Side Channel Analysis of SPARX-64/128: Cryptanalysis and Countermeasures

Sumesh Manjunath Ramesh[3,1,2] and Hoda AlKhzaimi[1,2,3]

[1] Center for Cyber Security, New York University Abu Dhabi, UAE
[2] Division of Engineering, New York University Abu Dhabi, Abu Dhabi, UAE
[3] Tandon School of Engineering, New York University, New York, USA
{r.sumesh.manjunath, hoda.alkhzaimi}@nyu.edu

Abstract. SPARX family of lightweight block cipher was introduced in Asiacrypt 2016. The family consists of three variants a) SPARX-64/128, b) SPARX-128/128 and c) SPARX-128/256. In this work, first, we propose a technique to perform Correlation Power Analysis (CPA) on the SPARX-64/128 cipher. Our technique uses a combination of first-order, second-order and modulo addition CPA methods. Using our proposed technique we extract 128 key bits of SPARX-64/128 cipher with low complexities in general; key guess complexity of 2^{12} and $65000 \approx 2^{16}$ power traces. We initially propose a countermeasure of SPARX-64/128 block cipher against side-channel attacks in terms of power analysis, a threshold implementation based on a serialized design of SPARX-64/128 core. The serialized design of SPARX-64/128 core is implemented in hardware and occupies 60 slices in FPGA. As a countermeasure, this serialized implementation is extended to propose a provably secure threshold implementation of SPARX-64/128 core (TI-SPARX). The TI-SPARX core occupies 131 slices in FPGA and runs at 144 MHz thus, giving a throughput of 9 Mbps. To the best of our knowledge, this is the first side channel attack and countermeasure result on SPARX-64/128 cipher.

Keywords: Side Channel Analysis · Lightweight Cryptography · SPARX · Correlation Power Analysis · Threshold Implementation.

1 Introduction

Cryptographic primitives such as stream ciphers, block ciphers and hash functions are essential building blocks for various security applications and protocols such as SSL, TLS, etc. In the past years, the focus on lightweight designs in the community has been increasing in order to build efficient and secure cryptographic primitives that can be utilized in extreme restricted physical environments such as embedded systems, Internet of Things devices, sensors, RFID tags, energy harvesting devices, and many others.

The designed primitives have been initiated to exhibit optimal criteria for specific metrics as in optimized performance, lower power/energy consumption, reduced area size, increased throughput, and security, among many others.

PRESENT and CLEFIA are lightweight block ciphers proposed as ISO standard for lightweight applications [1]. SIMON and SPECK block ciphers are proposed by the NSA [7]. SPARX[13] is a lightweight block cipher which comes with security bound against differential and linear characteristics. These are a Thefew block ciphers introduced specifically for lightweight applications both in hardware as well as software. Many of these ciphers are based on Substitution-Permutation Network, Feistel Network or Addition-Rotation-Xor (ARX) design methodology.

Every proposed design of cryptographic primitives is associated with certain security rationale and needs a thorough analysis against proposed security margins. Normally, this is achieved through classical statistical and non-statistical cryptanalysis techniques, as well as physical hardware cryptanalysis techniques embodied in side-channel analysis approaches. The primitives which are resistant to valid cryptanalysis techniques are highly recommended. In some cases, even if such primitives are resistant to classical cryptanalysis techniques, and they come with a strong provable security model, they might be vulnerable to certain Side Channel Analysis (SCA) approaches. These approaches facilitate the extraction of secret information from the measured leakage data. Power leakage analysis[19], Fault analysis[14], Electro-Magnetic leakage analysis[24], Acoustic leakage analysis[11] and Timing analysis[23] are a few examples on the exploitation of leakage information to get the secret key. Therefore, SCA is equally important because a) Cryptographic Primitives ultimately gets implemented in hardware and/or software, and b) Hardware/Software may exhibit a physical leakage of information about the internal processing values with the secret key while executing a cryptographic primitive.

Masking is one of the techniques proposed to prevent Side Channel Attack [17] [21]. Many variants of masking schemes were proposed to reduce leakage information. Threshold Implementation [26] is a claimed provably secure approach to protect the implementation of ciphers from side channels. This is based on secret sharing and multi-party computation techniques.

Related Work: In [12], authors retrieved 64-bit key of SIMON-32/64 block cipher using differential power analysis, thus making it vulnerable in hardware applications. In [3], authors proposed a Threshold implementation for SIMON and gave a side channel secure implementation of SIMON which can be used in hardware. Similarly, unprotected Present block cipher is vulnerable to Correlation Power analysis[20] and [5] protects the cipher against first-order side-channel attacks. In [10], the authors gave a secure Speck core and the methodology to secure ARX based ciphers.

Contribution: Our contribution in this research is that we analyze SPARX-64/128 block cipher against power leakage. First-Order Correlation Power Analysis (CPA) and Second-Order CPA techniques are combined to recover the complete secret key of SPARX-64/128 using 2^{12} key guesses and $65000 \approx 2^{16}$ power traces. We implement an area optimized hardware design of SPARX–64/128 and compare with round-based design, for performance based on area, speed, and

throughput. Finally, we propose a secure threshold implementation of SPARX-64/128 as a countermeasure for the proposed side channel attacks.
Organization: The document is organized into seven main sections. The background information on power analysis, threshold implementation and SPARX family algorithm are given in Section 2. Correlation Power Analysis (CPA) technique to recover all 128 bit key of round-based SPARX-64/128 cipher is given in Section 3. After the attack description, SPARX-64/128 serialized implementation is proposed in Section 4. Based on this serialized design, we propose as a threshold implementation in Section 5. The CPA attack details and results and the analysis of threshold implementation results are exhibited in Section 6. Finally, the conclusion of the analysis is presented in Section 7.

2 Background

In this section, we introduce the SPARX Family of Block Cipher along with the basic understanding of First-Order and Second-Order Correlation Power Analysis. In addition to that, CPA technique on n-bit addition modulo 2^n is explained. Finally, Threshold Implementation and its properties are described.

2.1 Preliminaries to Power Analysis (CPA)

A brief introduction to First-Order and Second-Order Correlation Power Analysis (CPA) and CPA for n-bit modulo addition 2^n is given in this section. We assume the power model for our attack to be Hamming Distance model.

Pearson Correlation Coefficient and CPA Pearson Correlation Coefficient (PCC) gives the measure of linear relationship (correlation) between two data sets. This is used in CPA to retrieve the secret key. The power traces of N encryption forms one data set and the *hamming distance* between one round function with a given hypothesis key for the same N plaintexts form another data set. The PCC is calculated for each hypothesis key. Say, for n-bit hypothesis key has total 2^n keys. The highest PCC among all hypothesis keys is the potential candidate secret key.

First-Order CPA In 1999, Paul Kocher et al.[19] introduced simple and differential power analysis. In these methods mostly single bit of secret key is retrieved and the experiment needs to be repeated for more bits. In [16], Correlation Power Analysis method uses Pearson Correlation Coefficient to retrieved more bits at a time. Usually, either first round or last round of the cipher is considered to extract the corresponding round key.

Let us consider a toy block cipher A which takes 16-bit plaintext, X, and 16-bit secret key, K, and output 16-bit ciphertext, Y. The encryption function using AES S-Box[18] is mentioned below.

$$\left.\begin{aligned} X = X_0 \| X_1, \quad K = K_0 \| K_1, \\ S_0 = X_0 \oplus K_0, S_1 = X_1 \oplus K_1, \\ Y_0 = sbox(S_0), Y_1 = sbox(S_1), \\ Y = Y_0 \| Y_1. \end{aligned}\right\} \tag{1}$$

First, power traces of N random plaintext encryption function is captured. Next, retrieve 8-bit K_0 of secret key and then the remaining 8-bit, K_1 of secret key K using CPA method. As in equation 1, X_0 is known plaintext and K_0 is unknown 8-bit of secret key. The total possible value for K_0 is 256. Therefore, by taking each possible value for K_0, say hypothesis key, evaluate only a part of the encryption function on the same N plaintexts and calculate the hamming distance between X_0 and Y_0. Now, find the correlation coefficient (PCC) between the power trace and calculated hamming distance for each hypothesis key. The candidate key for K_0 is the hypothesis key with maximum PCC value. The key guess complexity is 2^8. The same process is repeated for K_1. Therefore with 2^9 key guess complexity 16-bit secret key is retrieved whereas the brute force takes 2^{16} key guess complexity.

Second-Order CPA One way to resist first-order CPA attack is by using random mask with the input[17], [21]. This countermeasure can be thwarted, when the attacker is able to correlate between the points (sample) in a power trace for random mask generation and usage of mask with the input. The attacker can retrieve secret key as described in [27] and [15], but it requires a large number of traces to be successful.

CPA on Addition Modulo 2^n Unlike in the first-order CPA method, where K_0 and K_1 can be retrieved independently, the CPA on addition modulo 2^n function dependents on carry bit from previous bit addition. Hence, the secret bits must be extracted in certain order only. Therefore, first Least Significant Byte (8-bit) is retrieved and then second Least Significant Byte is retrieved till Most Significant Byte, using correlation method.

2.2 Threshold Implementation

Threshold Implementation (TI) [26] is a provably secure countermeasure to side channel attacks. TI uses secret sharing techniques on the input such as plaintext and secret key. It is assumed that the attacker will not be able to measure leakage information of all share at the same time, thus making it a good counter measure against side channel. In TI, the number of shares (n) for each variable is based on number of variables (s) (i.e $n \geq s + 1$) [26]. For secure implementation of linear transformation, each share must be processed independently, whereas for non-linear transformation it is complex. For non-linear transformation, three properties must be satisfied for a secure Threshold Implementation. First: *Non-Completeness*, which makes each sub function independent of at least one share

of all inputs, thus the attacker cannot see the complete output of the function. Second: *Correctness*, which provides the correct result once the output of each sub-function is combined together, thereby the output is not modified. Third: the *Balance* property, which says that the if the input shares are uniformly distributed then the output shares are also uniformly distributed. This is to ensure that there is no leakage because of the differences in the distribution between input and output shares. In [9], the authors showed TI methods to counter higher order side channel attacks. Threshold Implementation is a popular countermeasure used to protect many ciphers including TI-AES[8], TI-Simon[3][4], TI-Speck [10] and TI-Present [5].

2.3 Description of SPARX

In 2016, Daniel et al. proposed a SPARX family of ciphers, which is based on Long Trail Strategy method, to bound the cipher against differential and linear characteristics [13]. The SPARX has three variants based on block and key size : SPARX-64/128, SPARX-128/128 and SPARX-128/256. Each block in SPARX-n/k consist of $w = n/32$ words of 32 bits and key is divided into $v = k/32$ words.

The encryption algorithm consist of n_s steps. Each step consist of r_a round function and one linear-mix layer. In each round, there are two operations 1) Round Key addition, 2) Addition Box (A-Box) operation, which consist of Addition Rotation and Xor operations. The parameters for each variant of SPARX is given in [Table 1].

Table 1. Parameters of SPARX-64/128

Parameters	SPARX-64/128	SPARX-128/128	SPARX-128/256
State Word (w)	2	4	4
Key word (v)	4	4	8
Steps (n_s)	8	8	10
Rounds/Step (r_a)	3	4	4
Total Rounds	24	32	40

The Round function for all variant of SPARX is same, whereas the linear-mix layer between SPARX-64/128 and other two variants are different. Similarly, step structure between SPARX-64/128 and other two variants are different. Finally, key scheduling algorithm is different for all the three variants.

Hereafter, SPARX-64/128 and SPARX are used interchangeably and it refers to SPARX-64/128 variant unless otherwise specified. SPARX block size is 64 bit and key size is 128 bit. The state at step s and round r is represented as $X^0_{r,s}\|X^1_{r,s}\|X^2_{r,s}\|X^3_{r,s}$, where size of $X^i_{r,s}$ is 16-bit. The initial state is loaded with plaintext and represented as $X^0_{0,0}\|X^1_{0,0}\|X^2_{0,0}\|X^3_{0,0}$. After one round function the state is $X^0_{1,0}\|X^1_{1,0}\|X^2_{1,0}\|X^3_{1,0}$. At the end of one step (i.e.) after linear-mix layer,

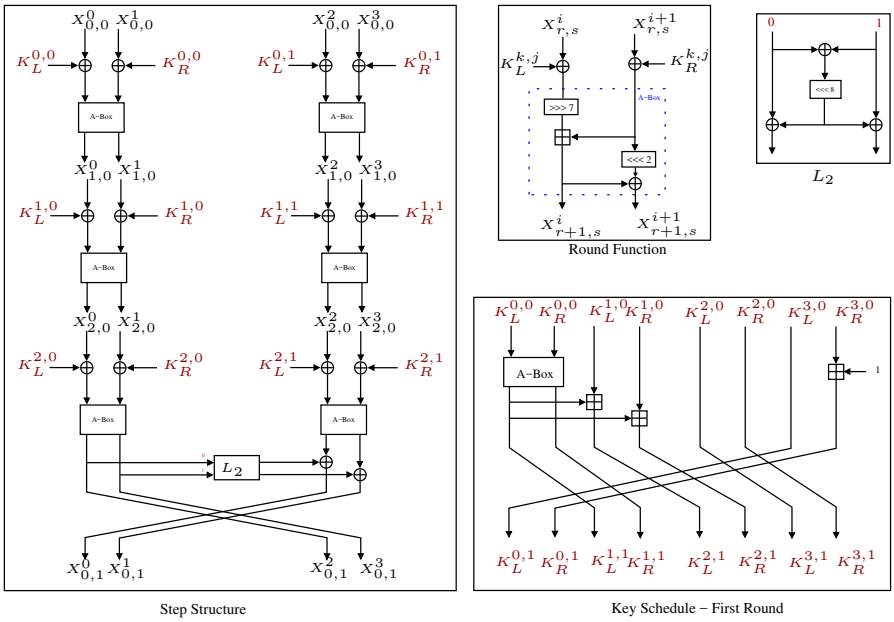

Fig. 1. One step of SPARX-64/128, One round function with A-Box, Linear Mix Layer and one round of Key scheduling algorithm

the state is $X^0_{0,1}\|X^1_{0,1}\|X^2_{0,1}\|X^3_{0,1}$. The i^{th} round key is $K^{0,i}_L\|K^{0,i}_R$, $K^{1,i}_L\|K^{1,i}_R$, $K^{2,i}_L\|K^{2,i}_R$,$K^{3,i}_L\|K^{3,i}_R$, where $K^{j,i}_L, K^{j,i}_R, (0 \leq j \leq 3)$ are 16-bit each. The 128-bit master key is $K^{0,0}_L\|K^{0,0}_R$, $K^{1,0}_L\|K^{1,0}_R$, $K^{2,0}_L\|K^{2,0}_R$,$K^{3,0}_L\|K^{3,0}_R$.

In 2017, Abdelkhalek et al. in [2], proposed an impossible differential distinguisher for 13 round to attack 16 round of SPARX-64/128. Recently in 2018, Ankele et al. in [25], proposed a chosen ciphertext differential attacks on 16 round of SPARX-64/128. Until now there has been no work on side channel analysis of SPARX block cipher.

3 CPA on SPARX-64/128: Full Key Recovery

The first-order and second-order CPA attack described in section 2.1 is combined to retrieve secret key bits used in a round function. Thereby, complete 128-bit key bits are extracted using the secret key bits from 4 round functions and key scheduling algorithm which is explained in detail below.

3.1 Attack on SPARX Round Function

Let us assume that one input to the round function $X^0_{r,s}\|X^1_{r,s}$ is known, another input $K^{0,i}_L\|K^{0,i}_R$ is unknown. SPARX round function after key addition and A-

Box operation it output $X^0_{r+1,s} \| X^1_{r+1,s}$. The first round function is shown in Figure 2.

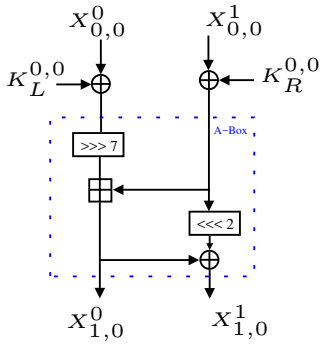

Fig. 2. SPARX-64/128 first round function. A round function consist of round sub-key XOR and A Box operation

It is noteworthy that, only first order CPA techniques cannot be used to retrieve either $K^{0,i}_L$ or $K^{0,i}_R$ because each act as mask to other in the operation. In contrast, by using second-order CPA technique, $X^0_{1,0}$ is removed from $X^1_{1,0}$, thereby revealing $K^{0,i}_R$, 16-bit secret by hypothesis keys. Here instead of retrieving all 16 bits at a time, eight least significant bits are extracted first and then fixing that eight least significant bits, the remaining eight significant bits are retrieved, thereby all 16-bits of secret key $K^{0,i}_R$ are retrieved in 2^9 key guess complexity.

Once $K^{0,i}_R$ secret key is retrieved, it is fixed to that value and then first-order CPA for addition modulo 2^{16} technique is applied to retrieve 16-bit $K^{0,i}_L$ secret key. This takes 2^9 key guess complexity. Thereby, 32-bit secret key $K^{0,i}_L \| K^{0,i}_R$ used in the first round function is extracted in 2^{10} key guess complexity.

Extract 16-bit secret key $K^{0,i}_R$: The following operations occur different time instance.

- At T_1: $X^0_{r+1,s} = \left(X^0_{r,s} \oplus K^{0,i}_L\right)_7 + \left(X^1_{r,s} \oplus K^{0,i}_R\right) \mod 2^{16}$,
- At T_2: $X^1_{r+1,s} = X^0_{r+1,s} \oplus \left(X^1_{r,s} \oplus K^{0,i}_R\right)_{-2}$.

For second-order DPA attack, power measurement at two time instance is subtracted to get modified power measurement of a trace.

As in Equation(2), $X^1_{r,s}$ is known and using first-order CPA technique, $K^{0,i}_R$ is extracted. To reduce the key guess complexity, first, eight least significant bits (lsb) is targeted. Hence, total hypothesis keys are $2^8 = 256$. The round function is simulated with these hypothesis keys keeping the eight most significant bits (msb) to zero. The hamming distance between output from the simulated hypothesis key and input is correlated with $P(t = T_1) - P(t = T_2)$ from power

traces. The hypothesis key which gives maximum correlation coefficient is the correct 8-bit lsb of $K_R^{0,i}$. Fixing the 8 lsb and repeating the simulation for 8 msb position. In this way, $K_R^{0,i}$ is extracted with $2^8 + 2^8 = 2^9$ hypothesis key guesses.

$$\left.\begin{aligned} P(t = T_1) - P(t = T_2) &\approx HD\left(X^0_{r+1,s}\right) - HD\left(X^1_{r+1,s}\right), \\ &\approx HW\left(X^0_{r+1,s} \oplus X^1_{r+1,s}\right), \\ &\approx HW\left(X^0_{r+1,s} \oplus \left(X^0_{r+1,s} \oplus \left(X^1_{r,s} \oplus K^{0,i}_R\right)_{-2}\right)\right), \\ &\approx HW\left(X^0_{r+1,s} \oplus X^0_{r+1,s} \oplus X^1_{r,s} \oplus K^{0,i}_R\right), \\ &\approx HW\left(X^1_{r,s} \oplus K^{0,i}_R\right). \end{aligned}\right\} \tag{2}$$

Extract 16-bit secret key $K_L^{0,i}$: 16-bit $K_R^{0,i}$ is fixed to the above extracted value. $X^0_{r,s}$ and $X^1_{r,s}$ is also known. Therefore, the modular addition is given in Equation (3)

$$\left.\begin{aligned} P(t = T_1) &\approx HD\left(X^0_{r+1,s}, X^0_{r,s}\right), \\ &\approx HD\left(\left(\left(X^0_{r,s} \oplus K^{0,i}_L\right)_7 + \left(X^1_{r,s} \oplus K^{0,i}_R\right)\right) \bmod 2^{16}, X^0_{r,s}\right), \\ &\approx HD\left(\left((known \oplus K^{0,i}_L)_7 + (known \oplus known)\right) \bmod 2^{16}, known\right). \end{aligned}\right\} \tag{3}$$

As per the Equation (3), only $K_L^{0,i}$ is unknown. Therefore, using CPA technique on Addition Modulo 2^n, we extract $K_L^{0,i}$. Thus, $K_L^{0,i}$ is extracted with $2^8 + 2^8 = 2^9$ hypothesis key guesses from Modular addition CPA technique.

Total Key Guess Complexity: To guess the correct value for $K_R^{0,i}$ sub-key, 2^9 key guesses are required and for $K_L^{0,i}$ sub-key, 2^9 key guesses are required. Therefore the total number of key guess complexity for one round function is 2^{10}, whereas the brute force for the same is 2^{32}.

3.2 Full Key Recovery on SPARX-64/128

The technique to retrieve 32-bit secret key from a round function and one round key scheduling algorithm are used to extract complete 128-bit master key. First, key scheduling algorithm is explained with few observations. The 128-bit master key is $K_L^{0,0} \| K_R^{0,0} \| \ K_L^{1,0} \| K_R^{1,0} \| \ K_L^{2,0} \| K_R^{2,0} \| \ K_L^{3,0} \| K_R^{3,0}$. After one round of key scheduling algorithm the second round key is $K_L^{0,1} \| K_R^{0,1} \| \ K_L^{1,1} \| K_R^{1,1} \| \ K_L^{2,1} \| K_R^{2,1} \| \ K_L^{3,1} \| K_R^{3,1}$ as shown in Figure 1.

Observation 1 *$K_L^{0,1}$ of second round key is same as $K_L^{3,0}$ of first round key.*

$$K_L^{3,0} = K_L^{0,1}$$

Observation 2 *The relationship between* $K_R^{0,1}$ *of second round key and* $K_R^{3,0}$ *of first round key is*

$$K_R^{3,0} = \left(K_R^{0,1} - 1\right) \ mod\ 2^{16}$$

Let's explain the full key recovery. The plaintext is loaded into the state $X_{0,0}^0 \| X_{0,0}^1 \| X_{0,0}^2 \| X_{0,0}^3$ and the first round key is $K_L^{0,0} \| K_R^{0,0} \| K_L^{1,0} \| K_R^{1,0} \| K_L^{2,0} \| K_R^{2,0} \| K_L^{3,0} \| K_R^{3,0}$.

For the first round function, the inputs are $X_{0,0}^0 \| X_{0,0}^1$ and $K_L^{0,0} \| K_R^{0,0}$ and after key addition and A-Box operation the output is $X_{1,0}^0 \| X_{1,0}^1$. By our proposed CPA technique as explained in Section 3.1, 32-bit secret key, $K_L^{0,0} \| K_R^{0,0}$, is extracted. Since, key and the plaintext known, the output of first round function, $X_{1,0}^0 \| X_{1,0}^1$, is also known.

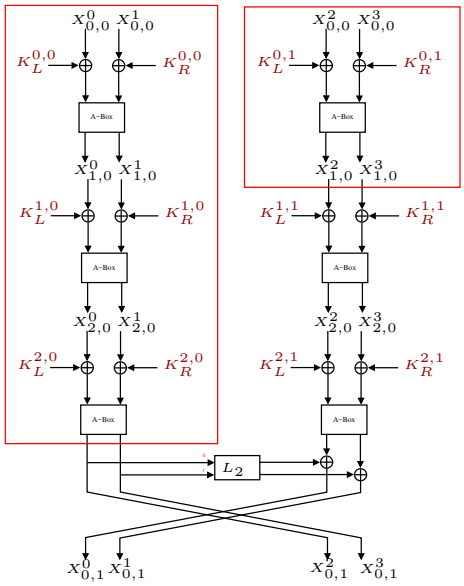

Fig. 3. SPARX-64/128 one step function. The four round functions used to retrieve 128-bit secret key is highlighted

For second round function, one input, $X_{1,0}^0 \| X_{1,0}^1$, is known and the key input $K_L^{1,0} \| K_R^{1,0}$ is unknown. Using the similar technique, $K_L^{1,0} \| K_R^{1,0}$ is extracted, thereby the output of second round function, $X_{2,0}^0 \| X_{2,0}^1$ is known. By following the same way, $K_L^{2,0} \| K_R^{2,0}$ secret key is extracted. So far, 96-bits of secret key is extracted which are $K_L^{0,0} \| K_R^{0,0}$, $K_L^{1,0} \| K_R^{1,0}$, $K_L^{2,0} \| K_R^{2,0}$. Now 32-bit, $K_L^{3,0} \| K_R^{3,0}$, secret key is not used in any of the round functions.

Let us retrieve 32-bit $K_L^{0,1} \| K_R^{0,1}$ second round key. As shown in figure 3, the input to the right side round function is 32-bit of plaintext, $X_{0,0}^2 \| X_{0,0}^3$ which is known and using our proposed technique, $K_L^{0,1} \| K_R^{0,1}$, 32-bit second round key is extracted. Now, using observation 1 and 2, $K_L^{3,0} \| K_R^{3,0}$ is retrieved from $K_L^{0,1} \| K_R^{0,1}$. Therefore, all 128-bit secret key is retrieved.
Complexity: For extracting 32-bit secret key from one round function is 2^{10}. To retrieve complete 128-bit secret key, we need to retrieve 32-bit secret key from four round functions, therefore the total time complexity is $2^{10}+2^{10}+2^{10}+2^{10} = 2^{12}$ key guesses.

4 SPARX Hardware Implementation

The SPARX algorithm is a lightweight block cipher and its potential uses will be in embedded devices, IoT devices where optimized hardware implementation of the cipher is important. Optimization means with respect to area, throughput and speed are required based on the applications. In this work, we optimize the area and propose as serialized implementation of SPARX core. The round function of SPARX is similar to the round function of SPECK with the difference in the key XOR operation, therefore our proposed serialized implementation is inspired from [10].

The block size for SPARX-64/128 is 64-bit. The SPARX algorithm have round function which takes only 32-bit inputs whereas linear-mix layer needs 64-bit input to process. First, serialized implementation of round function will be explained and then implementation of linear-mix layer and finally, complete step structure of SPARX is optimized.

4.1 Serialized Round Function

The round function of SPARX comprises key addition and A-Box operations. The input to the round function is 32-bit state and 32-bits of round key. The 32-bit state is split into two 16-bit data and stored in X and Y registers. The main operations are XOR, cyclic shift and modulo addition. XOR operation is linear and implemented bit-wise. Splitting the registers eliminates the need for implementing cyclic shift operations. Modulo addition of 16-bit input is implemented as 16 serialized one-bit full adder by storing the carry from previous one-bit addition. This takes less area in the hardware. Since the new values are stored in the same registers, there are two feedback functions: One for the X register and another for Y register as shown in Fig 4.
The X register in the left is split into two parts such as X[6:0] and X[15:7], thereby exposing eighth bit position of X register to the left feedback function and the first bit position of Y register is sent to the left feedback function. In the feedback function, corresponding key bits are XORed with eighth and first bit positions of X and Y registers, respectively. After key XOR, one-bit full addition is performed. The new carry is stored and the left feedback function output the sum. For the first seven clock cycles in each round, the output of left feedback

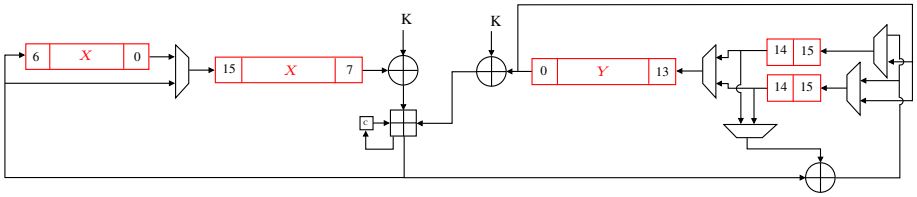

Fig. 4. Serialized implementation of round function of SPARX-64/128

function is fed into $X[6:0]$ part of register as new value and at the same time the old values in $X[6:0]$ are feed into $X[15:7]$ so that it can be processed in the left feedback function. After seven clock cycles, the sum is directly fed into $X[15:7]$ part of register, thereby exposing 8^{th} bit of new value in X register for next round as shown in Fig 4.

The values in Y register are processed twice, one in left feedback function and another in right feedback function. Hence, we need to duplicate values in Y register. Since the right feedback function consist of two cyclic shift operation and XOR operation, duplicate of two bits is sufficient instead of all 16 bits of Y register. One copy stores the old values for right feedback function and another copy stores new values for next round. Hence, the Y register is split into $Y[13:0]$ and $Y[14:15]$ and two copies of $Y[14:15]$ is maintained. The right feedback function performs XOR between output of left feedback function and fourteenth bit in Y register. The output is fed back into fifteenth bit position of Y register as new value for next round. After sixteen clock cycles, one round function is completed. At this instance, one copy of $Y[14:15]$ register has old values and another copy has new value. The register having new value is valid and the register having old value is invalid for next round function. So the valid register is used for next round and invalid register is used to store new values from the new round. Thus, the role of each registers are reverse at the start of each round. One round function takes sixteen clock cycles.

4.2 Serialized Linear-Mix layer

Linear-mix layer of SPARX-64/128 takes 64-bit input and after mixing it gives 64-bit output as shown in the Fig 1. The linear-mix layer is implemented in two steps. In first step, the output of the linear function is processed without the cyclic shift operation. In second step, cyclic left shift of output by 32 bits is implemented. The 64-bit input state are stored in four 16-bit registers such as X_L, Y_L, X_R and Y_R. X_L and Y_L registers stores the state values on the left branch and X_R and Y_R stores the state values on the right branch as shown in the Fig 5.

Linear-mix layer, is implemented as two linear functions (i.e.) left linear function and right linear function. The left linear function takes 32-bit data from X_L and Y_L registers and performs XOR and then cyclic shift operations. The right

linear function takes the output of left linear function and 32-bit data in X_R and Y_R registers as input and performs XOR operation.

In the left linear function, cyclic shift is implemented by directly exposing eighth bit position of X_L and Y_L registers,respectively, such as X_L[7:0], X_L[15:8], Y_L[7:0] and Y_L[15:8]. The eighth bit of X_L and Y_L registers are XORed and the value is again XORed with first bit of X_L and Y_L separately (i.e. two XOR) and these two bits, say, b_x, b_y, are the output of left linear function. Meanwhile, eighth bit of X_L is fed back into seventh bit position of X_L and at same time zeroth bit X_L is feed into fifteenth bit position of X_L. The similar feedback operations are performed for Y_L as well.

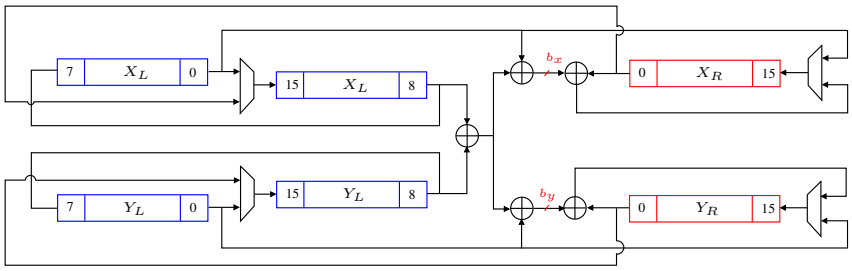

Fig. 5. Serialized implementation of linear mix layer of SPARX-64/128

In the right linear function, the output bit, b_x of left linear function is XORed with first bit of X_R. The XORed bit value is feedback into fifteenth bit position of X_R. Similarly, the output bit, b_y of left linear function is XORed with first bit position of Y_R and the XORed bit value is feedback into fifteenth bit position of Y_R. After sixteen clock cycles, the values in X_L, Y_L, X_R and Y_R registers are new values.

Now the rotation step is implemented. The values in X_L and X_R registers are swapped. Similarly, the values in Y_L and Y_R registers are also swapped. The swap values are taken from zeroth bit position and feed into fifteenth bit position.

Since X_L and Y_L registers are split to expose eighth bit position, while feeding new value at fifteenth position, eighth bit is fed into seventh bit position as shown in Fig 5. The rotation step takes sixteen clock cycles to complete. Therefore the linear mix layer takes 32 clock cycles to complete.

4.3 Serialized SPARX

In section 4.1 and section 4.2, serialized implementation of round function and linear mix layer are proposed independently. Now, we combine both the design to implement one step structure of SPARX-64/128. Four 16-bit registers such as X_L, Y_L, X_R and Y_R are loaded with 64 bit plaintext sequentially. X_L and X_R registers are split into three, to expose required bits for round and linear mix

layer functions. In the same way, Y_L and Y_R registers are split into three and corresponding bits are duplicated as shown in Fig 6.

Appropriate registers are selected for the round functions. For three rounds in a step, first X_L and Y_L registers are selected and executed with first round key bits, and once it is done, X_R and Y_R registers are now selected and again three rounds are executed with second round key bits. Finally, linear-mix layer is executed on all four registers simultaneously as shown in Fig 6.

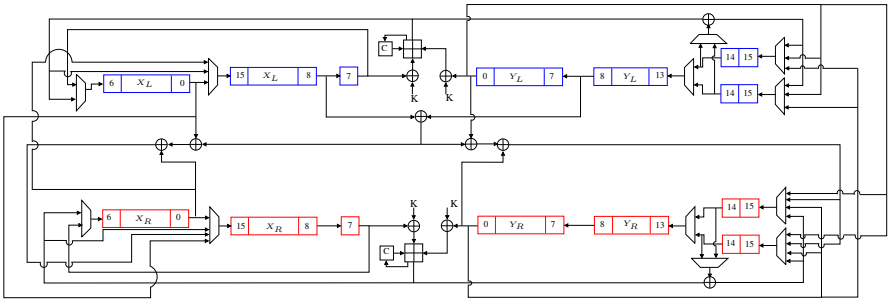

Fig. 6. Serialized implementation of SPARX-64/128 with round function and linear mix layer

5 Threshold Implementation of SPARX

The Threshold Implementation proposed in this work inspires TI implementation of Speck proposed in [10], because the round function in the SPARX is similar to the round function of SPECK expect the key addition operation. This similarity allows us to utilize the TI of Speck with little modification.

For a secure threshold implementation, *Correctness*, *Non-Completeness* and *Balance* properties needs to be satisfied. There are two variables : plaintext and secret key. Therefore, minimum three shares are required for TI for the reason as given in [26]. So, plaintext P is split as P_1, P_2, and P_3 shares, similarly key K into K_1, K_2, and K_3 shares. These shares are generated as shown in equation 4.

$$\begin{aligned} P_1 \xleftarrow{\$} \{0,1\}^{64}; P_2 \xleftarrow{\$} \{0,1\}^{64}; P_3 = P \oplus P_1 \oplus P_2 \\ K_1 \xleftarrow{\$} \{0,1\}^{64}; K_2 \xleftarrow{\$} \{0,1\}^{64}; K_3 = K \oplus K_1 \oplus K_2 \end{aligned} \tag{4}$$

The shares generated are random and the correctness is verified as given in equation 4. SPARX round function consist of XOR, cyclic shift and modulo addition operations; and linear-mix layer consist of XOR and cyclic shift operations. Since, XOR and cyclic operation are linear operations, they operate on each share independently without exposing other shares. Therefore, non-completeness property

is achieved on these operations. Modulo addition is the only non linear operation in round function. It is implemented using 1-bit full adder. The threshold implementation for 1-bit addition used in this work is inspired from [28].

At each clock cycle, one bit of two inputs are added. Therefore, for a given two n-bit inputs, starting from least significant bit to the most significant bit of two inputs, each bit are added using one-bit adder circuit, hence n clock cycles are required to complete the addition. Therefore, at i^{th} clock cycle, i^{th} bit is added, where $i \in \{0, 1, \ldots 15\}$.

Let us explain how TI is achieved in one-bit full adder. One-bit full adder takes two input bits and input carry bit from previous bit addition and output sum bit and output carry bit. For 16-bit addition, the input bits comes from two 16-bit data, let us say a and b, where a_i, b_j represents i^{th}, j^{th} bit of a and b, respectively. The carry bit is denoted as c, where least significant bit of c is zero ($c_1 = 0$). In threshold implementation, a and b are split into three shares and their corresponding carry bit also contains three shares as shown below.

$$\begin{aligned} a_i &= a_{i,1} \oplus a_{i,2} \oplus a_{i,3} \\ b_i &= b_{i,1} \oplus b_{i,2} \oplus b_{i,3} \\ c_i &= c_{i,1} \oplus c_{i,2} \oplus c_{i,3} \end{aligned} \tag{5}$$

Now, the three shares of sum bit s_i and output carry bit c_{i+1} is given below

$$\begin{aligned} s_{i,1} &= a_{i,1} \oplus b_{i,1} \oplus c_{i,1} \\ s_{i,2} &= a_{i,2} \oplus b_{i,2} \oplus c_{i,2} \\ s_{i,3} &= a_{i,3} \oplus b_{i,3} \oplus c_{i,3} \end{aligned} \tag{6}$$

$$\begin{aligned} c_{i+1,1} =& (a_{i,2} \cdot b_{i,2}) \oplus (a_{i,2} \cdot b_{i,3}) \oplus (a_{i,3} \cdot b_{i,2}) \\ & (a_{i,2} \cdot c_{i,2}) \oplus (a_{i,2} \cdot c_{i,3}) \oplus (a_{i,3} \cdot c_{i,2}) \\ & (b_{i,2} \cdot c_{i,2}) \oplus (b_{i,2} \cdot c_{i,3}) \oplus (b_{i,3} \cdot c_{i,2}) \\ c_{i+1,2} =& (a_{i,3} \cdot b_{i,3}) \oplus (a_{i,3} \cdot b_{i,1}) \oplus (a_{i,1} \cdot b_{i,3}) \\ & (a_{i,3} \cdot c_{i,3}) \oplus (a_{i,3} \cdot c_{i,1}) \oplus (a_{i,1} \cdot c_{i,3}) \\ & (b_{i,3} \cdot c_{i,3}) \oplus (b_{i,3} \cdot c_{i,1}) \oplus (b_{i,1} \cdot c_{i,3}) \\ c_{i+1,3} =& (a_{i,1} \cdot b_{i,1}) \oplus (a_{i,1} \cdot b_{i,2}) \oplus (a_{i,2} \cdot b_{i,1}) \\ & (a_{i,1} \cdot c_{i,1}) \oplus (a_{i,1} \cdot c_{i,2}) \oplus (a_{i,2} \cdot c_{i,1}) \\ & (b_{i,1} \cdot c_{i,1}) \oplus (b_{i,1} \cdot c_{i,2}) \oplus (b_{i,2} \cdot c_{i,1}) \end{aligned} \tag{7}$$

By XORing each shares, gives the correctness property. As given in equation 6 and 7, each share of sum and carry is independent of at least one share of each input, thus non-completeness property is also satisfied. As given in [28], the sum and carry shares are uniformly distributed. Thus, all three properties of threshold implementation are satisfied.

6 Results

We performed CPA attack on SPARX-64/128 using SAKURA-G board and also proposed a provably secure threshold implementation of SPARX in Spartan 3.

6.1 Full Key Recovery Results

We implemented SPARX-64/128 cipher in SAKURA-G FPGA board and captured the power traces using Tektronix MSO5204B oscilloscope. SAKURA-G board consist of two Spartan-6 FPGA where one is controller FPGA and another is the main FPGA. SPARX design is implemented in main FPGA. The board is connected with a laptop to send plaintext for encryption and receive the corresponding ciphertext. At the same time, an oscilloscope is connected with the board to sample the power consumption during the encryption process. These sampled power measurements are called power traces and it is stored in the PC via the oscilloscope. Once the power traces are collected, our proposed technique is applied and full 128-bit secret key is recovered. To recover 128-bit secret key, 65000 $\approx 2^{16}$ power traces are used and the number of key guess complexity is 2^{12}. Due to the limitation of the space, the PCC of all hypothesis keys for the $K_L^{0,0}$ only is shown in Fig 7. The actual key used to encrypt 65000 random plaintexts is $0x0001$ $0x0203$ $0x0405$ $0x0607$ $0x0809$ $0x0A0B$ $0x0C0D$ $0x0E0F$. The key used in first round function is $0x0001$ $0x0203$. $K_L^{0,0} = 0x0001$ is extracted successfully, as the PCC for hypothesis keys $0x00$ and $0x01$ in MSB and LSB position are maximum, respectively as shown in Fig 7.

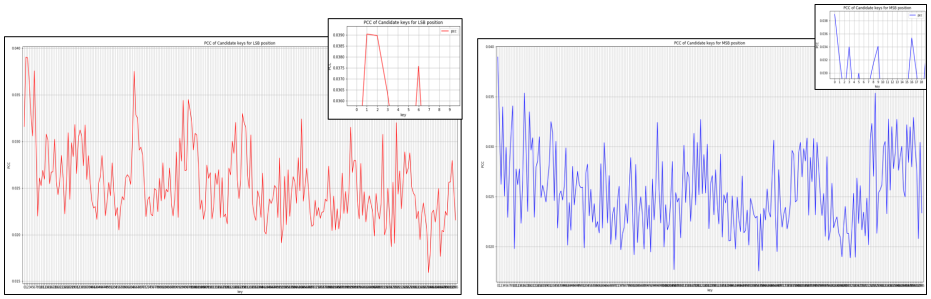

Fig. 7. PCC of all 256 hypothesis keys for LSB and MSB of $K_L^{0,0}$

.

6.2 FPGA Implementation of SPARX-64/128

We implemented the SPARX design as given in Section 4 and Section 5 in Verilog on Xilinx Spartan 3 FPGA using Xilinx ISE 14.7. The serialized implementation

of SPARX requires 32 Flip-Flops and 114 LUTs occupying 60 slices and the maximum speed achieved is 173 MHz. For TI-SPARX 64/128, the addition circuit of serialized implementation is modified and three individual serialized circuit for each share are required. The TI-SPARX 64/128 requires 69 Flip-Flops and 231 LUTs occupying 131 slices and it runs at 144 MHz. SPARX round based design is used in the side channel attack and is also used for comparison. Round based means one step structure of SPARX (three rounds and one linear-mix layer) runs in one clock cycle. Serialized SPARX occupies the least area of all three implementation and the round based occupies the most because round based need large area to implement one step in one clock cycle. The TI-SPARX which is based on serialized implementation occupies lesser area compared to round based, whereas it needs more area than serialized to preserve three properties of threshold implementation. The results are given in Table 2.

Table 2. Resource utilization of different implementations of SPARX 64/128. Flip-flops are used as registers, LUT for logic and shift registers. Slice contains certain number of LUTs, flip-flops and multiplexers

Implementation	Flip-Flops	LUTs	Slices	Speed (MHz)	Throughput (Mbps)
Round based SPARX	2671	4299	2370	204	22.9
Serialized SPARX	32	114	60	173	10.81
TI-SPARX	69	231	131	144	9

Serialized implementation of SPARX is compared with Speck 128/128, Simon 128/128 and PRESENT lightweight ciphers as they are NSA and ISO standard lightweight block ciphers, respectively. SPARX occupies almost 50% less area compared to PRESENT at the expense of lower throughput by 17.59 Mbps. At the same time, SPARX has almost three times higher throughput than Simon 128/128 but 1.7 times the area requirement of Simon 128/128. This is expected because Simon is proposed mainly for hardware. SPARX occupies almost 1.4 times the area required for Speck 128/128. The increase in the area is due to extra linear layer design in SPARX which is not present in Speck 128/128.

Table 3. Comparison of SPARX-64/128 with other ciphers on area and throughput

Cipher	Slices	Throughput (Mbps)	FPGA
TI-SPARX 64/128	**131**	**9**	**xc3s50**
TI Speck 128/128[10]	99	9.68	xc3s50
TI Simon 128/128 [3]	87	3.0	xc3s50
SPARX-64/128	**60**	**10.81**	**xc3s50**
Speck 128/128[10]	43	10.05	xc3s50
Simon 128/128[6]	36	3.6	xc3s50
PRESENT[22]	117	28.4	xc3s50-5

7 Conclusion and Future Work

Although, SPARX design has a provable security bounds against differential and linear cryptanalysis, there is still a margin of analysis to be performed when it comes to side channel cryptanalysis. In this research, we successfully demonstrated that SPARX-64/128 cipher is vulnerable to first order and second order power side channel analysis. We are able to recover full 128-bit secret key of SPARX-64/28 implemented in SAKURA-G board using 2^{12} key guess and $65000 \approx 2^{16}$ power traces. The attack mentioned for SPARX-64/128 worked for two other variants SPARX-128/128 and SPARX-128/256. As a countermeasure to secure SPARX-64/128, we proposed Threshold Implementation to improve the security of the initial SPARX core. First, a serialized implementation of SPARX is given. Then, a first-order side-channel-resilient threshold implementation for SPARX-64/128 is proposed using the previously given serialized implementation. As shown in Section 5, Non-Completeness, Correctness and Balance properties are preserved in the proposed secure TI design to indicate resiliency to side channel analysis.This provides a slower, yet a provably secure version of SPARX. For the future work, we will investigate a generalized model that would extend the threshold implementation to other variants of SPARX cipher. This model would be used to study the complexity analysis and the effectively of using threshold implementations on different lightweight design methodologies.

Acknowledgement This work is supported by Center of Cyber Security Abu Dhabi in NYUAD. The authors would like to acknowledge the support of Dr. K. K. Soundra Pandian and Mohammed Nabeel Thari Moopan.

References

1. `https://www.iso.org/standard/56552.html`
2. Ahmed Abdelkhalek and Mohamed Tolba and Amr M. Youssef: Impossible Differential Attack on Reduced Round SPARX-64/128. In: Progress in Cryptology - AFRICACRYPT 2017 - 9th International Conference on Cryptology in Africa, Dakar, Senegal, 2017. pp. 135–146 (2017)
3. Aria Shahverdi and Mostafa Taha and Thomas Eisenbarth: Silent Simon: A threshold implementation under 100 slices. In: IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2015, Washington, DC, USA, 2015. pp. 1–6 (2015)
4. Aria Shahverdi and Mostafa Taha and Thomas Eisenbarth: Lightweight Side Channel Resistance: Threshold Implementations of Simon. IEEE Trans. Computers **66**(4), 661–671 (2017)
5. Axel Poschmann and Amir Moradi and Khoongming Khoo and Chu-Wee Lim and Huaxiong Wang and San Ling: Side-Channel Resistant Crypto for Less than 2, 300 GE. J. Cryptology **24**(2), 322–345 (2011)
6. Aydin Aysu and Ege Gulcan and Patrick Schaumont: SIMON says: Break area records of block ciphers on fpgas. Embedded Systems Letters **6**(2), 37–40 (2014)
7. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK Families of Lightweight Block Ciphers. Cryptology ePrint Archive, Report 2013/404 (2013)

8. Begül Bilgin and Benedikt Gierlichs and Svetla Nikova and Ventzislav Nikov and Vincent Rijmen: A More Efficient AES Threshold Implementation. In: Progress in Cryptology - AFRICACRYPT 2014 - 7th International Conference on Cryptology in Africa, Marrakesh, Morocco, 2014. pp. 267–284 (2014)
9. Begül Bilgin and Benedikt Gierlichs and Svetla Nikova and Ventzislav Nikov and Vincent Rijmen: Higher-Order Threshold Implementations. In: Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C. pp. 326–343 (2014)
10. Cong Chen and Mehmet Sinan Inci and Mostafa Taha and Thomas Eisenbarth: SpecTre: A Tiny Side-Channel Resistant Speck Core for FPGAs. In: Smart Card Research and Advanced Applications - 15th International Conference, CARDIS 2016, Cannes, France, 2016. pp. 73–88 (2016)
11. Daniel Genkin and Adi Shamir and Eran Tromer: Acoustic Cryptanalysis. J. Cryptology **30**(2), 392–443 (2017)
12. Dillibabu Shanmugam and Ravikumar Selvam and Suganya Annadurai: Differential Power Analysis Attack on SIMON and LED Block Ciphers. In: Security, Privacy, and Applied Cryptography Engineering - 4th International Conference, SPACE 2014, Pune, India, 2014. (2014)
13. Dinu, D., Perrin, L., Udovenko, A., Velichkov, V., Großschädl, J., Biryukov, A.: Design Strategies for ARX with Provable Bounds: Sparx and LAX. In: Advances in Cryptology - ASIACRYPT 2016 (2016)
14. Eli Biham and Adi Shamir: Differential Fault Analysis of Secret Key Cryptosystems. In: Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, 1997. pp. 513–525 (1997)
15. Elisabeth Oswald and Stefan Mangard and Christoph Herbst and Stefan Tillich: Practical Second-Order DPA Attacks for Masked Smart Card Implementations of Block Ciphers. In: The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, 2006. pp. 192–207 (2006)
16. Eric Brier and Christophe Clavier and Francis Olivier: Correlation Power Analysis with a Leakage Model. In: Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, 2004. pp. 16–29 (2004)
17. Goubin, L., Patarin, J.: DES and differential power analysis (the "duplication" method). In: Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, 1999. pp. 158–172 (1999)
18. Joan Daemen and Vincent Rijmen: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography, Springer (2002), https://doi.org/10.1007/978-3-662-04722-4
19. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, 1999. pp. 388–397 (1999)
20. Lo, Owen and Buchanan, William J. and Carson, Douglas: Correlation Power Analysis on the PRESENT Block Cipher on an Embedded Device. In: Proceedings of the 13th International Conference on Availability, Reliability and Security. ARES 2018, ACM (2018)
21. Messerges, T.S.: Securing the AES finalists against power analysis attacks. In: Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, 2000. pp. 150–164 (2000)

22. Panasayya Yalla and Jens-Peter Kaps: Lightweight Cryptography for FPGAs. In: ReConFig'09: 2009 International Conference on Reconfigurable Computing and FPGAs, Cancun, Quintana Roo, Mexico, 2009 (2009)
23. Paul C. Kocher: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, 1996. pp. 104–113 (1996)
24. Quisquater and Jean-Jacques and Samyde, David: ElectroMagnetic Analysis (EMA): Measures and Counter-measures for Smart Cards. In: Smart Card Programming and Security. pp. 200–210. Springer Berlin Heidelberg (2001)
25. Ralph Ankele and Eik List: Differential Cryptanalysis of Round-Reduced Sparx-64/128. In: Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, 2018. pp. 459–475 (2018)
26. Svetla Nikova and Christian Rechberger and Vincent Rijmen: Threshold Implementations Against Side-Channel Attacks and Glitches. In: Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, 2006. pp. 529–545 (2006)
27. Thomas S. Messerges: Using Second-Order Power Analysis to Attack DPA Resistant Software. In: Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, 2000. pp. 238–251 (2000)
28. Tobias Schneider and Amir Moradi and Tim Güneysu: Arithmetic Addition over Boolean Masking - Towards First- and Second-Order Resistance in Hardware. In: Applied Cryptography and Network Security - 13th International Conference, ACNS 2015, New York, NY, USA, 2015. pp. 559–578 (2015)