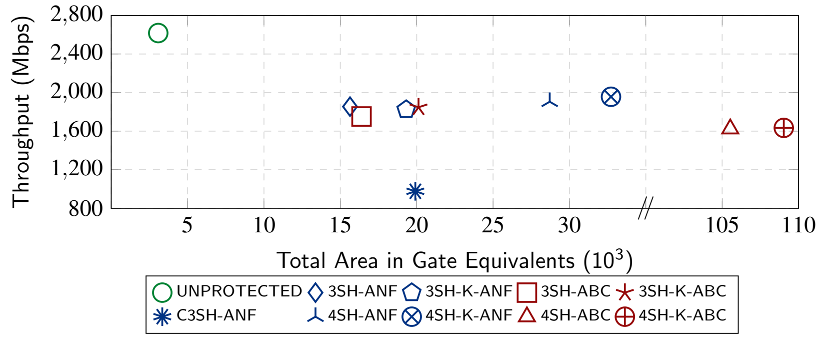


Fig. 8. Area vs. Throughput for all the selected profiles.

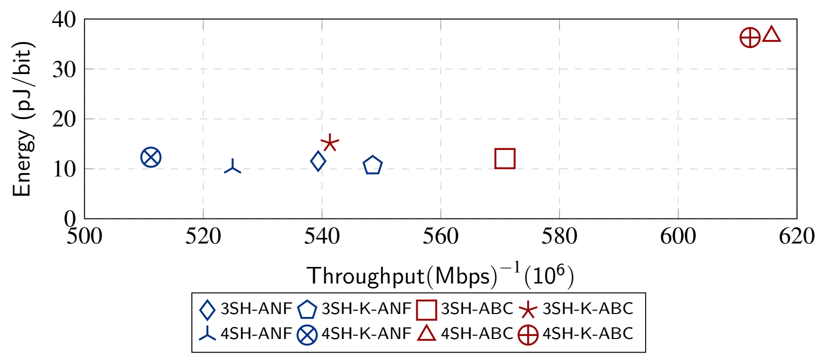


Fig. 9. Energy vs. Throughput^{-1} for the selected profiles (except the outliers).

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 clock cycles and extra intermediate state registers. As the maximum attainable frequency is the highest due to reduced critical path, the power consumption is also quite high.

It is also interesting to contrast ABC based implementation results with ANF ones. For 3SH the difference is small, whereas for 4SH the difference is quite significant (3.67 times). We believe the reason for this difference is the very large size of expressions in case of direct-sharing. The 12×4 mapping in this case leads to about 1100 nodes for one decomposed S-box. Contrasting this with the 8×4 mapping used in 3SH which has about 35-55 nodes depending on the specific $G()$ or $F()$, ABC performs significantly better for the latter. As ABC offers many commands and scripts for Boolean-minimization, even with significant effort, reduction in size for very large networks (above 1000 nodes) was about 10 to 30 percent. From these experiments, it is clear that any additional 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.

Fig. 8 shows area vs. throughput for all the profiles. It is evident that 3SH approach leads to smaller area utilization. However, it ends up taking twice the number of clock cycles as it requires an intermediate register. This leads to lower throughput compared to 4SH.

Let throughput^{-1} represent time per amount of data processed. Fig. 9 presents the variation of energy requirements

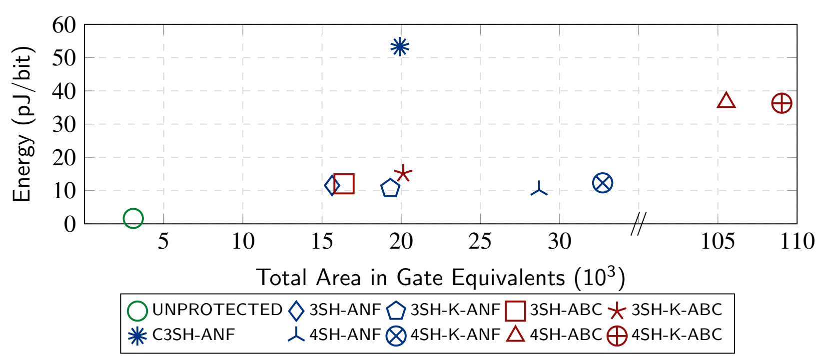


Fig. 10. Area vs. Energy for all the selected profiles.

corresponding to throughput^{-1} for all the profiles. As faster execution time and lower energy consumption is always desired, designs which lie closer to the origin are better. From the figure it is clear that ANF based 4-shared designs are the best in this regard.

As can be seen from Fig. 10, 4SH using ANF consumes comparable energy even though it has a significantly larger area than 3SH. This can be attributed to its higher throughput. Therefore, both the designs can be used depending on application requirements. Even though C3SH can run at the highest frequency, its performance and efficiency is poor compared to the other designs, hence using it is not recommended.

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 1V FPGA core power line. For CPA on the unprotected implementation we used a Tektronix MSO4034 at 2.5 Gs/s and for all TVLA experiments, we used a Teledyne LeCroy HDO6104A at 2.5 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).

Since GIFT has a small leakage signature owing to its small size, using a preamplifier is necessary to bring the signal above the noise floor. 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 (the operation was interleaved with GIFT). Further, we were using an external clock input to synchronize the oscilloscope base clock to the target board.

A. *CPA on the unprotected GIFT cipher*

As mentioned earlier, in this paper we only consider round based hardware implementations (FPGA / ASIC). For such an implementation, a full/part round is executed per clock-cycle in which 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.

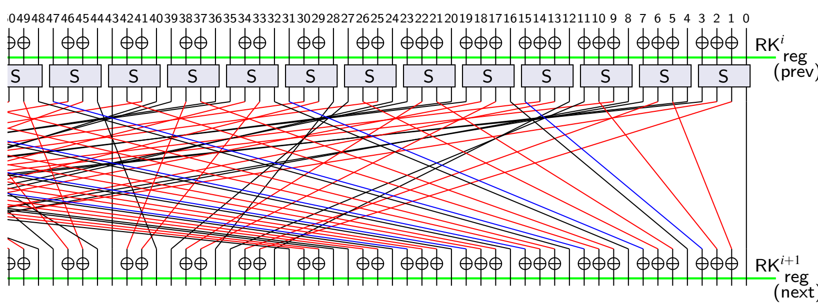


Fig. 11. A portion of the GIFT-128 round function. S is the GIFT S-box and RK_i is the i^{th} round-key. The state registers store the value corresponding to the position represented by the green horizontal lines.

Fig. 11 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 clock cycles 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. 11, value of the register reg (prev) is over-written by reg (next). Given the bitwise nature of the permutation layer, we have to consider one S-box at a time for leakage modeling. Unlike PRESENT, GIFT uses only 64 bits of the round key every round. This leads to only two key bits being used per S-box leading to reduced effectiveness of the CPA attack.

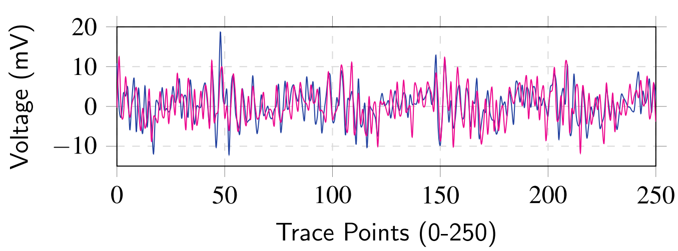
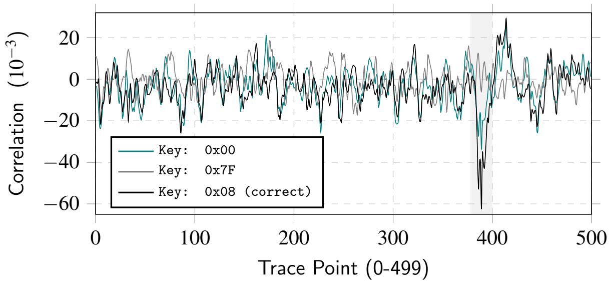


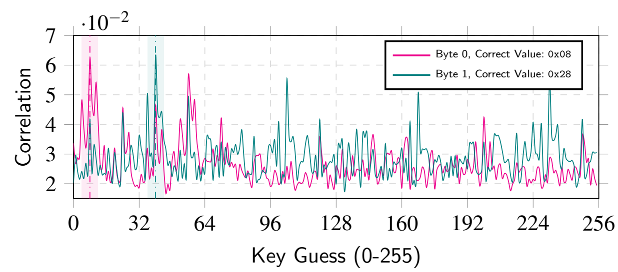
Fig. 12. **Power Trace (zoomed):** Four rounds of the Unprotected Implementation.

Fig. 12 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.

The first S-box takes input bits 0,1,2,3 and the outputs are permuted to positions 0,33,66,99, which are then XORed with the corresponding round key bits (33 and 66). Bits 0 and 99 pass through unchanged and are known since we know the ciphertext. Guessing two bits of the key (bit 33 and 66 in this case) allows us to compute the input of the S-box by inverting the S-box. 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 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 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. 13a 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. 12.



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

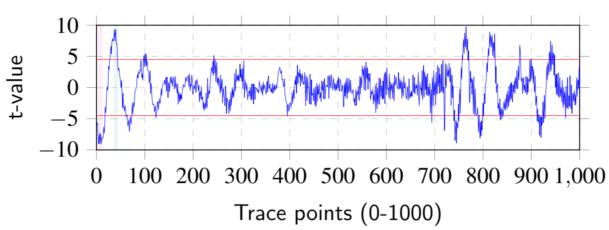


(b) Correlation values for the two bytes of the key (see text). Peaks at positions 8 and 40 correspond to the correct keys.

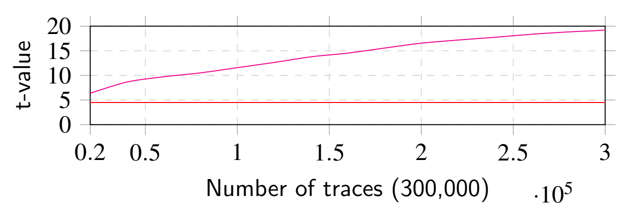
Fig. 13. Correlation for the Unprotected Implementation

In order to extract the complete round key, we repeated the the above steps for the other 8 bytes and recovered 64 bits. Fig. 13b 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. Leakage analysis using TVLA



(a) **Unprotected `GIFT`:** The results are for TVLA after 60,000 traces. The absolute max values are plotted in the incremental chart below.



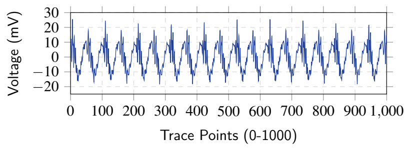
(b) **Unprotected `GIFT`:** Incremental TVLA results (20,000 to 300,000 traces). Even with just 20,000 traces (2 sets), the ± 4.5 threshold is exceeded.

Fig. 14. TVLA on the unprotected `GIFT` implementation

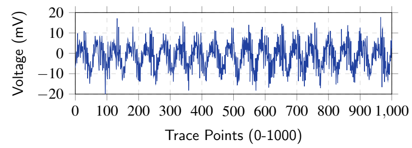
We performed Test Vector Leakage Analysis (TVLA [14]) to evaluate our implementations. More specifically, we used the *non-specific t-test* leakage detection methodology. Our implementation used incremental formulae for TVLA calculations. The experiments were performed in batches of 10,000 traces (set).

In order to test the TVLA trace capture setup and the SNR, we also performed analysis on the unprotected implementation. The results in Fig. 14a show significant leakage. The TVLA threshold of ± 4.5 was exceeded only after the first batch; further batches (Fig. 14b) demonstrate a linearly increasing t-value. The value of ± 4.5 is selected based on [14] (which corresponds to a 99.999% probability that the null hypothesis is false for large number of samples). This demonstrates a sufficiently high SNR (Signal to Noise Ratio) for the experimental setup.

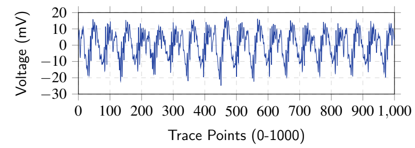
The 3SH implementation as mentioned in the previous section uses two registers and 80 clock cycles for 40 rounds. Within a round, the first clock-cycle is used to evaluate the G function and the second clock-cycle computes F, the *permutation*, *Rcon-update* and *key-update*. This causes the two clock cycles to consume different amounts of power; this is quite clear in the *power trace* shown in Fig. 15a. Incremental TVLA results (Fig. 15g) show that the implementation is secure against first order attacks.



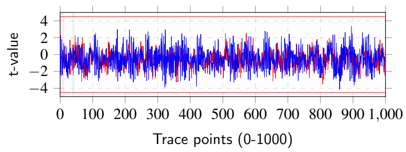
(a) **Protected 3SH:** Power trace for 9.5 rounds of execution. The alternate high-low peaks reveal the two cycles per round implementation.



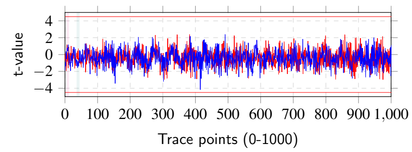
(b) **Protected 4SH:** Power trace for 19 rounds of execution.



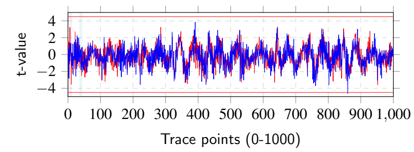
(c) **Protected C3SH:** Power trace for 2.5 rounds of execution.



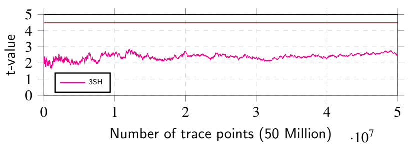
(d) **Protected 3SH:** TVLA after 50 million traces.



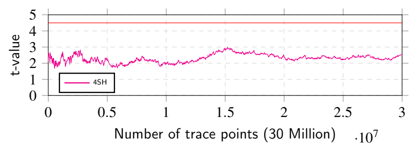
(e) **Protected 4SH:** TVLA after 30 million traces.



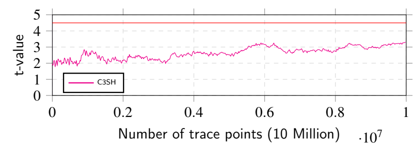
(f) **Protected C3SH:** TVLA after 10 million traces.



(g) **Protected 3SH:** Incremental TVLA results using 50 Million Traces.



(h) **Protected 4SH:** Incremental TVLA results using 30 Million Traces.



(i) **Protected C3SH:** Incremental TVLA results using 10 Million Traces.

Fig. 15. TVLA results for protected implementations

We used 50 Million traces for the experiment to detect the smallest of leakage for the 3SH implementation, which is recommended because of the good throughput and smallest area requirements. Similarly, the results for 4SH and C3SH in Fig. 15 show that the implementations are also resistant against first order attacks.

V. Conclusion

In this work, we present the first Correlation Power Analysis attack on the cipher `GIFT`. To protect against such attacks, we implement the threshold countermeasure. Furthermore, we perform design analysis over nine different strategies and give trade-off results for *area vs throughput* (Fig. 8), *energy vs throughput* (Fig. 9) and *area vs energy* (Fig. 10). Besides, we also perform TVLA on three of the protected implementations and show that they are secure against first-order power attacks. We support this claim by analyzing large sets of power traces collected from the respective protected FPGA implementations.

All the required hardware implementation results are reported in Table III. It is interesting to note certain facts from the presented results:

1) Even though the ANF 3-shares and 4-shares implementations are very different in structure, they are very close in power and energy metrics. The overall energy requirements for the two is comparable as the latter requires only 40 clock-cycles, half compared to the former. So, in a way, a design which consumes more power but finishes earlier can have a lower overall energy consumption.
2) The high throughput/area metric of 121.44 Kbps/GE shows that the ANF 3-shares profile is the overall best design in terms of performance per unit area. This design also has very low energy and power requirements making it suitable for most applications.
3) The 4-shares based design is only useful when low latency is important at the cost of higher area utilization. Because of the longer critical paths the highest frequency is limited compared to the other designs.
4) The combined 3-shares design achieves the highest operating frequency of 2.56 GHz. This is due to reduced critical paths because of multiple registers required around the F() and G() functions. As this design requires 320 clock cycles per round, the overall throughput is significantly lower than other designs. Hence, using such a design is not recommended for round-based implementations as most of the expected reduction in area is nullified by large multiplexers (4907 GE) and extra intermediate registers (this is not a problem in serialized implementations).
5) ANF based implementations take less area, consume less power and provide higher or similar throughput as compared to the ones using any Boolean minimization tools. This is especially true when the network is quite large.

In this work, we have 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

$$
\begin{aligned}
G_1(a_2, b_2, c_2, d_2, a_3, b_3, c_3, d_3) &= (g_{13}, g_{12}, g_{11}, g_{10}) \\
g_{10} &= a_3 + b_3 + c_3 + d_3 + a_2b_2 + a_2b_3 + a_3b_2 \\
g_{11} &= b_3 + a_2c_2 + a_2c_3 + a_3c_2 \\
g_{12} &= 1 + c_3 \\
g_{13} &= a_3 + b_3 + b_2c_2 + b_2c_3 + b_3c_2
\end{aligned}
$$

$$
\begin{aligned}
G_2(a_1, b_1, c_1, d_1, a_3, b_3, c_3, d_3) &= (g_{23}, g_{22}, g_{21}, g_{20}) \\
g_{20} &= a_1 + b_1 + c_1 + d_1 + a_1b_3 + a_3b_1 + a_3b_3 \\
g_{21} &= b_1 + a_1c_3 + a_3c_1 + a_3c_3 \\
g_{22} &= c_1 \\
g_{23} &= a_1 + b_1 + b_1c_3 + b_3c_1 + b_3c_3
\end{aligned}
$$

$$
\begin{aligned}
G_3(a_1, b_1, c_1, d_1, a_2, b_2, c_2, d_2) &= (g_{33}, g_{32}, g_{31}, g_{30}) \\
g_{30} &= a_2 + b_2 + c_2 + d_2 + a_1b_1 + a_1b_2 + a_2b_1 \\
g_{31} &= b_2 + a_1c_1 + a_1c_2 + a_2c_1 \\
g_{32} &= c_2 \\
g_{33} &= a_2 + b_2 + b_1c_1 + b_1c_2 + b_2c_1
\end{aligned}
$$

$$
\begin{aligned}
F_1(a_2, b_2, c_2, d_2, a_3, b_3, c_3, d_3) &= (f_{13}, f_{12}, f_{11}, f_{10}) \\
f_1 &= 1 + a_3 \\
f_{11} &= a_3 + b_3 \\
f_{12} &= 1 + b_3 + c_3 + d_3 + a_2d_2 + a_2d_3 + a_3d_2 \\
f_{13} &= d_3 + a_2b_2 + a_2b_3 + a_3b_2
\end{aligned}
$$

$$
\begin{aligned}
F_2(a_1, b_1, c_1, d_1, a_3, b_3, c_3, d_3) &= (f_{23}, f_{22}, f_{21}, f_{20}) \\
f_{20} &= a_1 \\
f_{21} &= a_1 + b_1 \\
f_{22} &= b_1 + c_1 + d_1 + a_1d_3 + a_3d_1 + a_3d_3 \\
f_{23} &= d_1 + a_1b_3 + a_3b_1 + a_3b_3
\end{aligned}
$$

$$
\begin{aligned}
F_3(a_1, b_1, c_1, d_1, a_2, b_2, c_2, d_2) &= (f_{33}, f_{32}, f_{31}, f_{30}) \\
f_{30} &= a_2 \\
f_{31} &= a_2 + b_2 \\
f_{32} &= b_2 + c_2 + d_2 + a_1d_1 + a_1d_2 + a_2d_1 \\
f_{33} &= d_2 + a_1b_1 + a_1b_2 + a_2b_1
\end{aligned}
$$

Appendix B
ANF equations for 4-shares

$$
\begin{aligned}
S_1(a_2,b_2,c_2,d_2,a_3,b_3,c_3,d_3,a_4,b_4,c_4,d_4) &= (s_{13},s_{12},s_{11},s_{10})\\
s_{10} &= 1+a_2+b_2+c_2+d_2+a_2b_2+a_2b_3+a_2b_4+a_4b_3\\
s_{11} &= a_2+c_2+d_2+a_2b_2+a_2b_3+a_2b_4+a_4b_3+a_2c_2\\
&\quad +a_2c_3+a_2c_4+a_4c_3\\
s_{12} &= b_2+c_2+a_2d_2+a_2d_3+a_2d_4+b_2d_2+b_2d_3+b_2d_4\\
&\quad +a_4d_3+b_4d_3+b_2c_2d_2+b_2c_3d_2+b_2c_4d_2+b_3c_4d_2\\
&\quad +b_4c_3d_2+b_2c_2d_3+b_2c_3d_3+b_2c_4d_3+b_4c_2d_3\\
&\quad +b_4c_3d_3+b_4c_4d_3+b_2c_2d_4+b_2c_3d_4+b_2c_4d_4\\
&\quad +b_3c_2d_4+b_4c_3d_4\\
s_{13} &= a_2+b_2d_2+b_2d_3+b_2d_4+b_4d_3+a_2c_2d_2+a_2c_3d_2\\
&\quad +a_2c_4d_2+a_3c_4d_2+a_4c_3d_2+a_2c_2d_3+a_2c_3d_3\\
&\quad +a_2c_4d_3+a_4c_2d_3+a_4c_3d_3+a_4c_4d_3+a_2c_2d_4\\
&\quad +a_2c_3d_4+a_2c_4d_4+a_3c_2d_4+a_4c_3d_4
\end{aligned}
$$

$$
\begin{aligned}
S_2(a_1,b_1,c_1,d_1,a_3,b_3,c_3,d_3,d_3,a_4,b_4,c_4,d_4) &= (s_{23},s_{22},s_{21},s_{20})\\
s_{20} &= a_3+b_3+c_3+d_3+a_3b_3+a_3b_4+a_3b_1+a_1b_4\\
s_{21} &= a_3+c_3+d_3+a_3b_3+a_3b_4+a_3b_1+a_1b_4+a_3c_3\\
&\quad +a_3c_4+a_3c_1+a_1c_4\\
s_{22} &= b_3+c_3+a_3d_3+a_3d_4+a_3d_1+b_3d_3+b_3d_4+b_3d_1\\
&\quad +a_1d_4+b_1d_4+b_3c_3d_3+b_3c_4d_3+b_3c_1d_3+b_4c_1d_3\\
&\quad +b_1c_4d_3+b_3c_3d_4+b_3c_4d_4+b_3c_1d_4+b_1c_3d_4\\
&\quad +b_1c_4d_4+b_1c_1d_4+b_3c_3d_1+b_3c_4d_1+b_3c_1d_1\\
&\quad +b_4c_3d_1+b_1c_4d_1\\
s_{23} &= a_3+b_3d_3+b_3d_4+b_3d_1+b_1d_4+a_3c_3d_3+a_3c_4d_3\\
&\quad +a_3c_1d_3+a_4c_1d_3+a_1c_4d_3+a_3c_3d_4+a_3c_4d_4\\
&\quad +a_3c_1d_4+a_1c_3d_4+a_1c_4d_4+a_1c_1d_4+a_3c_3d_1\\
&\quad +a_3c_4d_1+a_3c_1d_1+a_4c_3d_1+a_1c_4d_1
\end{aligned}
$$

$$
\begin{aligned}
S_3(a_1,b_1,c_1,d_1,a_2,b_2,c_2,d_2,a_4,b_4,c_4,d_4) &= (s_{33},s_{32},s_{31},s_{30})\\
s_{30} &= a_4+b_4+c_4+d_4+a_4b_4+a_4b_1+a_4b_2+a_2b_1\\
s_{31} &= a_4+c_4+d_4+a_4b_4+a_4b_1+a_4b_2+a_2b_1+a_4c_4\\
&\quad +a_4c_1+a_4c_2+a_2c_1\\
s_{32} &= b_4+c_4+a_4d_4+a_4d_1+a_4d_2+b_4d_4+b_4d_1+b_4d_2\\
&\quad +a_2d_1+b_2d_1+b_4c_4d_4+b_4c_1d_4+b_4c_2d_4+b_1c_2d_4\\
&\quad +b_2c_1d_4+b_4c_4d_1+b_4c_1d_1+b_4c_2d_1+b_2c_4d_1\\
&\quad +b_2c_1d_1+b_2c_2d_1+b_4c_4d_2+b_4c_1d_2+b_4c_2d_2\\
&\quad +b_1c_4d_2+b_2c_1d_2\\
s_{33} &= a_4+b_4d_4+b_4d_1+b_4d_2+b_2d_1+a_4c_4d_4+a_4c_1d_4\\
&\quad +a_4c_2d_4+a_1c_2d_4+a_2c_1d_4+a_4c_4d_1+a_4c_1d_1\\
&\quad +a_4c_2d_1+a_2c_4d_1+a_2c_1d_1+a_2c_2d_1+a_4c_4d_2\\
&\quad +a_4c_1d_2+a_4c_2d_2+a_1c_4d_2+a_2c_1d_2
\end{aligned}
$$

$$
\begin{aligned}
S_4(a_1, b_1, c_1, d_1, a_2, b_2, c_2, d_2, a_3, b_3, c_3, d_3) = (s_{43}, s_{42}, s_{41}, s_{40})&\\
s_{40} &= a_1 + b_1 + c_1 + d_1 + a_1b_1 + a_1b_2 + a_1b_3 + a_3b_2\\
s_{41} &= a_1 + c_1 + d_1 + a_1b_1 + a_1b_2 + a_1b_3 + a_3b_2 + a_1c_1\\
&\quad + a_1c_2 + a_1c_3 + a_3c_2\\
s_{42} &= b_1 + c_1 + a_1d_1 + a_1d_2 + a_1d_3 + b_1d_1 + b_1d_2 + b_1d_3\\
&\quad + a_3d_2 + b_3d_2 + b_1c_1d_1 + b_1c_2d_1 + b_1c_3d_1 + b_2c_3d_1\\
&\quad + b_3c_2d_1 + b_1c_1d_2 + b_1c_2d_2 + b_1c_3d_2 + b_3c_1d_2\\
&\quad + b_3c_2d_2 + b_3c_3d_2 + b_1c_1d_3 + b_1c_2d_3 + b_1c_3d_3\\
&\quad + b_2c_1d_3 + b_3c_2d_3\\
s_{43} &= a_1 + b_1d_1 + b_1d_2 + b_1d_3 + b_3d_2 + a_1c_1d_1 + a_1c_2d_1\\
&\quad + a_1c_3d_1 + a_2c_3d_1 + a_3c_2d_1 + a_1c_1d_2 + a_1c_2d_2\\
&\quad + a_1c_3d_2 + a_3c_1d_2 + a_3c_2d_2 + a_3c_3d_2 + a_1c_1d_3\\
&\quad + a_1c_2d_3 + a_1c_3d_3 + a_2c_1d_3 + a_3c_2d_3
\end{aligned}
$$

References

[1] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Advances in cryptology-CRYPTO'96*. 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 *CHES*. 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 *CHES*. Springer, 2012, pp. 76–91.

[11] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov and V. Rijmen, "A more efficient AES threshold implementation," in *Africacrypt*. 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," *CHES*, pp. 25–28, 2017.

[13] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *CHES*. Springer, 2004, pp. 16–29.

[14] G. Becker, J. Cooper, E. DeMulder, G. Goodwill, J. Jaffe, G. Kenworthy, T. Kouzminov, A. Leiserson, M. Marson, P. Rohatgi *et al.*, "Test vector leakage assessment (TVLA) methodology in practice," in *International Cryptographic Module Conference*, vol. 1001, 2013, p. 13.

[15] 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.

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

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] J. Jean, T. Peyrin, S. M. Sim, and J. Tourteaux, "Optimizing implementations of lightweight building blocks," *IACR Transactions on Symmetric Cryptology*, vol. 2017, no. 4, pp. 130–168, 2017.

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

[24] C. De Canniere, "Analysis and design of symmetric encryption algorithms," *Doctoral Dissertaion, KULeuven*, 2007.

[25] 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.