2017-09-06

Threshold Implementations of the Present Cipher

Mohammad Farmani
Worcester Polytechnic Institute

Threshold Implementations of the Present Cipher

by

Mohammad Farmani

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Electrical and Computer Engineering

July 2017

APPROVED:

Professor Thomas Eisenbarth	Professor Berk Sunar
Major Advisor	Thesis Committee
Professor Alexander Wyglinski	Professor John A. McNeill
Thesis Committee	Department Head

Abstract

The process of securing data has always been a challenge since it is related to the safety of people and society. Nowadays, there are many cryptographic algorithms developed to solve security problems. However, some applications have constraints which make it difficult to achieve high levels of security. Light weight cryptography aims to address this issue while trying to maintain low costs.

Side-channel attacks have changed the way of cryptography significantly. In this kind of attacks, the attacker has physical access to the crypto-system and can extract the sensitive data by monitoring and measuring the side-channels such as power consumption, electromagnetic emanation, timing information, sound, etc. These attacks are based on the relationship between side-channels and secret data. Therefore, there need to be countermeasures to eliminate or reduce side channel leaks or to break the relationship between side-channels and secret data to protect the crypto systems against side-channel attacks.

In this work, we explore the practicality of Threshold Implementation (TI) with only two shares for a smaller design that needs less randomness but is still leakage resistant. We demonstrate the first two-share Threshold Implementations of light-weight block cipher Present. Based on implementation results, two-share TI has a lower area overhead and better throughput when compared with a first-order resistant three-share scheme. Leakage analysis of the developed implementations reveals that two-share TI can retain perfect first-order resistance. However, the analysis also exposes a strong second-order leakage.

Acknowledgements

I would like to express my gratitude first and foremost to my thesis advisor Thomas Eisenbarth for assistance, patience, and support during this research.

This research would not have been possible without the assistance of the Cong Chen for the data analysis. Research results presented in this thesis have resulted in a joint publication with Cong Chen [11].

Finally, I would like to extend my deepest gratitude to my parents Abdollah Farmani and Azam Meskarian; my brothers Mojtaba Farmani and Ali Farmani without whose love, understanding and support I could never have completed this Master's degree.

This research was supported in part by the National Science Foundation under Grant No. 1261399.

Contents

Chapter 1

Introduction

1.1 Motivation

Making cryptographic algorithms secure against side channel analysis usually needs a significant area overheads. Moreover, some of the methods are complex when we want to apply them on a hardware level. Hence, it may make our countermeasure partially insecure [12, 21, 30].

Regarding this issue, there is scheme, Threshold Implementation (TI), that has become popular nowadays due to its outstanding features. Unlike other schemes [41, 21], TI is straightforward and does not need a change in design flow which allows it to be easily applied to a wide range of ciphers while reducing implementation error. An additional advantage of TI is that the implementation is completely reliable against first order side-channel attacks and allows for protection against higher order side-channel attacks [31, 6].

A drawback to TI is, like most other masking schemes, it causes large area and time overheads, often requiring huge amounts of randomness for remasking, making practical applications difficult. The considerable increase in area overhead and the requirement for a high-performance random number generator makes TI an option too expensive for a wide range of applications. TI has been generalized by Reparaz *et al.* to provide protection against higher-order attacks, and reducing the number of shares to $d + 1$, where d is the desired protection order. Thus, it is sufficient to use two shares for first-order protection. De Cnudde *et al.* has done the first evaluation of $d+1$ shares for an AES cipher [15].

1.2 Our Contribution

Although the practicality of the first order resistant three share TI to light weight ciphers has already been explored in [35] [14], we develop the two share Threshold Implementation (2-TI) on Present cipher [33] and explore the practical implications for reducing the number of required shares from three to two for first-order resistance. The result of 2-TI reduces the area overhead and randomness by a factor of three to two. Hence, this reduction increases the relevance of TI for a wider range of applications for side channel protection.

The algebraic depth of the nonlinear functions of lightweight ciphers is low which allows for the suitable implementation of TI. This feature results in cheap and efficient masking and minimizes the need for additional randomness. Additionally, our design does not require re-masking during the round functions. This implementation significantly reduces the amount of required randomness in comparison to other masked implementations such as AES which during one block encryption requires more than 8,000 new random bits [14].

Applying 2-TI on nonlinear functions (S-box) is more difficult and usually needs some pipeline stages which has adverse effects on implementation size and latency. Furthermore, Cong Chen's analysis [11] reveals a strong second-order leakage both theoretically and practically.

1.3 Thesis Outline

In Chapter 2 we introduce the relevant terminologies and methods on cryptography algorithms, side-channel attacks, and ways of protecting against them, we also describe a lightweight cipher, namely Present in more detail. Then we introduce the theoretical discussion of two-share TI in Chapter 3. The protected version of Present is introduced in Chapter 4. We show the implementation results and analysis in Chapter 5 and conclude the work in Chapter 6.

Chapter 2

Background

In the upcoming era of computing, smart devices will have limitations regarding memory, computing power and battery supply as well as bandwidth and vulnerability to attacks [17, 27]. For example, as one of the applications Internet of Things (IoT) has brought us many benefits but also raises problems like security and privacy [1]. On the other hand, the tradeoffs between performance, security, and cost are highly important [27]. Due to these constraints, there are lots of studies trying to implement lightweight cryptography (LWC) in their applications [33, 26, 19] to find the best compromise between security, power consumption, high performance and low footprint. In the last years, several lightweight block ciphers have been proposed including PRIDE, PRESENT, CLEFIA, PRINCE, KLEIN, SIMON and SPECK [1].

Area minimal implementations of cryptography are highly enviable for vast group of embedded systems. Hence, many area efficient crypto cores are proposed which most of them are based on lightweight block cipher designs, like Present, Simon, Speck, or Katan. One common feature that is most common among are efficient hardware is Serialization, for example instead of doing the same tasks in parallel; we can use one of the tasks and try to apply it to inputs in different clock cycles in the serial which we call it Serialization. This feature can reduce the area implementations at the cost of increased run time. To reach the same output, area-critical functions are divided into sub-functions that can be applied repetitively. For instance, in block ciphers, S-box layer usually is difficult to minimize in hardware due to high nonlinearity. A typical area-optimized of an S-box based cipher uses only one S-box. Then, the S-box is applied to the different parts of the inter-

mediate state consecutively. For example if we have 64-bit intermediate state and a 4 bit S-box, the S-box should be applied 16 times to cover all of the 64 bits of the intermediate state. This vertical type of serialization is supported by almost all state-of-the-art block ciphers by using one S-box (unlike DES that uses 8 different S-boxes). To reduce the size of large S-boxes, other techniques are applied (generally for more complex algebraic functions), by dividing them into sub-functions that are linked together. In [34] Canright proposes the implementations that compute the AES S-box by taking advantage of tower field representation. Also, In [34] shows the implementation of the Present S-box into mappings of algebraic degree 2, which alleviates the side-channel protection and reduce the size at the expense of doubling the computation time. This type of serialization is referred as a horizontal type which is determined by the algebraic complexity of the nonlinear layer, while the vertical type of serialization is determined by the cipher at implementation time that is usually determined by the number of S-boxes. The parameters for area efficient hardware implementations of typical vertical serialization range from data path sizes of 1 bit for Simon or Katan, 4 bit for Present, up to 8 bit for AES. That means for AES, Present and Simon or Katan, 8 bit, 4 bit and 1 bit of the cipher state are updated per cycle, respectively. Although serial data paths decrease the combinational logic of the crypto core to low single-digit percentages of the entire design [37, 13], they also increase the latency of the crypto core significantly. In other words, in applications where the latency is not critical, the registers storing the key and state almost determine the area of a cipher. Hence, we can obtain a considerable area efficiency by breaking the memory restraint, for instance by hiding state and key in dedicated memory such as block RAMs [20] or shift registers [2] for FPGAs, or by externalizing key storage [13]. Since the remainder of the work uses Present for proof-of-concept implementations, we provide more details on this cipher here in this chapter after brief explaining about ciphers. Ciphers are divided into to major groups, symmetric and asymmetric.

2.1 Symmetric Key Cryptography

In symmetric key cryptography (also called secret-key or shared key cryptography) the sender and receiver share a common key for both encryption and decryption [26]. If the key is compromised, the attacker can easily decrypt it.

The advantage of this type of cryptography is its faster service without using many resources [17]. The symmetric key encryption happens in two modes - block cipher and stream cipher [42]. In the block cipher mode, the data is divided into some blocks while in stream cipher the data is divided as small as single bits and the encryption takes place after randomizing it [42]. Examples of symmetric key encryption technics are DES Algorithm, Triple DES algorithm, AES algorithm, Present algorithm, and Blowfish algorithm [29].

Data Encryption Standard (DES) was developed by IBM which is a symmetric key algorithm. DES always operates on blocks of equal size of 64 bit and a key size of 56 bits. It uses both permutations and substitutions in the algorithm. Former studies have shown that DES was no longer invulnerable to the attacks [23].

The operation of Triple DES is similar to DES. It uses three 64-bit keys and the procedure for encryption is the same as DES, but the process is repeated three times. In this process it is encrypted with the first key, decrypted with the second key and finally encrypted with the third one [23].

DESL and DESXL are other forms of improved DES algorithm. Substitution boxes (S-boxes) can take up 32% of the area in DES implementations. One can decrease the gate complexity of DES by replacing the eight original S-boxes with a single new one. This way seven S-boxes as well the multiplexer will be eliminated from the system. The lightweight DES variant is called DESL, and the chip is 20% smaller than DES. Its design enables DESL to resist common attacks such as linear and differential and the Davies-Murphy attack [17]. If key whitening is applied to the cipher, it results in DESXL cipher, and the security level reaches 118 bits.

2.1.1 Present

Present is a block cipher proposed in 2007, optimized for low area overhead [8]. It is a substitution-permutation network featuring a 4×4 bit S-box and a permutation layer consisting only of bit shifts, making it low cost in hardware. It features a block size of 64 bits and a key size of 80 or 128 bits and has 31 rounds. Present has been optimized for many application scenarios, but the area-minimal implementations with a 4-bit data-path. It has also been standardized as a lightweight cryptographic block cipher as ISO/IEC 29192-2:2012. Each round of Present cipher consists of three steps including a key-addition layer, a substitution layer which is a non-linear function, and a permutation layer. The general view of the Present cipher algorithm is

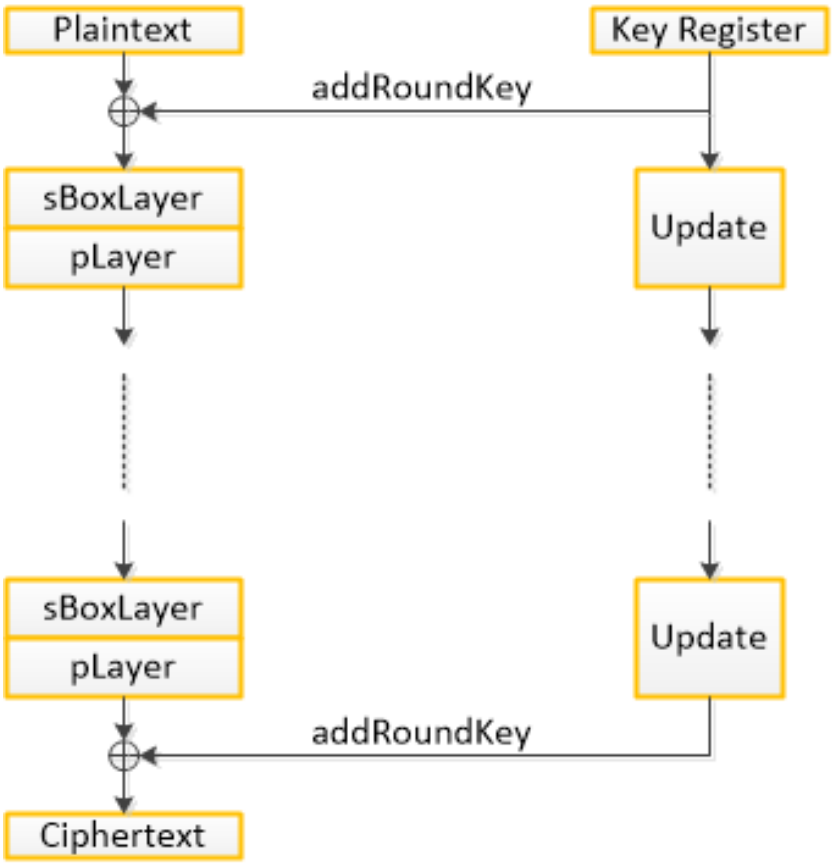


Figure 2.1: General view of Present algorithm

shown in Figure 2.1. Here under different layers of Present cipher has been described:

addRoundKey In each round, the following operation is being done with the operands of current state $s_{63}s_{62}s_0$ and round key $K_i = k^i_{63}k^i_{62}k^i_0$ for $1 \leq i \leq 32$:

$$s_j = s_j \oplus k^i_j \quad \text{for} \quad 0 \leq j \leq 63$$

sBoxlayer The S-box in Present is a non-linear 4-bit to 4-bit function S: $F_2^4 \rightarrow F_2^4$ shown in the following table in hexadecimal notation.

The substitution layer can be performed with 16 parallel S-box or use only one S-box 16 times which depends on the application requirement.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

pLayer The permutation is straightforward. It rewires all of the 64-bit data based on the following table. Every bit i of the STATE is replaced with a bit of $P(i)$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

The key schedule The key can be 80-bit or 120-bit which is updated in the key schedule part. The key can be 80-bit or 120-bit, although we use an 80-bit key in this work. In addRoundKey, the 64 left most bits of the current key, $k_{79}k_{78}k_{77}\ldots k_{17}k_{16}$ is used for each round. After using the round key, the 80-bit key register K= $k_{79}k_{78}\ldots k_0$ is updated by shifting, using S-box, and xoring with round-counter as follows. The key register is rotated to the left by 61 bit positions, then the S-box is applied to the first four bits of the key from the left. The round-counter value of round i is exclusive-ored with five specific bits of K, $k_{19}k_{18}k_{17}k_{16}k_{15}$.

2.2 Asymmetric Key Cryptography

In this technique, different keys are used for encryption and decryption. One key is public (published) while the second one is private. Public key methods are substantial since they can send encryption keys or other data securely even when both users cannot agree on a secret key in private algorithm [23].

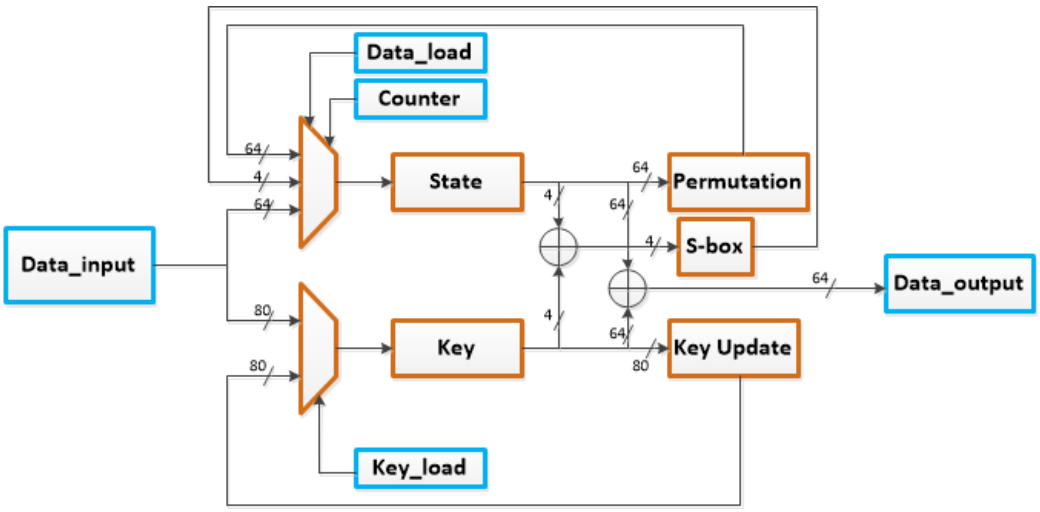


Figure 2.2: Hardware architecture of the Present Cipher

Among public-key algorithms, there are three established families: ECC, RSA (Rivest-Shamir-Adleman) and discrete algorithms [17]. ECC is considered to be the most attractive algorithm due to its smaller operand length and lower computational requirements [17]. The most commonly used public key encryption is RSA algorithm which can be utilized both for encryption and digital signature purposes [23]. The key size in this algorithm should be greater than 1024 bits for a reasonable level of security [23].

2.3 Physical Attacks

Cryptographic algorithms are usually made to resist to algebraic cryptanalysis. However, most of them do not cover physical attacks which can be divided into two classes: Passive attacks and active attacks. Active attacks analyze the chip at the logic level and disturb the operation or reverse-engineer functions. Passive attacks which are also known as side channel attacks (SCA) can record the acoustic, power or electromagnetic emanation while a cryptographic core is running on the device.

2.3.1 Side-channel Attacks

When Kocher et al. [22] published their study on differential power analysis (DPA), it was publicly known that if one analyzes a power traces, which is obtained when a cryptographic primitive is running, the information can be revealed. After a few years, correlation power analysis (CPA) was adopted over DPA due to the higher efficiency and also fewer traces needed [9]. In DPA, the idea is to recover part of the secret key by targeting an intermediate state of the algorithm and trying to predict its value by making hypotheses on the portion of the key involved [1]. Then, one should try to uncover the link between the predictions and the traces using the Pearson correlation coefficient between these two variables. Usually, an appropriate leakage model based on the Hamming weight or the Hamming distance is used [1].

The general idea behind the attacks which are based on the power consumption is explained in [38] in detail. When the Crypto Device consumes power, the current through the device will change. This current can be calculated by measuring the voltage of resistance that is serialized with the crypto device, V_0, and dividing its value by its resistance,R. The power consumption of the device can be calculated using:

$$Power(CryptoDevice) = V_{cc} \times \frac{V_o}{R} \tag{2.1}$$

As can be observed in Equation (2.1), the power consumption of a device is proportional to the V_0. One oscilloscope records the value of V_0 and this value would be proportional to the power consumption of the device. Two types of attacks using power consumption are Simple power analysis and differential power analysis.

Simple Power Analysis (SPA)

Simple power analysis (SPA) is a method that involves direct interpretation of power consumption measurements which are collected during cryptographic operations [22]. A trace is a set of power consumption measurements that are taken across a cryptographic operation. Different power measurement on the trace is due to different instruction operation during the cryptographic algorithm. For example, in Figure 2.3 an SPA trace is shown from a typical smart card as it performs a DES operation [22]. The sixteen rounds can be seen clearly.

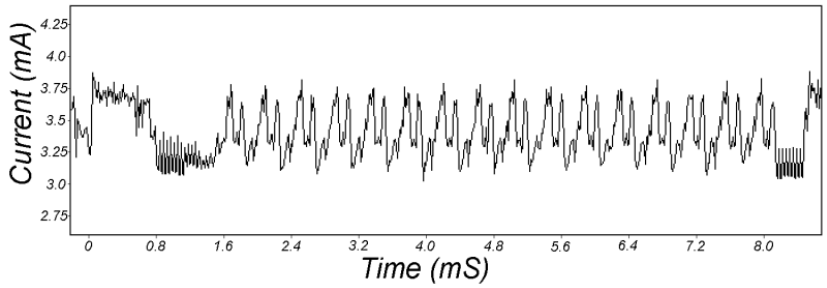


Figure 2.3: SPA trace showing an entire DES operation [22]

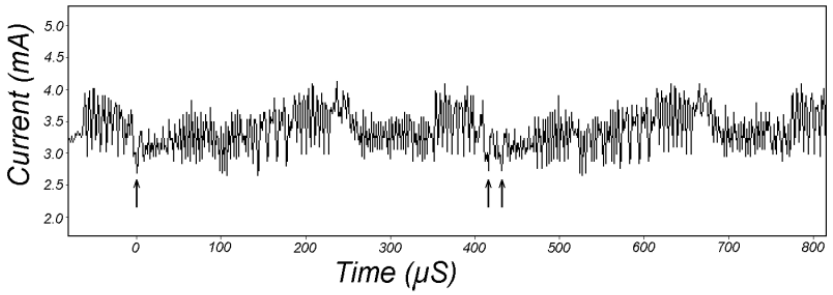


Figure 2.4: An SPA trace showing round 2 and 3 [22]

Figure 2.4 is a more detailed view of the same trace showing the second and third rounds out of the 16 DES rounds [22]. More details are visible. For instance, left arrow shows one rotation of 28-bit DES key registers C and D in round two, and right arrows show two rotation of them in round 3. Since the sequence of instructions execution can be revealed by SPA and the execution path has a direct relation to the processed data, the SPA can be used to break the cryptographic implementation.

In simple power analysis type of attack we assume that the attacker has access to one or a few measurements and to have a successful attack, he should also know the details of the implementation.

Differential Power Analysis (DPA)

Differential Power Analysis Attacks do not need detailed information about the attacked crypto device. They require more power traces contrary to SPA attacks. Hence, it is required to physically access a crypto device to apply

DPA attack on it. Differential Power Analysis attack is a statistical method for analyzing sets of measurements to identify data-dependent correlations. It involves partitioning a set of traces into subsets and then computing the difference between the averages of these subsets. If there is a correlation between subsets and the trace measurements, the average will approach a non-zero value. If enough traces are available, even small correlations can be isolated.

They use enough power traces based on in order to analyze the power consumption at fixed moments of time as a function of the processed data [28]. For DPA attack, we should consider the following steps:

1. Choosing an Intermediate Point of the Algorithm to Attack: In this step, we choose a point of the intermediate result of the cryptographic algorithm executed in the crypto device, which depends on a data d and a part of key k. In such attacks, d can be a plaintext or ciphertext.

2. Power Consumption Measurement: The next step of the DPA is to measure the power consumption of the crypto device while the cryptographic algorithm is being executed D times on the device. For each execution, the attacker should know the corresponding data d_i, $d = (d_1, d_2, \ldots, d_D)$. For each execution, the attacker measure the power consumption related to each d_i as $t_i = (t_{i,1}, \ldots, t_{i,T})$. The number of samples in each trace is denoted by T. Hence, the measured traces corresponding to each data d_i can be denoted in a matrix $\mathbf{T}$ of size $D \times T$.

3. Simulation of Hypothetical Intermediate Value: Here, for each data d_i and every possible value of k, we find the hypothetical intermediate values. Therefore, the calculations conclude in a matrix $\mathbf{V}$ of size $D \times K$, where:

$$v_{i,j} = f(d_i, k_j) \qquad i = 1, \ldots, D \qquad j = 1, \ldots, K \tag{2.2}$$

4. Modeling the Power Consumption Values of Intermediate Values: In this step, we want to obtain the hypothetical power consumption values from intermediate values. In this regard, there are different models that we can use to get the hypothetical power consumption values such as Least Significant Bit (LSB), Most Significant Bit (MSB), Hamming Weight (HW), and Hamming Distance (HD). LSB and MSB consider the right most

bit and left most bit respectively. Hamming Weight model cares about the number of bits with value 1 in the result. Hamming Distance model takes into account the number of transitions of bits that occur from one state to another. The elements of matrix V, $v_{i,j}$, are mapped to elements $h_{i,j}$ to form the matrix H:

$$h_{i,j} = Power - model(v_{i,j})$$

The knowledge of the attacker about the target device has a great impact on the effectiveness of the attack. If the power model matches the actual power traces more precisely, the attack has a better result at the end. Hamming Distance and Hamming Weight models are two most famous power models.

5. Comparing the Power Models with Actual Power Traces: In this step, the power model matrix, H, and the actual power trace matrix,T, are compared together. Each actual power trace should compare with the corresponding hypothetical power model related to all the possible keys. In other words, each column t_j of the matrix T is compared with each column $h_{i,j}$ of the matrix H. The result of comparison is a matrix R of size $K \times T$ that its elements $r_{i,j}$ shows the comparison result of h_i and t_j. The indices of matrix H with the highest value reveal two important information. First, the position where the intermediate result is being processed. Second, information about the actual key that is used in the algorithm on a crypto device.

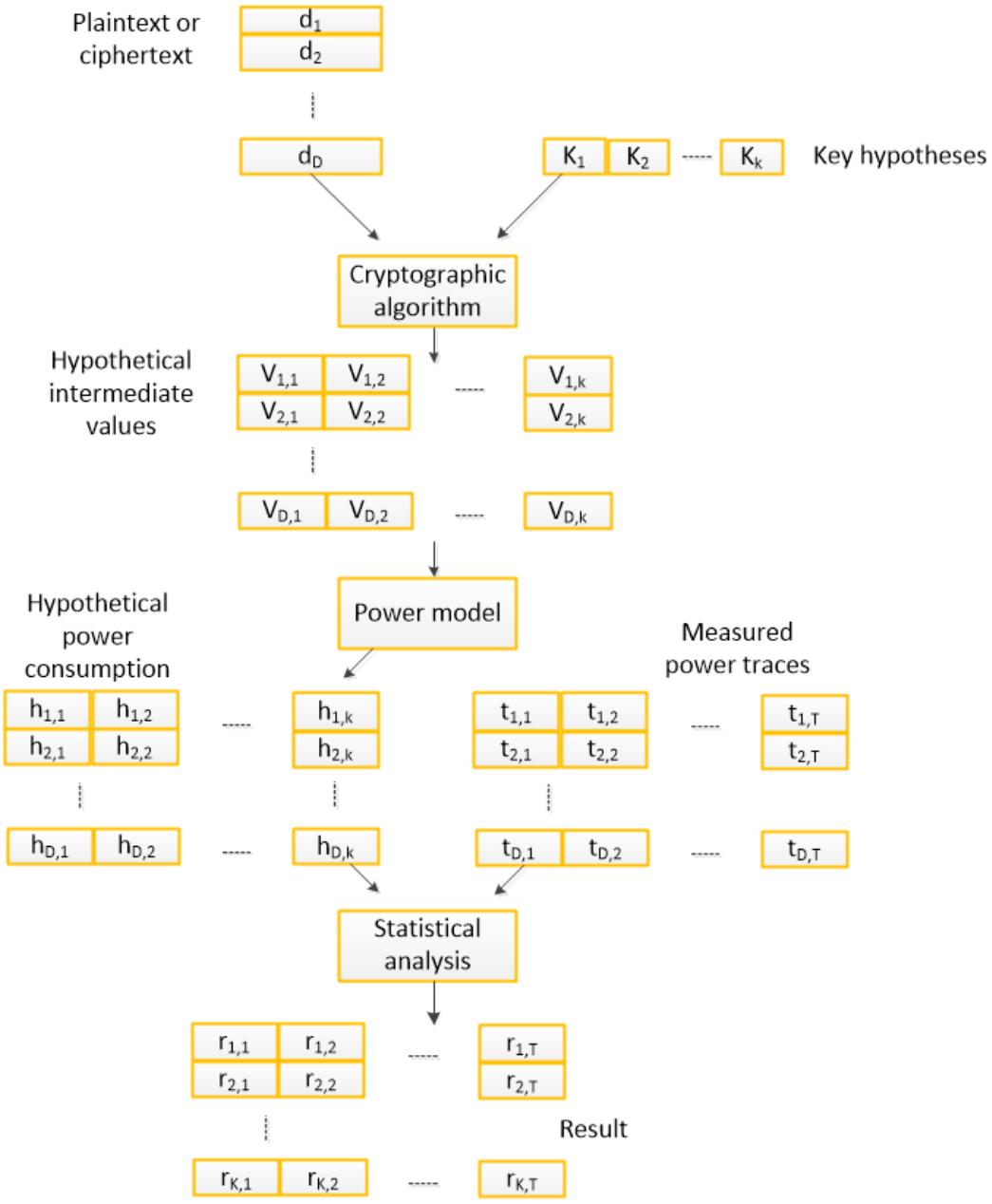


Figure 2.5: Block Diagram of DPA attack

Chapter 3

Two-Share Threshold Implementation Analysis

Threshold Implementation is an (n, n) secret sharing, which means that to obtain the secret value we require all the shares. In such a scheme even the knowledge of up to $n-1$ shares does not reveal any information about the secret value. The shares of a secret value X is represented by $\hat{X} = (X_1 X_2 X_n)$. To generate a different share of value, we use xor function. To divide X into m shares, we need $m-1$ random numbers. Each random number can be used as one of the shares. For computing the share m, we xor all of the random numbers and the secret value X together.

Many countermeasure techniques presented to counter the side-channel attacks. When they applied to Hardware, resulted in leakage due to glitches. In order to address this problem, Nikova et al. [32] presented a Threshold Implementation (TI) technique. The very first proposal of this technique protects against first-order side-channel leakages. Threshold Implementation has shown extensive applicability in a wide range of crypto algorithms from symmetric algorithms [34, 31, 5, 6, 39] to asymmetric algorithms [10, 36] that have been protected successfully. To protect against higher-order attacks, threshold Implementation technique has been developed [7]. Moreover, the drawback of threshold implementation technique is addressed in [35]. Threshold Implementation requires three properties to implement an algorithm side-channel resistant in the presence of glitches. By using additive Boolean masking, i.e., adding randomness, susceptible states are transformed into a shared delineation. Functions $F(.)$ are changed to meet the requirements of correctness, uniformity, and non-completeness.

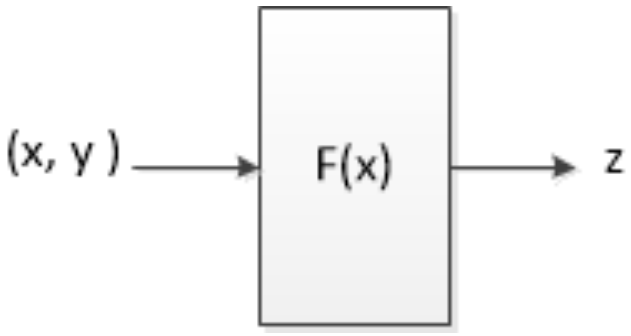


Figure 3.1: Simple function

Correctness: The output is divided into n shares, the correctness means that if combine all of these output shares, the result should be a valid output.

$$\begin{aligned} (x_1, y_1) \quad \oplus \quad (x_2, y_2) \cdots \oplus (x_n, y_n) = (x, y) \\ z_1 \quad \oplus \quad z_2 \cdots \oplus z_n = z \end{aligned} \tag{3.1}$$

Non-Completeness: It needs that any sub-functions of a shared function F used to evaluate any output share have to be missing of at least one input share for first order side-channel resistance. In [7] has been shown that to obtain d-th order SCA resistance, any d sub-functions should be missing of at least one input share. To reveal the secret key all the shares are needed while this property ensures that all the shares are not present in the system at any time instance. Hence, the glitches in the final implementation can not leak the information of the secret key.

Uniformity: If the input shares are uniformly distributed, all intermediate states and the output shares must be uniformly distributed. To be first-order resistant, this property ensures the mean leakage to be independent of the shares. Requires all intermediate states (shares) to be uniformly distributed.

3.1 Threshold Implementation with Two Shares

The Effective method of Nikova et al. [32] admits implementing any d-th order algebraic functions in a simple way. However, implementations of functions with higher degree need extremely more efforts to keep the number of shares

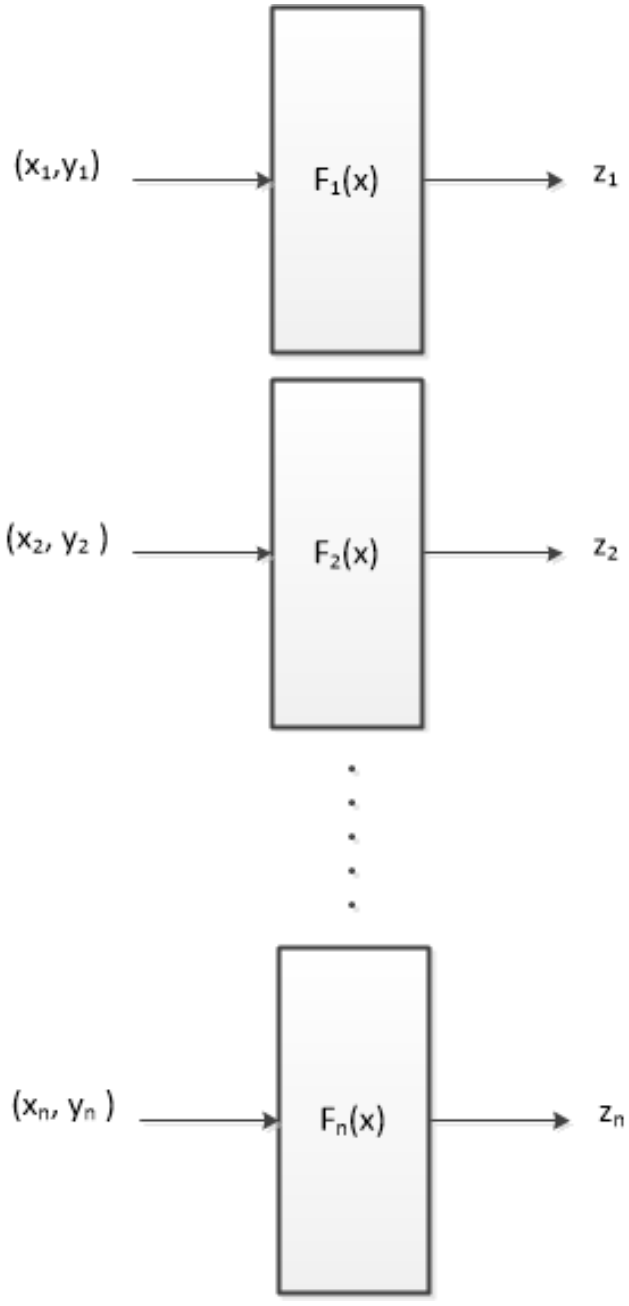


Figure 3.2: Shares of function

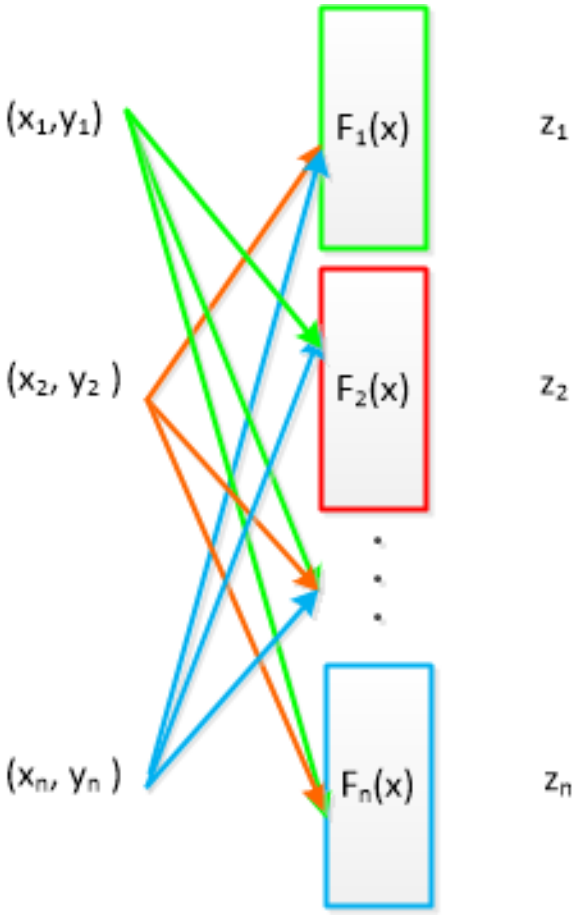


Figure 3.3: Non-completeness in TI, every input share goes to other $n-1$ shares

to minimum possible, i.e., three shares, to implement functions. In [35] offers an improved approach to use fewer shares.

The efficient implementation of 4-bit S-boxes with three shares were investigated in [24]. Also, the threshold implementations using algebraic functions of the AES S-box to implement the area-efficient S-box with four shares [31] or five shares [6].

The approach discussed in [35] uses similar techniques like the ones used by the above papers, but with just two shares, which leads to reducing area overhead and the need for randomness.

The approach for the linear operations is simple in implementation. It is used in linear operation parts of different threshold implementations [6,10,4].

In the following, the simplest nonlinear function $c = ab$ is shown with two shares:

$$c_0 = a_0 b_0 \qquad c_1 = a_1 b_1 \qquad c_2 = a_0 b_1 \qquad c_3 = a_1 b_0 \tag{3.2}$$

In the upper equation, c_2 and c_3 combine inputs from different shares. It may violates the property of the non-completeness if a and b are dependent. Hence, a and b should be statistically independent.

The 4-share output of equation (3.2) is not desirable from the area-efficiency prospective. To minimize the number of shares, the share can be combined together in the next cycle, e.g. $c'_0 = c_0 + c_2$ and $c'_1 = c_1 + c_3$. To meet the non-completeness property, a register-stage should be used in the next clock cycle. It will increase the number of registers as well as latency.

As discussed in [35], the proliferation of the shares gets more complicated for higher-degree functions. To achieve minimal hardware implementation, the higher-order algebraic functions break into minimal degree building blocks to avoid share proliferation concern.

To ensure uniformity and obtain a basic nonlinear building block, we implement $z = ab + c$ in two pipeline stages as:

$$z'_0 = a_0 b_0 + c_0 \quad z'_1 = a_1 b_1 + c_1 \quad z_0 = z'_0 + a_0 b_1 \quad z_1 = z'_1 + a_1 b_0 \tag{3.3}$$

z'_i and z_i are computed in different cycles. Both z'_i and z_i are effortlessly uniform. Unlike (3.2) this equation only requires to store two intermediate states. Utilizing extra register stages for the non-linear function increases both area overhead and latency according to the number of register stages needed. If the data path of the implementation become enough small, this latency can be compensated.

Table 3.1: Comparison of leakage for a 2-sharing (S_2) and 3-sharing (S_3) of a bit x in a Hamming weight model. The 2-sharing (S_2) shows a leakage in the variance $\sigma(S_2)$.

x	$S_2(x)$	$S_3(x)$	$wt(S_2)$	$wt(S_3)$	$\mu(S_2)$	$\mu(S_3)$	$\sigma(S_2)$	$\sigma(S_3)$
0	$\{00, 11\}$	$\{000, 011, 101, 110\}$	$\{0, 2\}$	$\{0, 2, 2, 2\}$	1	$3/2$	2	1
1	$\{01, 10\}$	$\{001, 010, 100, 111\}$	$\{1, 1\}$	$\{1, 1, 1, 3\}$	1	$3/2$	0	1

3.1.1 potential risks

Share rotation To increase side-channel resistance, in [34] it was proposed that in every step, we should rotate the shares. While this would be highly hazardous since we are using just two shares. If one share overwrites to the other share, the leakage will dependent on both shares that will reveal the secrets. Therefore, updating the registers must be managed carefully in a design step.

Increased Higher-order leakage The dependence of the variance on the value of the share x can theoretically explain the potential higher order leakage. For instance, we compare a 2-sharing S_2 and a 3-sharing S_3 of a bit x. $S_2(x) = \langle x_0, x_1 \rangle$ and $S_3(x) = \langle x_0, x_1, x_2 \rangle$ respectively. We consider the Hamming weight leakage model ($wt(\cdot)$) for the shares. The means and variances of the possible states for both sharings are listed in Table 3.1.

Both the 2-share and 3-share threshold implementation of x show that the the value of x is independent of mean leakage $\mu(S_i)$. However, the variance of S_2 depends on x, in particular $\text{var}(S_2(x = 0)) = 2 \neq 0 = \text{var}(S_2(x = 1))$ unlike 3-sharing S_3, where the variances in both cases are identical as well. This indication shows a strong second-order leakage for 2-sharings.

Chapter 4

Design and Implementation

4.1 Application to Present

In this section, we apply two-share Threshold Implementation to the Present cipher. In [24], the authors presented the 3-TI Present S-box. To achieve this, they decomposed the non-linear S-box of degree 3 into the combination of two quadratic functions—G function—plus some linear functions, and then implement them with three shares. We follow their idea to use the same decomposition but then implement them with 2-TI while still retaining *uniformity*, *non-completeness*, and *correctness*. According to [24], the S-box of Present can be decomposed as:

$$S(X) = A(G(G(BX \oplus c)) \oplus d) \tag{4.1}$$

Where $G(.)$, A, B, and the constant vectors of c, d are given as follows:

$$\begin{aligned} G(x, y, z, w) =& (g_3, g_2, g_1, g_0) : \\ g_3 =& x + yz + yw \\ g_2 =& w + xy \\ g_1 =& y \\ g_0 =& z + yw \end{aligned} \tag{4.2}$$

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}, B = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}, c = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}, d = \begin{bmatrix} 0 & 1 & 0 & 1 \end{bmatrix} \tag{4.3}$$

4.1.1 Present with Two Shares

A 2-sharing scheme of $G(.)$ can be expressed as follows:

$$\begin{aligned} G_0(x_0, y_0, z_0, w_0, x_1, y_1, z_1, w_1) =& (g_{03}, g_{02}, g_{01}, g_{00}) \\ g_{03} =& x_0 + y_0 z_0 + y_0 z_1 + y_0 w_0 + y_0 w_1 \\ g_{02} =& w_0 + x_0 y_0 + x_1 y_0 \\ g_{01} =& y_0 \\ g_{00} =& z_0 + y_0 w_0 + y_0 w_1 \end{aligned} \tag{4.4}$$

$$\begin{aligned} G_1(x_0, y_0, z_0, w_0, x_1, y_1, z_1, w_1) =& (g_{13}, g_{12}, g_{11}, g_{10}) \\ g_{13} =& x_1 + y_1 z_0 + y_1 z_1 + y_1 w_0 + y_1 w_1 \\ g_{12} =& w_1 + x_0 y_1 + x_1 y_1 \\ g_{11} =& y_1 \\ g_{10} =& z_1 + y_1 w_0 + y_1 w_1 \end{aligned} \tag{4.5}$$

The above sharing satisfies both *correctness* and *uniformity* when the input shares are uniformly distributed. However, *non-completeness* is not fulfilled since two shares of the same inputs are fed into the same functions in some of the above equations.

We serialize the computations into two steps to achieve *non-completeness* as illustrated in the following equations:

$$
\begin{aligned}
G_0^1(x_0, y_0, z_0, w_0) =& (g_{03}^1, g_{02}^1, g_{01}^1, g_{00}^1) \\
g_{03}^1 =& x_0 + y_0 z_0 + y_0 w_0 \\
g_{02}^1 =& w_0 + x_0 y_0 \\
g_{01}^1 =& y_0 \\
g_{00}^1 =& z_0 + y_0 w_0
\end{aligned} \tag{4.6}
$$

$$
\begin{aligned}
G_0^2(x_1, y_0, z_1, w_1, g_{03}^1, g_{02}^1, g_{01}^1, g_{00}^1) =& (g_{03}^2, g_{02}^2, g_{01}^2, g_{00}^2) \\
g_{03}^2 =& g_{03}^1 + y_0 z_1 + y_0 w_1 \\
g_{02}^2 =& g_{02}^1 + x_1 y_0 \\
g_{01}^2 =& g_{01}^1 \\
g_{00}^2 =& g_{00}^1 + y_0 w_1
\end{aligned} \tag{4.7}
$$

$$
\begin{aligned}
G_1^1(x_1, y_1, z_1, w_1) =& (g_{13}^1, g_{12}^1, g_{11}^1, g_{10}^1) \\
g_{13}^1 =& x_1 + y_1 z_1 + y_1 w_1 \\
g_{12}^1 =& w_1 + x_1 y_1 \\
g_{11}^1 =& y_1 \\
g_{10}^1 =& z_1 + y_1 w_1
\end{aligned} \tag{4.8}
$$

$$
\begin{aligned}
G_1^2(x_0, y_1, z_0, w_0, g_{13}^1, g_{12}^1, g_{11}^1, g_{10}^1) =& (g_{13}^2, g_{12}^2, g_{11}^2, g_{10}^2) \\
g_{13}^2 =& g_{13}^1 + y_1 z_0 + y_1 w_0 \\
g_{12}^2 =& g_{12}^1 + x_0 y_1 \\
g_{11}^2 =& g_{11}^1 \\
g_{10}^2 =& g_{10}^1 + y_1 w_0
\end{aligned} \tag{4.9}
$$

The superscript indicates the level of the circuit. Until now, we achieved a *correct*, *non-complete* and *uniform* two-share implementation of $G(.)$. the conversion of the remaining linear operations is discussed next.

4.1.2 Hardware Implementation

As depicted in Figure 4.1, to provide the *non-completeness* to the design, we use registers to separate the two parts of the G. The second part of the

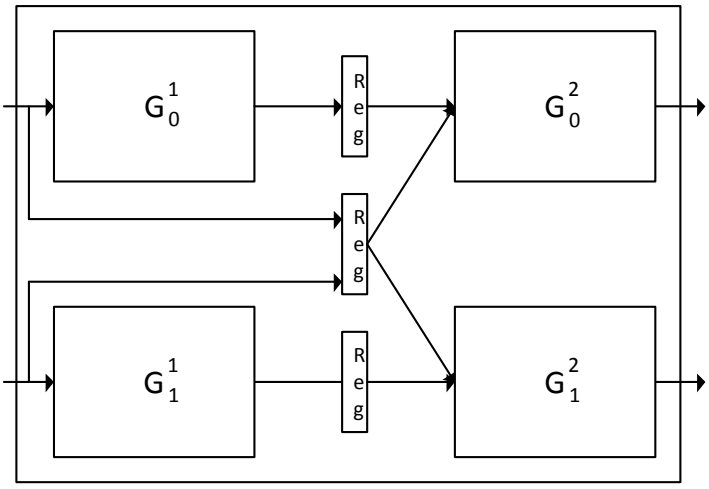


Figure 4.1: Hardware architecture of the 2-share G module

shares (G_0^2 and G_1^2) use not only the outputs of the first part of the shares (G_0^1 and G_1^1) but also some of their inputs as well (depicted in Figure 4.1). One 6-bit register and two 4-bit registers are used before the second part of the G module, to store the inputs x_0, x_1, z_0, z_1, w_0, and w_1; and the outputs of the first part of the G module, respectively.

In Figure 4.2, the S-box architecture is depicted which includes two G modules, and functions $BX + c_0$ and $AX + d_0$ for the first share as well as functions $BX + c_1$ and $AX + d_1$ for the second share in which $c_0 + c_1 = c$ and $d_0 + d_1 = d$. Furthermore, due to *non-completeness*, we use another row of registers in between two $G(.)$ functions in the S-box. One may argue that registers should also be inserted between non-linear functions (e.g., $G(.)$) and linear functions (e.g., $AX + d_0$), since when they are merged the two shares of certain variables may be combined again which fails the *non-completeness* requirement. While this is true in general cases, our design avoids this problem as G_0^2 and G_1^2 are both independent of one share of the inputs and hence any linear combination of $g_{13}^2, g_{12}^2, g_{11}^2, g_{10}^2$ or $g_{03}^2, g_{02}^2, g_{01}^2, g_{00}^2$ still satisfies *non-completeness*.

Figure 4.3 shows the whole Present cipher with two shares. The design includes two control inputs namely `key_load` and `data_load`. If `key_load`

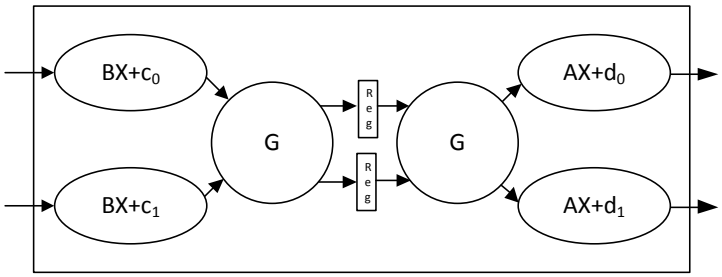


Figure 4.2: Hardware architecture of the 2-share S-box module

is high, at the rising edge of the clock signal, the 80-bit input key shares- Key A and Key B- are copied to the registers Key A and Key B respectively. When the `data_load` signal is high, at the rising edge of the clock signal, 64 right-most significant bits of the input shares (`data_in A[63:0]`, `data_in B[63:0]`) are copied to state registers. It is worth mentioning that when the `data_load` is set, i.e., loading new two shares of plaintext into the state registers results in a reset of the state machine. That why this design does not have a reset signal. When the two-share keys and two-share plaintexts are loaded, both `key_load` and `data_load` must be set to zero. After that, it takes 31 rounds to `Data_out A`, and `Data_out B` have valid ciphertexts. In each round, the S-box and permutation operations respectively operate the inputs to update the state registers for the next round. Considering the hardware design, each $G(.)$ function needs one cycle and then every S-box needs four clock cycles to compute table lookup. According to the Figure 4.3, each 64-bit input stored in the State register needs to use S-box 16 times. Hence, it needs 4 clock cycles for the first S-box due to its latency, plus 15 clock cycles for other 15 S-boxes in a pipeline, also one more clock cycle for the permutation operation. Therefore, we need 20 cycles for each round of the Present cipher. Hence, we define another control signal, 'counter,' in which it updates the state registers and Key registers after each 20 cycles. After each cycle of these 20 cycles, the state registers are shifted to the right by 4 bits, and the four most significant bits of the state registers are replaced by the outputs of substitution and permutation network. The Present cipher has 31 rounds. Hence, a full encryption of a 64-bit input takes 620 clock cycles. We also design an unprotected Present cipher to show the area overhead of the protected Present versus unprotected one as well as its

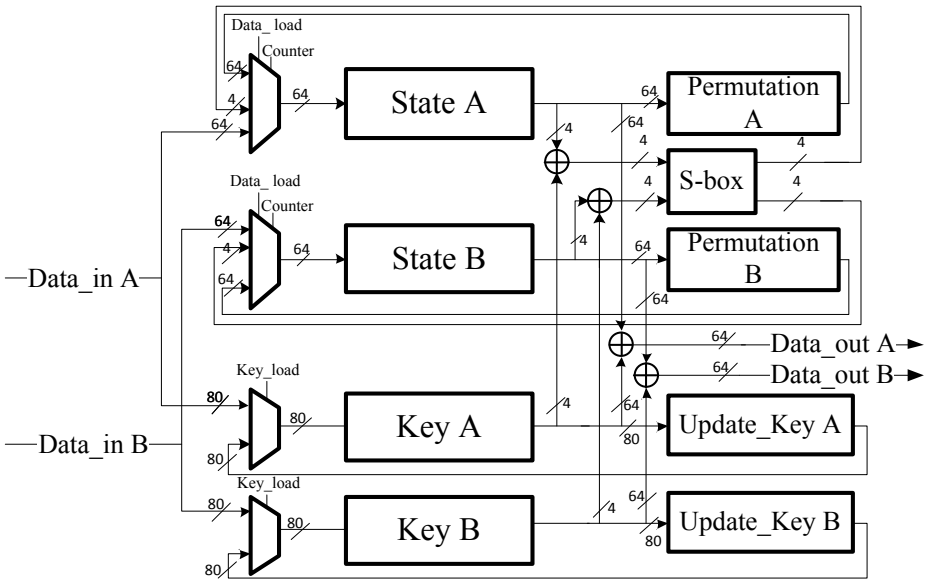


Figure 4.3: Hardware architectures of the 2-shares Present Cipher.

impact on maximum frequency and throughput. The comparison results are shown in Table 5.1.

As we mentioned before, more complex and important part of implementation is the non-linear part of cipher that is S-box. At the beginning of this Chapter, we show that how we can decompose the S-box into smaller parts. The G function is the non-linear part that the formula for that is presented. Then, we apply 2 share threshold technique on it. To meet the required conditions for threshold implementation technique, it is shown that how we can implement the hardware for G function. Afterwards, the linear parts also attached to the G function to build the S-box. In the end, we present the hardware architecture of the 2-shares Present Cipher.

Chapter 5

Results

5.1 Implementation Results

Table 5.1 summarizes the overhead and performance of two-share implementations of a present cipher. Note that we only implement Present64/80 as an example to show the advantage of the two-share scheme. All the designs are implemented in Verilog and synthesized for Virtex-5 (xc5vlx50)[1].

Concerning Present, we have three implementations: Unprotected, Regular 3-TI, and the novel 2-TI Present. Regarding slice registers used, regular 3-TI implementation used more than three times of the unprotected one. This is because we should use extra registers to guarantee the *non-completeness* of a first-order resistant 3-share Present cipher. Also, 2-share implementation

Table 5.1: Implementation results of two-share Present.

Design	**Slice** (Regs)	**Slice** (LUTs)	**Max. Frequency** (MHz)	**Throughput** (Mbps)
Present on Virtex 5				
3-TI Present	466 (3.0x)	715 (3.1x)	397.289	45.567
2-TI Present	370 (2.4x)	742 (3.2x)	490.252	50.61
Present	154 (1x)	234 (1x)	394.563	40.73

[1]The results of this work was published in ASIACRYPT 2016 [11]

Table 5.2: Area complexity comparison for different implementation of Present cipher.

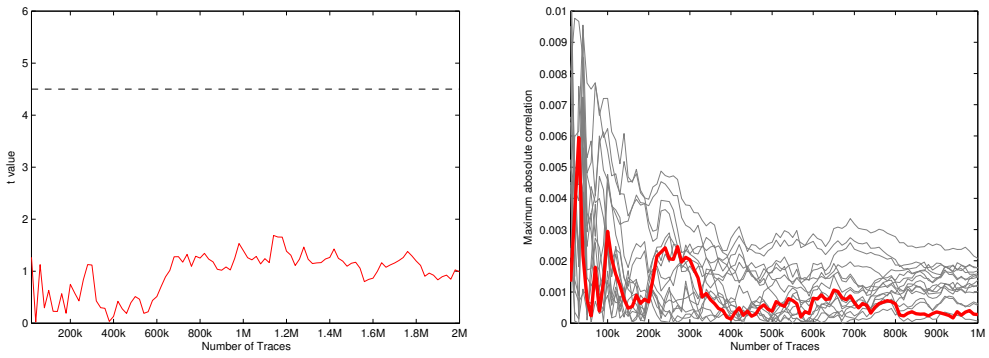
year	**ref**	**Algorithm**	**Platform**	**Slice** (Regs)	**Slice** (LUTs)	**Max. Freq** (MHz)
2012	[18]	Present Serial	Virtex-5 XC5VLX50 FF324-1	203	237	245.76
2012	[18]	Present Iterative	Virtex-5 XC5VLX50 FF324	200	285	250.89
2015	[40]	Present	Virtex-5 XC5VLX50	201	222	236.57
2016	[25]	PRE	Spartan-6 XC6SLX16 CSG324C	136	229	221.63
2016	[25]	PRE_01	Spartan-6 XC6SLX16 CSG324C	137	308	160.13
2016	[25]	PRE_02	Spartan-6 XC6SLX16 CSG324C	89	226	172.92
2016	our work	Present	Virtex-5 XC5VLX50	154	234	394.563

costs more than two times of unprotected Present because of the same reason mentioned before. Moreover, it is worth mentioning that the 2-TI first order resistant implementation uses fewer registers than 3-TI. For example, we use extra registers in $G(.)$ function as explained in Section 4.1. These registers help to reduce the critical path, which explains the speed-up and resulting increase in throughput for 2-TI Present. Leakage analysis of the circuits has been done by Cong Chen.

In Table 5.2, the area complexity of our Present cipher implementation versus other works has been compared, which all of them work with 80-bit key and have 64-bit datapath except PRE02 implementation that uses a 16-bit datapath. The architecture proposed in this work demonstrates that it can be appealing for some applications. Since we decided to apply Threshold Implementation technique on the Present cipher, and the implementation of this technique can be complex on nonlinear functions of the Present cipher. We decided to minimize the nonlinear part of the cipher. Therefore, we design the Present cipher in a way that we just need two S-boxes. One S-box for the substitution layer to apply on plaintext in every round and the other S-box is used in key update part to do the substitution operation to update the key in every round. Hence, this solution can help us in two ways. First, it reduces the size of the hardware implementation and makes it compact. Second, we can apply the Threshold Implementation technique on it in a more straightforward way that is described completely in Chapter 3. However, this architecture leads to a worse latency. The methods that proposed in [25] try to implement smaller and faster implementation considering area-performance tradeoffs. The aim of the work in [25] is to reduce the datapath width as a whole for both substitution layer and permutation layer. The novel architecture of this work has 16-bit datapath. While this architecture demonstrates a better area-optimized architecture, it uses 16 S-boxes that makes it complex and unsuitable for TI technique.

5.1.1 Theoretical Analysis

First, we discuss the strong second-order leakage of two-share TI scheme using two-share Present S-box look-up as a target, namely the key-dependent intermediate value $y = S(x \oplus k)$ where x, y, k are 4-bit input plaintext, S-box output, and sub-key receptively.

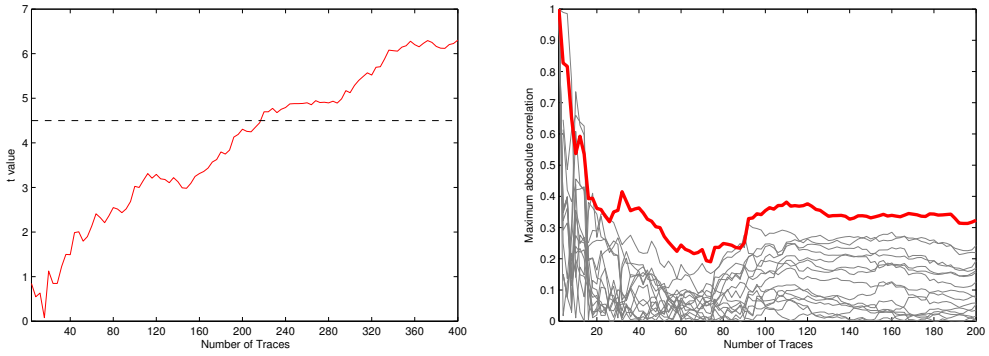


(a) 1st order t-test (b) 1st order CPA

Figure 5.1: First-order leakage analysis of synthetic data. Left: first-order paired t-test. Right: first-order CPA; Red line corresponds to the correct key guess

Synthetic samples and leakage model First, we generate a noise free synthetic leakage samples of the 2-TI Present S-box based on Hamming weight model. As shown in Section 4.1, a 2-TI S-box processes two shares (4 bits for each share) in parallel and hence we use the Hamming weight of both output shares (8 bits in total) as the synthetic leakage samples. Further, for a second order analysis, the synthetic data should be center-and-then-squared. Concerning the leakage model, we use the Hamming weight of the regular S-box output which equals the bitwise XOR between the two output shares in the 2-TI S-box.

First-order analysis We perform first-order *non-specific paired t-test* on the synthetic data and attempt to exploit any leakage using classic CPA as well. For this purpose, 1 million synthetic leakage samples for random input plaintext are generated as well as another 1 million for fixed inputs. The result of t-test using the 2 million samples is shown in Figure 5.1(a) where the t value is less than 2 as the number of traces (synthetic samples) increases to 2 million. Then, a classic first-order CPA is performed on the 1 million samples associated with the random inputs using the above-mentioned leakage model. The results in Figure 5.1(b) shows the correct key cannot be distinguished from the wrong key hypotheses with as much as 1 million samples and the attacks fail.



(a) 2nd order t-test (b) 2nd order CPA

Figure 5.2: Second-order leakage analysis of synthetic data. Left: second-order paired t-test. Right: second-order CPA; Red line corresponds to the correct key guess

Second-order analysis Then, we proceed with second-order *non-specific paired t-test* and CPA. For this purpose, 200 synthetic leakage samples for random input plaintext are generated as well as another 200 for fixed inputs.Figure 5.2(a) shows that t value exceeds 4.5 with only a couple of hundreds of samples while classic CPA can recover the correct key with less than a hundred samples as shown in Figure 5.2(b).

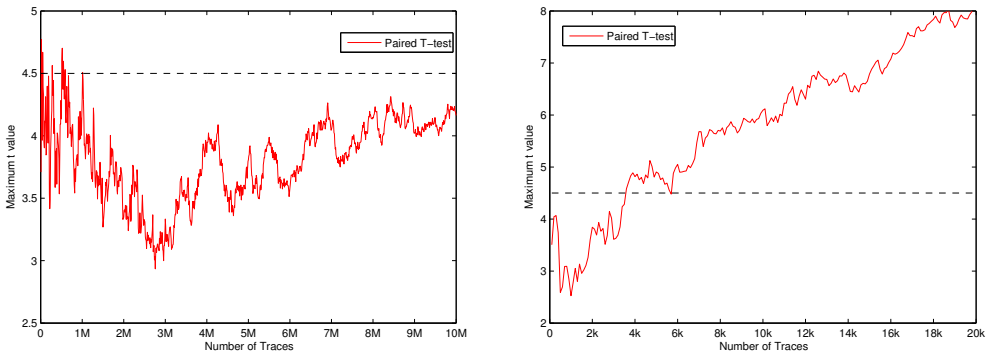
In summary, the theoretical analyses also show the first-order resistance of 2-TI scheme but reveal a strong second-order leakage. This strong second-order leakage is caused by the differing variances, as pointed out in Section 3.1.1. Note that we use perfect Hamming weight model for synthetic data without adding any noise. Hence, the CPA with a Hamming weight model can efficiently recover the key because it captures the leakage well. In fact, CPA on a perfect Hamming weight leakage is comparable to a profiled attack, in the absence of noise. However, in the real world, actual leakages are more complex, and CPA with Hamming weight model will not be as efficient as in this synthetic scenario. In the following, we will analyze practical implementations to show this.

5.1.2 Practical Analysis

Next, we discuss the leakage analysis results for the two-share implementations of Present. First, we apply the *non-specific* paired t-test method

from [16] to detect any data-dependent leakage. Fixed (F) and random (R) measurements are interleaved using the FRRF pattern.

2-TI Present 10 million traces are captured for the two-share Present implementation and then analyzed using the paired t-test. The first order t-statistic is still below 4.5 with 10 million measurements, as shown in Figure 5.3(a). The second order t-statistics exceeds the threshold with about 6000 traces as shown in Figure 5.3(b). Again, the results suggest that two-share TI holds the promise of first order resistance, but fails terribly on the second order resistance.



(a) 1st order t-test (b) 2nd order t-test

Figure 5.3: Leakage detection results for the two-share implementation of Present for first order (left) and second order (right) leakage over the number of traces. Note that the dimensions change for both axes.

We performed second-order CPA on 5 million random traces (center-and-then-squared) on *2-TI Present*, targeting at the S-box output to exploit the leakage. Recall our 2-TI Present in which the 64-bit state registers are right rotated by 4 bits per clock cycle so that the least significant nibble is continuously fed into the S-box look-up and output is written back to the most significant nibble after 4 clock cycles. Therefore, a Hamming distance leakage occurs between consecutive output nibbles. In this attack, we use the Hamming distance power model between the first two consecutive S-box outputs which depends on the least significant key byte and thus 2^8 key hypotheses are required. The max correlations per key hypothesis over the number of traces are shown in Figure 5.4 and the results s how that correct

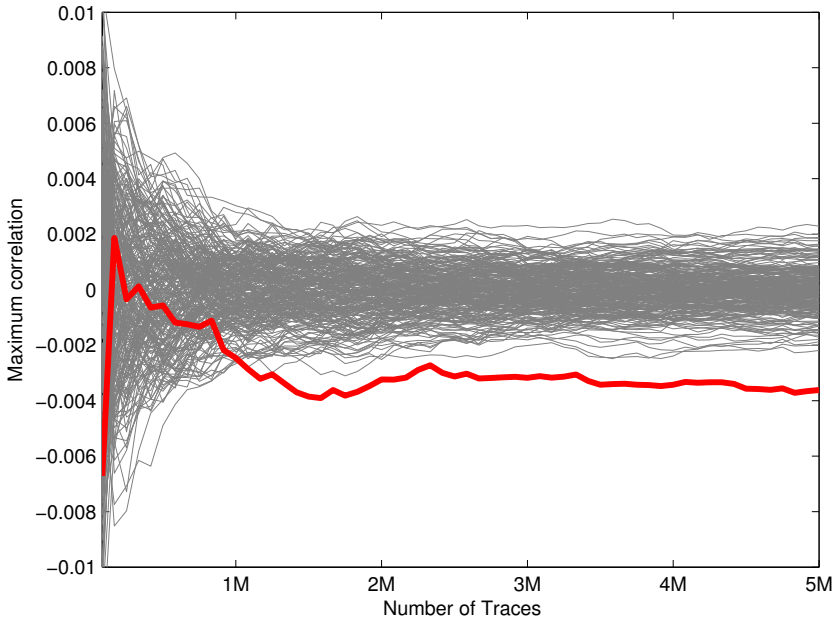


Figure 5.4: Second-order CPA of two-share Present

key can be successfully recovered with more than 1 million traces which demonstrate the practical exploitability of detected leakage.

The results from validating our simulation analysis for the idealized case from Section 3.1.1 and Section 5.1.1, which suggests high second-order leakage.

Unlike the theoretical analysis results in Section 5.1.1 where the number of traces needed for successful second-order t-test and CPA are of the same order magnitude, a lot more traces are needed for practical second-order CPA with Hamming distance model to exploit the leakage detected by t-test with only hundreds to thousands of traces. This is mainly because: 1) Practical implementation do not leak a perfect Hamming weight or Hamming distance leakage; 2) Noises also render the practical attacks inefficient.

While two-share TI shows potential in preventing first order leakage with less overhead, its poor performance on second order leakage resistance compared with three-sharing makes it less worthwhile.

Chapter 6

Conclusion

The first practical threshold implementation with only two shares is offered by this work. It is explained that why we use a lightweight cipher, Present, as a target for threshold implementation. Moreover, we show that how applying two shares can lead to a more compact cipher implementation with less randomness. Although moving to two shares makes implementation process of nonlinear part of the cipher more complex, it reduces the area overhead of the cipher. Moreover, leakage analysis shows that this implementation retains a perfect first-order resistance.

It is worth-mentioning that regarding leakage analysis, though two-share shows noticeable first-order resistance that it meets the goal, we should consider that it has a strong second-order leakage. Hence, it can put the practical aim of this technique in doubt, while three-share TI meets the first-order resistance and have a better resistance against higher order attack in comparison with two-share TI. To decide the practical validity of this technique, it may deserve further analysis.

6.1 Future Work

It could be interesting and beneficial to consider applying higher order TI technique on Present Cipher and analyze the results to evaluate the complexity and overhead of the design. Also, The way the circuit is designed could also be changed and improved. Furthermore, recently a new cipher, gift, has been offered by Subhadeep Banik et al. [3]. It is claimed that while this new cipher uses smaller area and has faster performance, it is correcting

the well-known weakness of Present regarding linear hull. Hence, this cipher could be a good target to apply TI technique on it.

Bibliography

[1] Alexandre Adomnicai, Benjamin Lac, Anne Canteaut, Jacques Fournier, Laurent Masson, Renaud Sirdey, and Assia Tria. On the importance of considering physical attacks when implementing lightweight cryptography. In *Lightweight Cryptography Workshop— NIST*, 2016.

[2] A. Aysu, E. Gulcan, and P. Schaumont. SIMON Says: Break Area Records of Block Ciphers on FPGAs. *Embedded Systems Letters, IEEE*, 6(2):37–40, June 2014.

[3] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. Gift: A small present. *Cryptographic Hardware and Embedded Systems-CHES*, pages 25–28, 2017.

[4] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. Trade-Offs for Threshold Implementations Illustrated on AES. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(7):1188–1200, July 2015.

[5] Begül Bilgin, Joan Daemen, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Gilles Van Assche. Efficient and First-Order DPA Resistant Implementations of Keccak. In Aurlien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications*, Springer LNCS, pages 187–199. 2014.

[6] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. A More Efficient AES Threshold Implementation. In David Pointcheval and Damien Vergnaud, editors, *Progress in Cryptology –AFRICACRYPT 2014*, volume 8469 of *Springer LNCS*, pages 267–284. 2014.

[7] Begl Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-Order Threshold Implementations. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, volume 8874 of *Springer LNCS*, pages 326–343. 2014.

[8] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings*, pages 450–466, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[9] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 16–29. Springer, 2004.

[10] Cong Chen, Thomas Eisenbarth, Ingo von Maurich, and Rainer Steinwandt. Masking Large Keys in Hardware: A Masked Implementation of McEliece. In *Selected Areas in Cryptography — SAC 2015*. Springer LNCS, August 2015. Preprint available at `http://eprint.iacr.org/924`.

[11] Cong Chen, Mohammad Farmani, and Thomas Eisenbarth. A tale of two shares: why two-share threshold implementation seems worthwhile-and why it is not. In *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22*, pages 819–843. Springer, 2016.

[12] Jean-Sébastien Coron, Emmanuel Prouff, and Matthieu Rivain. Side Channel Cryptanalysis of a Higher Order Masking Scheme. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings*, pages 28–44, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[13] Christophe De Canniere, Orr Dunkelman, and Miroslav Knežević. KATAN and KTANTAN–A Family of Small and Efficient Hardware-

Oriented Block Ciphers. In *Cryptographic Hardware and Embedded Systems–CHES 2009*, pages 272–288. Springer, 2009.

[14] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking aes with d+ 1 shares in hardware. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 194–212. Springer, 2016.

[15] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with $d + 1$ Shares in Hardware. In Benedikt Gierlichs and Y. Axel Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016: 18th International Conference*, pages 194–212. Springer Berlin Heidelberg, 2016.

[16] A. Adam Ding, Cong Chen, and Thomas Eisenbarth. Simpler, Faster, and More Robust T-test Based Leakage Detection. In *Constructive Side-Channel Analysis and Secure Design - 7th International Workshop, COSADE 2016, Graz, Austria, April 14-15, 2016, Revised Selected Papers*, pages 163–183.

[17] Thomas Eisenbarth and Sandeep Kumar. A survey of lightweight-cryptography implementations. *IEEE Design & Test of Computers*, 24(6), 2007.

[18] Neil Hanley and Maire ONeill. Hardware comparison of the iso/iec 29192-2 block ciphers. In *VLSI (ISVLSI), 2012 IEEE Computer Society Annual Symposium on*, pages 57–62. IEEE, 2012.

[19] M Joye and JJ Quisquater. Ches 2004. lncs, vol. 3156, 2004.

[20] Elif Bilge Kavun and Tolga Yalcin. RAM-Based Ultra-Lightweight FPGA Implementation of PRESENT. In *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, pages 280–285. IEEE, 2011.

[21] M. Kirschbaum and T. Popp. Evaluation of a DPA-Resistant Prototype Chip. In *Computer Security Applications Conference, 2009. ACSAC '09. Annual*, pages 43–50, Dec 2009.

[22] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in cryptologyCRYPTO99*, pages 789–789. Springer, 1999.

[23] Yogesh Kumar, Rajiv Munjal, and Harsh Sharma. Comparison of symmetric and asymmetric cryptography with existing vulnerabilities and countermeasures. *International Journal of Computer Science and Management Studies*, 11(03), 2011.

[24] Sebastian Kutzner, PhuongHa Nguyen, Axel Poschmann, and Huaxiong Wang. On 3-Share Threshold Implementations for 4-Bit S-boxes. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design*, volume 7864 of *Springer LNCS*, pages 99–113. 2013.

[25] Carlos Andres Lara-Nino, Miguel Morales-Sandoval, and Arturo Diaz-Perez. Novel fpga-based low-cost hardware architecture for the present block cipher. In *Digital System Design (DSD), 2016 Euromicro Conference on*, pages 646–650. IEEE, 2016.

[26] Gregor Leander, Christof Paar, Axel Poschmann, and Kai Schramm. New lightweight des variants. In *International Workshop on Fast Software Encryption*, pages 196–210. Springer, 2007.

[27] Diana Maimut and Khaled Ouafi. Lightweight cryptography for rfid tags. *IEEE Security & Privacy*, 10(2):76–79, 2012.

[28] S. Mangard, M. E. Oswald, and T. Popp. *Power Analysis Attacks - Revealing the Secrets of Smart Cards*. Springer, 2007.

[29] Mohit Mittal. Performance evaluation of cryptographic algorithms. *International Journal of Computer Applications*, 41(7), 2012.

[30] Amir Moradi and Oliver Mischke. *How Far Should Theory Be from Practice?*, pages 92–106. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[31] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In Kenneth G. Paterson, editor, *Advances in Cryptology — EUROCRYPT 2011*, volume 6632 of *Springer LNCS*, pages 69–88. 2011.

[32] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Information and Communications Security*, volume 4307 of *Springer LNCS*, pages 529–545. 2006.

[33] Pascal Paillier and Ingrid Verbauwhede. Cryptographic hardware and embedded systems-ches 2007. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages E1–E1. Springer, 2007.

[34] Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. Side-Channel Resistant Crypto for less than 2,300 GE. *Journal of Cryptology*, 24(2):322–345, 2011.

[35] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. *Consolidating Masking Schemes*, pages 764–783. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[36] Oscar Reparaz, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. A Masked Ring-LWE Implementation. In *Cryptographic Hardware and Embedded Systems–CHES 2015*, pages 683–702. Springer, 2015.

[37] Carsten Rolfes, Axel Poschmann, Gregor Leander, and Christof Paar. Ultra-Lightweight Implementations for Smart Devices–Security for 1000 Gate Equivalents. In *Smart Card Research and Advanced Applications*, pages 89–103. Springer, 2008.

[38] Aria Shahverdi. *Lightweight Cryptography Meets Threshold Implementation: A Case Study for Simon*. PhD thesis, WORCESTER POLYTECHNIC INSTITUTE, 2015.

[39] Aria Shahverdi, Mostafa Taha, and Thomas Eisenbarth. Silent Simon: A Threshold Implementation under 100 Slices. In *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on*, pages 1–6, May 2015.

[40] JJ Tay, MLD Wong, MM Wong, C Zhang, and I Hijazin. Compact fpga implementation of present with boolean s-box. In *Quality Electronic Design (ASQED), 2015 6th Asia Symposium on*, pages 144–148. IEEE, 2015.

[41] Kris Tiri and Ingrid 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*, DATE '04, pages 10246–, Washington, DC, USA, 2004. IEEE Computer Society.

[42] Ritu Tripathi and Sanjay Agrawal. Comparative study of symmetric and asymmetric cryptography techniques. *International Journal of Advance Foundation and Research in Computer (IJAFRC)*, 1(6):68–76, 2014.