The ISC Int'l Journal of Information Security
ISeCure

July 2017, Volume 9, Number 2 (pp. 119–130)
http://www.isecure-journal.org

A New CPA Resistant Software Implementation for Symmetric Ciphers with Smoothed Power Consumption: SIMON Case Study[☆]

Morteza Safaei Pour [1], and Mahmoud Salmasizadeh [2,*]
[1] *Sharif University of Technology, Department of Electrical Engineering, Tehran, Iran*
[2] *Sharif University of Technology, Electronics Research Institute, Iran, Tehran*

A R T I C L E I N F O.

Article history:
Received: 27 April 2017
First Revised: 29 July 2017
Last Revised: 30 July 2017
Accepted: 31 July 2017
Published Online: 31 July 2017

Keywords:
Side Channel Attack, DPA, Software Countermeasure, Smoothing Power.

A B S T R A C T

In this paper we propose a new method for applying hiding countermeasure against CPA attacks. This method is for software implementation, based on smoothing power consumption of the device. This method is evaluated on the SIMON scheme as a case study, however, it is not relying on any specific SIMON features. Our new method includes only *AND* equivalent and *XOR* equivalent operations since every cryptographic algorithm can be implemented with two basic operations, namely *AND* and *XOR*. Therefore, hamming weight and hamming distance take constant values at each moment of time. This can decrease data-dependency between processed values and consumed power. In order to practically evaluate the resulting implementation overheads and the resistance improvement against CPA, we implement the proposed coding scheme on SIMON, a lightweight block cipher, on a smart card with the ATmega163 microprocessor. We define resistance as the number of traces, which for less than that number, the correct key cannot be distinguished from all other hypothetical keys by its correlation coefficient in any moment of time. The results of this implementation show 350 times more immunity against correlation attacks.

1 Introduction

Since the first time that differential power analysis (DPA) was proposed by Kocher in 1999 [1], side channel attacks have been an important subject in association with cryptographic devices. In the past two decades, many progresses in power analysis attacks have been achieved. Different classes of countermeasures against power analysis attacks have been proposed [2]. Software based countermeasure methods recently are noticed because these kinds of methods do not need any special hardware or changes in the device architecture. These methods can be implemented in software and on all conventional microprocessors.

The proposed method in this paper tries to smooth the consumed power by considering power models. This is a method for applying hiding countermeasure in the software such that all the intermediate states have constant hamming weight and hamming distance. Hence, in some sense, the information leakage gets lower. Our new method is based on this fact that

☆ This article is an extended version of an ISCISC'13 paper.
* Corresponding author.

Email addresses: safaeipour_morteza@ee.sharif.edu (M. Safaei Pour), salmasi@sharif.edu (M. Salmasizadeh)

ISSN: 2008-2045 © 2017 ISC. All rights reserved.

every cryptographic algorithm can be implemented with only two operands *AND*, *XOR*. Therefore, our new method maps computation to a new domain by defining equivalent *AND* and equivalent *XOR*; these equivalent operands are designed so that the hamming weight and the hamming distance take constant values at each moment of time. As a result, data dependency between computation and power consumption will be decreased. Finally, when computation terminates, output can be decoded and ciphertext will be obtained.

This paper is divided into different sections; Section 2 introduces the related work, as well as a short introduction to the SIMON block cipher that is used for the case study; Section 3 presents implementation of protected and unprotected SIMON using the proposed method; in Section 4 the evaluated results and the data of the correlation power analysis attacks on different implementations are stated and compared with each other. Finally, conclusions are presented in Section 5.

2 Background

This section introduces the related work. Also, a short introduction to the SIMON block cipher that is used for the case study is provided.

2.1 Previous Works

So far various array of countermeasures against differential power attacks have been proposed. Most of them are applied in architecture level or cell levels of a cryptographic device. These countermeasures are complicated, expensive and not suitable for inexpensive industrial applications such as sensors, RFIDs and smart cards. Another main class is masking [3, 4] methods which in spite of their strengths, they are still vulnerable to higher order attacks. Meanwhile, there have been some efforts for proposing new countermeasures based on hiding class in the software and programming level.

In the first attempts, in 2010 Chen [5] tried to utilize DPL (dual rail logic) methods [6] in software on special multi-core microprocessors. They assumed that there are two one-core microprocessors that work parallel and they are precisely synchronized with each other by an additional circuit. On the other hand, in the software, the moment the first microprocessor processes an operation, the second microprocessor processes complement of that operation in that exact moment of time. For instance, if the first microprocessor processes *AND* (*OR*) the second microprocessor should process *OR* (*AND*) operation. The authors claim that this method shows 80 times improvement in the resistance against CPA; however, this method is still vulnerable to EM attacks due to the high correlation of executed intermediate data with consumed power and consequently, with the power of electromagnetic radiation by the two distinctive microprocessor. It also suffers from the synchronization problem that an adversary can use to attack the system. Moreover, this method requires special hardware and additional circuits that forces extra cost and complexity.

In 2011, Hoogvorst [7] proposed a method that used the DPL idea in the software and works on conventional microprocessors. The authors claim that with the help of their special programming schemes and rules, they could decrease dependency between the consumed power and the processed data. To apply an operation like *AND* on two bits a and b, primarily they were encoded and were put together in an 8-bit register like $[0\,0\,0\,0\,\bar{b}\,b\,\bar{a}\,a]$. Then, the code corresponding to the intended operation such as *AND*, *OR*, *XOR*, etc. is placed in the unused four bits of the register; for example $[0\,0\,0\,1\,\bar{b}\,b\,\bar{a}\,a]$ that $[0\,0\,0\,1]$ shows an *AND* operation. The microprocessor reads the value of the memory in the address of $[0\,0\,0\,1\,\bar{b}\,b\,\bar{a}\,a]$ that the value is $[0\,0\,0\,0\,0\,0\,\overline{a.b}\,a.b]$ which is precomputed and placed in memory. With the help of this procedure all the required processes can be obtained. Actually this is similar to what an ALU does in a processor. Also, at the beginning of each step the used register are set to zero which is equivalent to precharge step in DPL. The authors did not present any experimental results in their paper to support their claims, however, Rauzy [8] presented a program to secure the assembly code of a cryptographic cipher implemented with bit-sliced format [9] based on this paper. The name of this program is Paioli. The implementation results on PRESENT [10] show that Hoogvorsts method is 34 times more resistant. The disadvantage of this method is using an 8-bit register for processing an operation on two 1-bit. Also, this method's memory usage highly increases due to the required memory for every inputs and operands. In [8], the authors showed that the least significant bit on ATmega163 microprocessor leaks very differently from the other 7 bits, which approximately have a same leakage characteristics. For this purpose, they ran eight versions of an unprotected bit-sliced implementation of PRESENT, each of them using only one of the 8 possible bits. They used the Normalized Inter-Class Variance(NICV) [11] as a metric to evaluate the leakage level of the variables of each of the 8 versions.

Han in 2012 [12] introduced a bit-wise balanced encoding data representation method. Its general idea is very similar to Hoogvorst's work [7], every bit a encoded to $a\bar{a}$. Then, for every operation like *XOR*, SBOX and permutation, a new equivalent operation is stated. Equivalent of SBOX is achieved straightforwardly by changing the values of look up tables to the encoded ones. They show that for XOR the relation $encode(A \oplus B) = encode(A) \oplus encode(B) \oplus (0101)_2$ is confirmed. Thus, the new XOR defined as $(encode(B) \oplus (0101)_2) \oplus encode(A)$ that is easier than look up table method used in [7]. However, it cause a problem that makes the attack possible against it [13]. If B has a value that depends on the key, the hamming weight of the intermediate value $encode(B) \oplus (0101)_2$ is not constant. Therefore, it is vulnerable to correlation attacks. The authors, implement the method on LBlock [14] and PRESENT [10] as case study algorithms.

In 2014, Servant [13] used (x, y)-codes; it means that all the y-bit words have hamming weight equal to x. Authors use AES for the case study. Each nibble is encoded with this coding system because the calculations in AES are 8-bit based. Each nibble can set to 16 different values. They use $(3, 6)$-codes since the number of these codes are $\binom{6}{3} = 20$. Actually each nibble is encoded to 6-bit words. This method uses lookup tables for all the processes. If the coding is shown by C, an operation by $\perp$ and lookup table by T, the process can be $T[(C(A) \ll 8) \parallel C(B)] = C(A \perp B)$ on a 16-bit processor. The $(\ll 8)$ symbol is an 8-bit left shift. The memory usage to implement AES by this method is $4kB$ whereas for the simple AES implementation is $256B$ and the calculation speed decreases 4.2 times by using this method.

In 2014, Chen [15] proposed a coding system that makes the hamming weight and hamming distance constant. For example two coding that were used mostly in [15] are $enc_I = \bar{b}_3 b_3 \bar{b}_2 b_2 \bar{b}_1 b_1 \bar{b}_0 b_0$ and $enc_{II} = b_0 \bar{b}_2 b_1 b_3 \bar{b}_1 b_2 \bar{b}_0 \bar{b}_3$. This coding system is implemented on the PRINCE block cipher [16]. They proposed different coding systems and methods for each step of PRINCE e.g., addRoundkey, SBOX, MixColumn etc. The disadvantage of this method is the use of different coding systems for each step. The authors claimed that the leakage is mostly because of converting one coding to another coding. Another drawback is that the proposed method is based on PRINCE and cannot be applied on other block ciphers. The coding systems proposed for inputs and outputs of SBOX are just designed in order to make the hamming distance constant for PRINCE's SBOX, so it is not constant for another SBOX, therefore it cannot be applied to other ciphers such as SIMON block cipher.

2.2 SIMON Block Cipher

SIMON is a lightweight block cipher algorithm that was introduced by NSA in 2013 [17]. SIMON2n/mn uses Feistel rule of motion and has two n-bit words and an m-word key. For instance SIMON32/64 has a 32-bit block (consists of a 16-bit word left and a 16-bit word right) and a 64-bit key. Encryption and decryption of this algorithm only consists of three operations: XOR, AND and the j bits left circular shift (S^j). Equation (1) defines each round of SIMON and $F(x)$ is defined in (2). Also, this algorithm is shown in Figure 1.

$$R_k(x_{i+1}, x_i) = (x_i \oplus F(x_{i+1}) \oplus k, x_{i+1}) \quad (1)$$

$$F(x) = ((S^8 x) \& (S^1 x)) \oplus (S^2 x) \quad (2)$$

A detailed description for this algorithm and key scheduling can be found in [17].

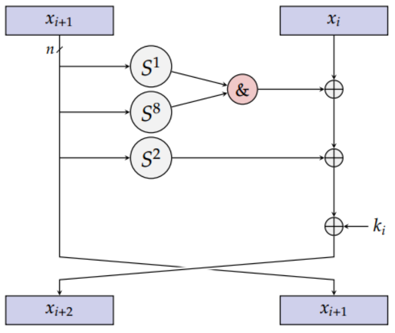


Figure 1. SIMON round function

3 Proposed Schemes and Countermeasures

In this section the proposed method is introduced to make cryptographic implementations more resistant against CPA. This method is trying to decrease leakage and correlation between the consumed power and the cryptographic key during encryption by making the consumed power even more constant. The proposed method uses an encoding scheme in which all the steps and operations have constant hamming weight and hamming distance.

An encoding system is needed in which equivalent operations fulfil the conditions about constant hamming weight and hamming distance at each moment. In order to keep consistency with previous works, we use the word “coding instead of mere suitable word “mapping. Primarily, plaintext and cryptographic key are encoded based on the proposed encoding scheme, both of them are mapped to the new domain as shown in Figure 2. It is known that each block cipher can be implemented only by basic operations such as AND, XOR and NOT. For instance, bit-slice implementation uses just basic operations [9]. Therefore, if equivalent of basic operations are defined in this new domain, any block cipher can be implemented with this method. Finally, the equivalent ciphertext in the new domain should be decoded and the original

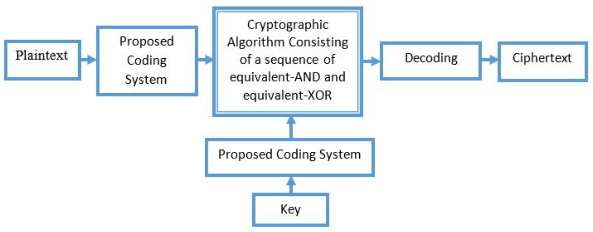

Plaintext
Proposed Coding System
Cryptographic Algorithm Consisting of a sequence of equivalent-AND and equivalent-XOR
Decoding
Ciphertext
Proposed Coding System
Key

Figure 2. Outline of the proposed scheme

Input bit	Set A	Set B	Set C
x	$\bar{x}\bar{x}xx$	$\bar{x}x\bar{x}x$	$x\bar{x}\bar{x}x$

Table 1. Three sets A, B and C of proposed coding system

Input bit	Corresponding member in set A	Corresponding member in set B	Corresponding member in set C
0	1100	1010	0110
1	0011	0101	1001

Table 2. Mapping of bit zero and one to three sets A,B and C

ciphertext will be obtained. This is the overview of what proposed coding consists of. In the following we explain each part in detail. For the proposed coding, three sets A, B and C are defined as stated in Table 1. There are an equivalent member for bit 0 and an equivalent member for bit 1 in each set. With this coding system one bit is encoded to 4 bits. In set B, assuming we set $x = 1$, the mapping of bit 1 in set B will result in $\bar{1}1\bar{1}1 = 0101$. For more illustration the mapping of bit 0 and 1 in three sets A, B and C are presented in Table 2. If $\mathbf{a} = \bar{a}\bar{a}aa \in A$, $\mathbf{b} = \bar{b}b\bar{b}b \in B$, $\mathbf{c} = c\bar{c}\bar{c}c \in C$ and $a, b, c \in \{0, 1\}$ then;

$$\mathrm{HW}(\mathbf{a}) = \mathrm{HW}(\mathbf{b}) = \mathrm{HW}(\mathbf{c}) = 2 \tag{3}$$

$$\mathrm{HD}(\mathbf{a}, \mathbf{b}) = \mathrm{HW}(\mathbf{a} \oplus \mathbf{b}) = \mathrm{HW}(a \oplus b\ \overline{a \oplus b}\ \overline{a \oplus b}\ a \oplus b) = 2 \tag{4}$$

$$\mathrm{HD}(\mathbf{a}, \mathbf{c}) = \mathrm{HW}(\mathbf{a} \oplus \mathbf{c}) = \mathrm{HW}(\overline{a \oplus c}\ a \oplus c\ \overline{a \oplus c}\ a \oplus c) = 2 \tag{5}$$

$$\mathrm{HD}(\mathbf{b}, \mathbf{c}) = \mathrm{HW}(\mathbf{b} \oplus \mathbf{c}) = \mathrm{HW}(\overline{b \oplus c}\ \overline{b \oplus c}\ b \oplus c\ b \oplus c) = 2 \tag{6}$$

Thus based on (3) hamming weight of all the members in these three sets equals to 2 and based on (4), (5) and (6) hamming distance of every two members from two different sets equals to 2. Next step is to define *equivalent-XOR* and *equivalent-AND* such that the result of *equivalent-XOR*$(\mathbf{e}, \mathbf{d})$ and *equivalent-AND*$(\mathbf{e}, \mathbf{d})$ for every two members $\mathbf{e}$ and $\mathbf{d}$ from two different sets be in third set like in (7) and (8).

$$\mathbf{a} \oplus \mathbf{b} = \mathbf{c} \rightarrow \textit{equivalent-XOR}(\mathbf{a}, \mathbf{b}) = \mathbf{c} \in C \tag{7}$$

$$\mathbf{a}.\mathbf{b} = \mathbf{c} \rightarrow \textit{equivalent-AND}(\mathbf{a}, \mathbf{b}) = \mathbf{c} \in C \tag{8}$$

Equivalent-XOR. Considering properties of three sets A, B, C and Equations (3), (4), (5) and (6), *equivalent-XOR* is the same as normal bitwise *XOR*. If $\mathbf{a} = \bar{a}\bar{a}aa \in A, \mathbf{b} = \bar{b}b\bar{b}b \in B, \mathbf{c} = c\bar{c}\bar{c}c \in C$ such that $a, b, c \in \{0, 1\}$ then:

$$\mathbf{a} \oplus \mathbf{b} = \bar{a}\bar{a}aa \oplus \bar{b}b\bar{b}b = a \oplus b\,\overline{a \oplus b}\,\overline{a \oplus b}\,a \oplus b \in C \tag{9}$$

$$\mathbf{a} \oplus \mathbf{c} = \bar{a}\bar{a}aa \oplus c\bar{c}\bar{c}c = \overline{a \oplus c}\,a \oplus c\,\overline{a \oplus b}\,a \oplus b \in B \tag{10}$$

$$\mathbf{b} \oplus \mathbf{c} = \bar{b}b\bar{b}b \oplus c\bar{c}\bar{c}c = \overline{b \oplus c}\,\overline{b \oplus c}\,b \oplus c\,b \oplus c \in A \tag{11}$$

The result of the *XOR* of every two members from two different sets will be in the third set. For example, *XOR* of two bits 1 and 1 equals $1 \oplus 1 = 0$, equivalently it can be calculated by *XOR* 0011 in set A (mapping of bit 1 in set A) and 0101 in set B (mapping of bit 1 in set B) thus the result is 0110 in the set C (mapping of bit 0 in set C). Therefore, the *equivalent-XOR* is normal bitwise *XOR*. It should be noted that *equivalentXOR* must not be applied on two members from a same set.

With the help of this property, one member from each set can be converted to the corresponding member in another set. As an illustration, every member of set A could be XORed by mapping of bit 0 in set B to produce the corresponding member in set C. For instance, *XOR* of the mapping of bit 1 in set A (0011) with the mapping of bit 0 in set B (1010) equals to 1001 that is the mapping of 1 in set C. Hence, one member can be converted to the corresponding member in another set by using this method.

Equivalent-NOT. Considering Table 1 and 2, it is concluded that *equivalent-NOT* is exactly the same as normal bitwise *NOT*. Thus complementing a member (corresponding to bit a) from each set produces another member (corresponding to bit $\bar{a}$) of the same set as illustrated in Equations (12),(13) and (14). The hamming weight and the hamming distant in this situation is always constant and equal to 4. Therefore, it fulfils our intended requirements.

$$a \in \{0, 1\} \rightarrow \mathbf{a} = \bar{a}\bar{a}aa \in A \rightarrow \bar{\mathbf{a}} = \overline{\bar{a}\bar{a}aa} = aa\bar{a}\bar{a} \in A \tag{12}$$

$$b \in \{0, 1\} \rightarrow \mathbf{b} = \bar{b}b\bar{b}b \in B \rightarrow \bar{\mathbf{b}} = \overline{\bar{b}b\bar{b}b} = b\bar{b}b\bar{b} \in B \tag{13}$$

$$c \in \{0, 1\} \rightarrow \mathbf{c} = c\bar{c}\bar{c}c \in C \rightarrow \bar{\mathbf{c}} = \overline{c\bar{c}\bar{c}c} = \bar{c}cc\bar{c} \in C \tag{14}$$

For example: $a = 1 \rightarrow \mathbf{a} = 0011 \in A \rightarrow \bar{\mathbf{a}} = \overline{0011} = 1100 \in A \rightarrow 0$ in set A

However an *XOR* operation with corresponding member of bit 1 in another set can be used as an *equivalent-NOT* operation too, but using simple complementing of a member is more convenient and simple while XORing needs more clock cycles and steps.

Equivalent-AND. Contrary to *equivalent-XOR*, *equivalent-AND* is more complicated because *AND* of two members of two different sets is not in any set. Following, two methods are proposed to calculate *equivalent-AND* that fulfil our conditions.

• Method-1 for *equivalent-AND* In the Figure 3 method-1 for *equivalent-AND* presented. This method is implemented only with *XOR* and LUT. Two members from two different sets are considered (for example $\mathbf{a} \in A$ and $\mathbf{b} \in B$). In (I) and (II), with the help of LUT1 and LUT2, $\mathbf{a}^{'}$ and $\mathbf{b}^{'}$ obtained. In (III), *XOR* of these two values were calculated such that $\mathbf{d}$ in (IV) was achieved. Finally in (V), LUT3 takes $\mathbf{d}$ as an input and produces $\mathbf{c}_{and}$. Therefore, with the help of Equations (15) and (16) the hamming weight and the hamming distance in each step are constant.

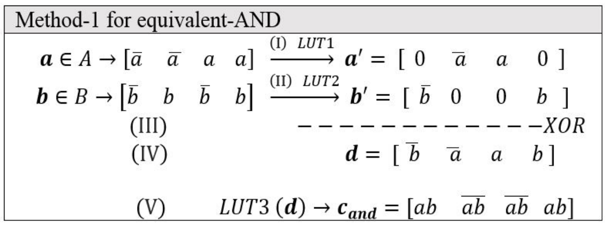

Method-1 for equivalent-AND
$\boldsymbol{a} \in A \rightarrow [\bar{a} \quad \bar{a} \quad a \quad a] \xrightarrow[]{(I)\ LUT1} \boldsymbol{a}' = [\,0 \quad \bar{a} \quad a \quad 0\,]$
$\boldsymbol{b} \in B \rightarrow [\bar{b} \quad b \quad \bar{b} \quad b] \xrightarrow[]{(II)\ LUT2} \boldsymbol{b}' = [\,\bar{b} \quad 0 \quad 0 \quad b\,]$
(III) ------------XOR
(IV) $\boldsymbol{d} = [\,\bar{b} \quad \bar{a} \quad a \quad b\,]$
(V) $LUT3\ (\boldsymbol{d}) \rightarrow \boldsymbol{c}_{and} = [ab \quad \overline{ab} \quad \overline{ab} \quad ab]$

Figure 3. Steps of method-1 for *equivalent-AND*

$$\mathrm{HD}(\mathbf{a},\mathbf{a}^{'}) = \mathrm{HD}(\mathbf{b},\mathbf{b}^{'}) = \mathrm{HD}(\mathbf{a}^{'},\mathbf{d}) = \mathrm{HD}(\mathbf{b}^{'},\mathbf{d}) = 1 \quad (15)$$
$$\mathrm{HD}(\mathbf{d},\mathbf{c_{and}}) = 2 \quad (16)$$

• Method-2 for *equivalent-AND*. In Figure 4, method-2 for *equivalent-AND* is presented. This method is implemented only with *XOR* and LUT. This method is only different to method-1 in the last step. Two members from two different sets are considered (for example $\mathbf{a} \in A$ and $\mathbf{b} \in B$). In (I) and (II), with the help of LUT1 and LUT2, $\mathbf{a}'$ and $\mathbf{b}'$ obtained. In (III), *XOR* of these two values were calculated such that $\mathbf{d}$ in (IV) achieved. Finally, in (V) the output of the LUT4 and LUT5 are XORed to produce $\mathbf{c}_{and}$. Input and output of LUT4 and LUT5 shown in Table 3.

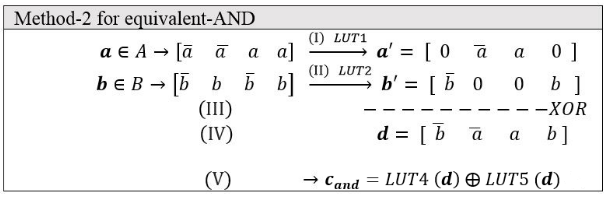

Method-2 for equivalent-AND
$\boldsymbol{a} \in A \rightarrow [\bar{a} \quad \bar{a} \quad a \quad a] \xrightarrow[]{(I)\ LUT1} \boldsymbol{a}' = [\,0 \quad \bar{a} \quad a \quad 0\,]$
$\boldsymbol{b} \in B \rightarrow [\bar{b} \quad b \quad \bar{b} \quad b] \xrightarrow[]{(II)\ LUT2} \boldsymbol{b}' = [\,\bar{b} \quad 0 \quad 0 \quad b\,]$
(III) ----------XOR
(IV) $\boldsymbol{d} = [\,\bar{b} \quad \bar{a} \quad a \quad b\,]$
(V) $\rightarrow \boldsymbol{c}_{and} = LUT4\ (\boldsymbol{d}) \oplus LUT5\ (\boldsymbol{d})$

Figure 4. Steps of method-2 for *equivalent-AND*

$$\mathrm{HD}(\mathbf{d},\mathrm{LUT4}(\mathbf{d})) = 1 \quad (17)$$
$$\mathrm{HD}(\mathbf{d},\mathrm{LUT5}(\mathbf{d})) = 1 \quad (18)$$

$a\ b$	$\mathbf{d} = [\bar{b}\,\bar{a}\,a\,b]$	$LUT4(\mathbf{d})$	$LUT5(\mathbf{d})$	$LUT4(\mathbf{d}) \oplus LUT5(\mathbf{d})$
0 0	[1 1 0 0]	[1 1 0 1]	[1 0 1 1]	[0 1 1 0]
0 1	[1 0 1 0]	[1 0 1 1]	[1 1 0 1]	[0 1 1 0]
1 0	[0 1 0 1]	[1 1 0 1]	[1 0 1 1]	[0 1 1 0]
1 1	[0 0 1 1]	[0 1 1 1]	[1 1 1 0]	[1 0 0 1]

Table 3. Input and output values of LUT4 and LUT5

Based on Equations (17) and (18), the hamming weight and the hamming distance in each step is constant. In Figure 4 the procedure for two members from two sets A and B are shown; but it can be easily generalized for the members from other sets. In order to explicitly clarify which register each data is stored in, the assembly code of the *equivalent-AND* method-2 implementation is shown in Table A.1 in Appendix B. In this table, $A1$ and $A2$ are two names for two specific registers of our microprocessor (for instance $r1$ and $r2$). Assume $A1 = [\bar{b}_0\, b_0\, \bar{b}_0\, b_0\, \bar{a}_0\, \bar{a}_0\, a_0\, a_0] \in [B|A]$ and $A2 = [\bar{a}_1\, \bar{a}_1\, a_1\, a_1\, \bar{b}_1\, b_1\, \bar{b}_1\, b_1] \in [A|B]$. The required sequence of operations are stated in Table A.1. This table by specifying the changed register and providing the corresponding proofs, illustrate that in each step the HW and HD are always constant. Also, this registers can be used again for calculating *equivalent-AND* of two new values without any leakage.

4 Evaluation Methodology

To evaluate any proposed method, it should be applied on a cryptographic algorithm and compared to unprotected cases. The unprotected case means that the steps are exactly the same as protected implementation and only coding system and intermediate values are different. In this paper the algorithm SIMON32/64 is used for the case study because it is a well-known lightweight lock cipher, only composed of *XOR*, *AND* and the circular left shift. SIMON32/64 has 32-bit block and cryptographic key length 64-bit.

In the proposed method each bit is coded to four bits. Therefore, each 2-bit can be put in an 8-bit register. First, 32-bit plaintext is encoded; because the plaintext is not sensitive and it is not related to the key, it can be encoded easily with no leakage. For each round, the round key is calculated in a conventional way, then it is encoded based on the proposed method. For a fixed key all the round keys are fixed too, thus encoding the round keys do not make any problem and vulnerability against DPA and CPA.

Applying *equivalent-XOR* and *equivalent-AND* based on explanations in Section 3 are convenient but circular shifts in SIMON32/64 algorithm can change the hamming distance and increase the leakage. It is possible to implement bit-sliced SIMON32/64. In bit-sliced implementation of a cryptographic algorithm there is no need for shift operations. But in this paper a novel implementation is proposed. In this novel implementation first the input data reordered with a specific pattern so that there is no need for the circular shift operation and just the order of reading from the memory change.

Assume that $p_1, p_2, \ldots, p_{16}$ are 16-bit of left block of data. Each bit is encoded to four bits. The encoding is shown by $\tau(\cdot)$. In order to apply circular shift easily, the data block is reordered and put in 8-bit registers. Each 8-bit register is shown with [. | .] sign. In Figure 5 the reordered data and its left circular shifts are illustrated. By using this order of bits, there is no need for circular shift anymore and just the order changes in reading the registers. Bytes specified with underline need an additional swap operation that with proper set assigning of data which is explained in the next paragraph, cause no leakage.

$$[\tau(p_{16})|\tau(p_8)][\tau(p_{15})|\tau(p_7)][\tau(p_{14})|\tau(p_6)][\tau(p_{13})|\tau(p_5)]\ldots$$
$$\ldots[\tau(p_{12})|\tau(p_4)][\tau(p_{11})|\tau(p_3)][\tau(p_{10})|\tau(p_2)][\tau(p_9)|\tau(p_1)]$$

$$\text{For } S^1\text{:}\quad [\tau(p_{15})|\tau(p_7)][\tau(p_{14})|\tau(p_6)][\tau(p_{13})|\tau(p_5)][\tau(p_{12})|\tau(p_4)]\ldots$$
$$\ldots[\tau(p_{11})|\tau(p_3)][\tau(p_{10})|\tau(p_2)][\tau(p_9)|\tau(p_1)]\underline{[\tau(p_8)|\tau(p_{16})]}$$

$$\text{For } S^8\text{:}\quad \underline{[\tau(p_8)|\tau(p_{16})][\tau(p_7)|\tau(p_{15})][\tau(p_6)|\tau(p_{14})][\tau(p_5)|\tau(p_{13})]}\ldots$$
$$\ldots\underline{[\tau(p_4)|\tau(p_{12})][\tau(p_3)|\tau(p_{11})][\tau(p_2)|\tau(p_{10})][\tau(p_1)|\tau(p_9)]}$$

$$\text{For } S^2\text{:}\quad [\tau(p_{14})|\tau(p_6)][\tau(p_{13})|\tau(p_5)][\tau(p_{12})|\tau(p_4)][\tau(p_{11})|\tau(p_3)]\ldots$$
$$\ldots[\tau(p_{10})|\tau(p_2)][\tau(p_9)|\tau(p_1)]\underline{[\tau(p_8)|\tau(p_{16})][\tau(p_7)|\tau(p_{15})]}$$

Figure 5. Reordered data in input block and after different number of left circular shifts

Another issue is to determine the sets that should be assigned to the intermediate data. After studying different possibilities, assigning $[B \mid A]$ to data and round keys (to each 2-bit) are decided. There is no problem if a swap operation is needed because every two members from two different sets have constant hamming distance. Two bytes $[A_1 \mid B_1]$ and its swap $[B_1 \mid A_1]$ would have a constant hamming weight and hamming distance 4, since $\text{HW}([A_1 \mid B_1]) = \text{HW}([B_1 \mid A_1]) = \text{HW}([A_1 \oplus B_1 \mid B_1 \oplus A_1]) = 4$. Assigned sets for each encoded 4-bit in an 8-bit register for different steps of a round are presented in Figure 6. In the last step of each round, there is a conversion from $[A \mid B]$ to $[B \mid A]$ that is applied by a simple *XOR* with the constant $0x66$. It only changes the assigned sets and it does not change the original data values.

5 Evaluation Results

There are two general methods for evaluating the countermeasure methods against CPA and DPA attacks. The normal aspect is based on comparing the maximum absolute value of correlation coefficient ver-

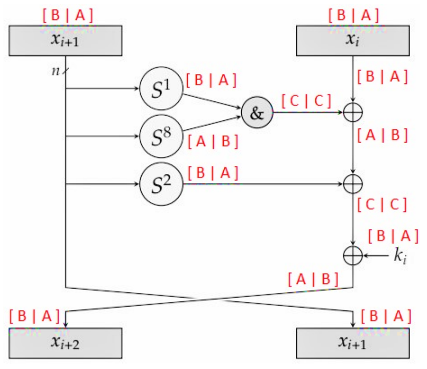


Figure 6. Determination of each set that should assigned to the intermediate data

sus number of power traces in the specific time interval [13, 15]. In this aspect the attacker does not have knowledge about exact moment where the attacks apply. The number of power traces that for more traces, the maximum correlation coefficient of the correct key is more than the corresponding coefficient of all other hypothetical keys, is the minimum number of traces required for identifying the correct key. Thus, this number can be used for comparing two different implementations.

Another method is when the attacker has knowledge about the exact moment of leakage [8]. In this case, if at any moment of time, the correlation coefficient of the correct key is more than the correlation coefficient of all the other hypothetical keys, correct key can be extracted. Hence, the minimum number of power traces required to extract the correct key based on this assumption can be used for comparing two different implementations. The experimental setup for this paper is explained in detail in Appendix A.

5.1 Signal to Noise Ratio Measurement

The aim of the hiding methods is reduction of the signal to noise ratio by decreasing the signal variation. Actually, the SNR is closely related to the information leakage. In power traces, anywhere that have a higher SNR, the leakage and effectiveness of the power analysis attacks is more probable. Thus, the SNR would be a proper factor for comparing security of various implementations. SNR could be calculated based on (19) [2].

$$\begin{aligned}\text{SNR} &= \frac{Var(P_{Signal})}{Var(P_{noise})}\\ &= \frac{Var(P_{data-dependent} + P_{op-dependent})}{Var(P_{el.noise} + P_{sw.noise})}\end{aligned} \qquad (19)$$

$P_{data-dependent}$ is the power dependent on the processed data and $P_{op-dependent}$ is the power dependent on the type of the operation. $P_{el.noise}$ is the electrical noise in the circuit and $P_{sw.noise}$ is the switching noise of other bits that are not the target of attack, therefore, because in our experiment attacks apply on the whole byte the $P_{sw.nosie}$ is equal zero. Also, because in all the traces, the operations are same (at each specific time) the valuse of $P_{op-dependent}$ is equal zero too. Thus, the Equation (19) is reduced to (20).

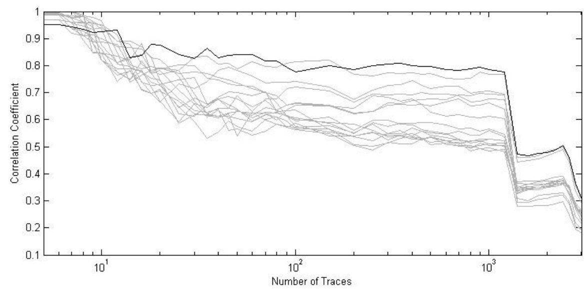
$$\mathrm{SNR} = \frac{Var(P_{data-dependent})}{Var(P_{el.noise})} \tag{20}$$

This way of measuring the SNR is related to the location of the attack. Assume that we want to calculate the SNR of the fifth byte during the second round. For measuring P_{noise}, we choose a key and a plaintext, measure and collect about 10000 power traces during encryption. In this situation collected traces are only depend on noise. Then, for each moment, variance of 10000 corresponding points are calculated. The result is an array that shows the noise variance for all the moments.

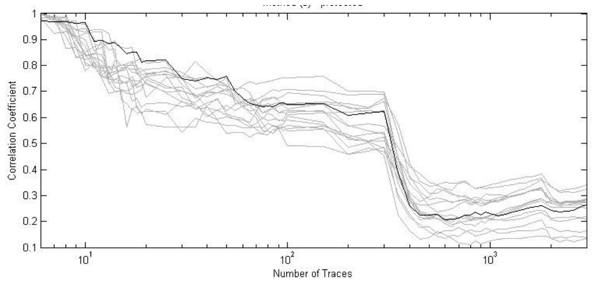
For measuring $P_{data-dependent}$, we keep the key constant and generate high number of random plaintexts (approximately 60000), send them to the smart card and collect corresponding power traces. Based on the key and each plaintext, we calculate the targeted value (for instance the value of fifth byte after *AND* operation in round 2). This value can take up to 4 different values. Then, we divide power traces based on these values and put them in one of the four groups. For each group we calculate the average, then we calculate the variance of 4 resulted average arrays. Therefore, for calculating SNR of each moment, we just divide the variance of power over the variance of noise in that moment.

5.2 Experimental Results

All the methods were implemented on the smart card, the traces were captured and the CPA attack applied on all of them. We conducted the experiment many times for different keys and plaintexts but since the results are almost close to each other, for convenient, we provide one of the experiment's results in the context of this paper. The experimental results are as follows. In Figure 7a and 7b maximum absolute value of the correlation coefficient curves explained respectively, for unprotected implementation and protected implementation with the proposed encoding scheme by *equivalent-AND* method-1. The attacks applied to seventh byte because after attacking on all the bytes, this byte shows lower resistance and security improvement. The unprotected implementation is very vulnerable and attacker with just 11 power traces can identify the correct key. But the protected implementation is more resistant against CPA and with 3000 power traces still the correct key is not identifiable. Thus with the first aspect, there are more than



(a) Unprotected

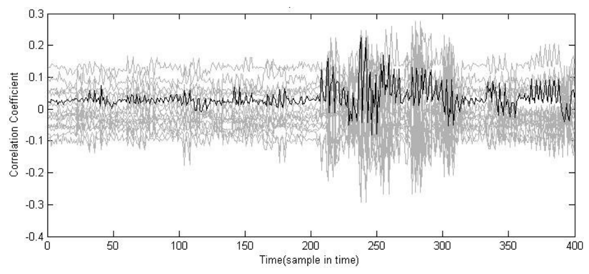


(b) Protected

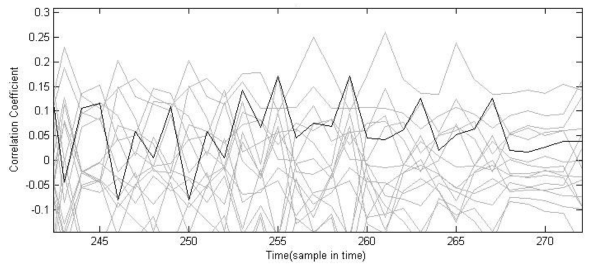
Figure 7. Maximum absolute value of the correlation coefficient curves for implementations with proposed coding system and *equivalent-AND* method-1

300 times improvement when proposed coding with *equivalent-AND* method-1 is used. The correlation coefficient curves of protected implementation with proposed coding and *equivalent-AND* method-1 450 power traces can be seen in Figure 8a. As you can see in Figure 8b, 450 power traces is the minimum number of traces that for the first time, correlation coefficient for the correct key in one moment is more than other hypothetical keys. This corresponding number for unprotected implementation is about 10. Therefore with the strict aspect, the security improvement against CPA is about 45 times. In Figure 9b and 9a, SNR for protected implementation with the proposed coding and *equivalent-AND* method-1 and corresponding unprotected implementation is shown. The SNR of the protected implementation is much lower than the SNR of the unprotected implementation.

In Figure 10a and 10b maximum absolute value of the correlation coefficient curves of the fifth byte respectively, for unprotected implementation and protected implementation with the proposed coding system and *equivalent-AND* method-2 illustrated. The correct key can be identified with nearly 1000 power traces in unprotected implementation. But the protected implementation is more resistant against CPA and with 6500 power traces still the correct key is not identifiable. You can see the correlation coefficient curves of protected implementation with proposed coding and *equivalent-AND* method-2 for 3000 and 3500 power traces respectively in Figure 11a and 11b. As you can see in Figure 11c, 3500 power traces

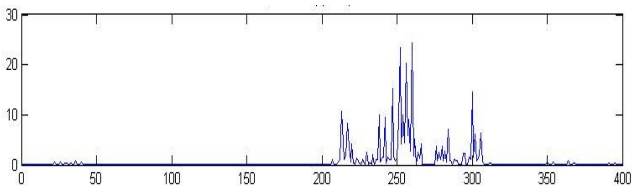


(a) Protected for 450 power traces

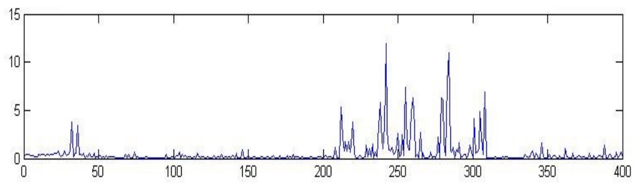


(b) Protected for 450 power traces (zoomed)

Figure 8. The correlation coefficient curves for implementations with proposed coding and *equivalent-AND* method-1

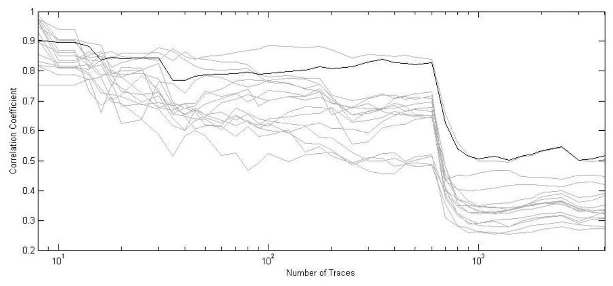


(a) Unprotected

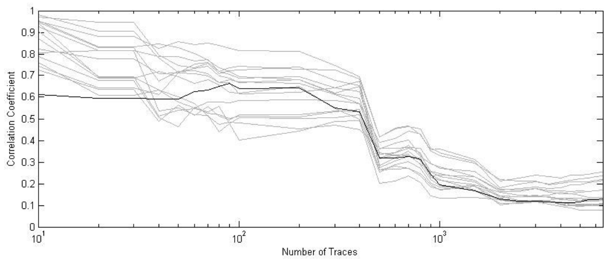


(b) Protected

Figure 9. SNR for a round of SIMON32/64 implementations with proposed coding and *equivalent-AND* method-1



(a) Unprotected



(b) Protected

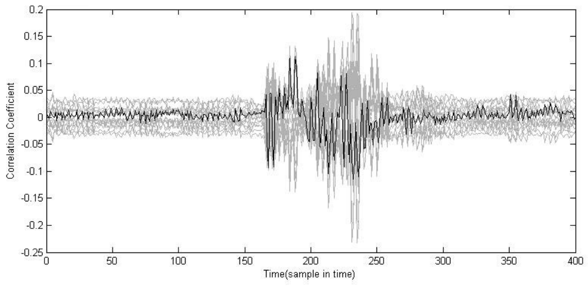
Figure 10. Maximum absolute value of the correlation coefficient curves for implementations with proposed coding system and *equivalent-AND* method-2

is the minimum number of traces that for the first time, correlation coefficient for the correct key in one moment is more than other hypothetical keys. This corresponding number for unprotected implementation is around 10. Therefore with the strict aspect, the security improvement against CPA is about 45. In Figure 12, SNR for protected implementation with proposed coding and *equivalent-AND* method-2 and corresponding unprotected implementation is shown. The SNR of the protected implementation is much lower than the SNR of the unprotected implementation and the protected implementation with *equivalent-AND* method-1. This is because of the LUT4 and LUT5 in *equivalent-AND* method-2 have more even uniform distribution in comparison to the LUT3 in *equivalent-AND* method-1 input-output distribution.

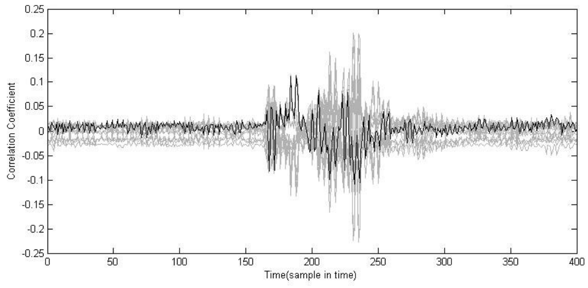
The leakage of the AND operation for this microprocessor is very high even when the hamming weight and the hamming distance is constant in each step. To study the leakage of the AND operation, the proposed coding with *equivalent-AND* method-2 is implemented differently. AND operation is substituted with constant $0x96$ and $0x69$ with respectively the LUTs in step (I) and (II) in *equivalent-AND* method-2 as stated in Figure 13. In Figure 14 the correlation coefficient curves for this implementation and for 3000 power traces illustrated. Comparing this figure with Figure 11a, we see that the correlation coefficient is lower than 0.2 in Figure 11a whereas the correlation coefficient is about 0.6 in Figure 14 that shows high level of leakage for AND operation compared to LUTs and reading from memory in equal conditions.

In Table 4 the number of clock cycles for protected implementation of the proposed coding with *equivalent-AND* method-1 and method-2 and simple implementation of SIMON32/64 without using any coding system or protection are presented. The proportion of the protected implementations clock cycles to the simple implementation clock cycles are respectively 2.23 and 2.43 for *equivalent-AND* method-1 and method-2. Also, the memory needed for these implementations is presented in Table 4.

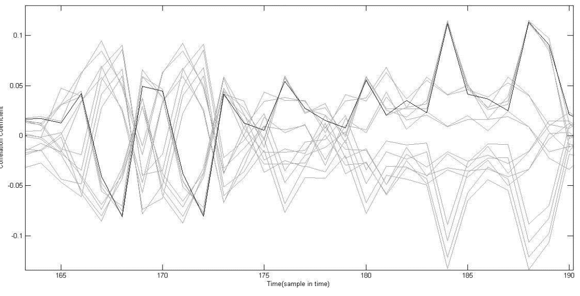
Table 5 states the results of the proposed methods in this paper and the methods studied in [8]. Because in [8] the same strict aspect is used, comparison is



(a) Protected for 3000 power traces

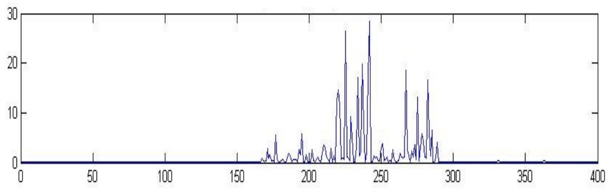


(b) Protected for 3500 power traces

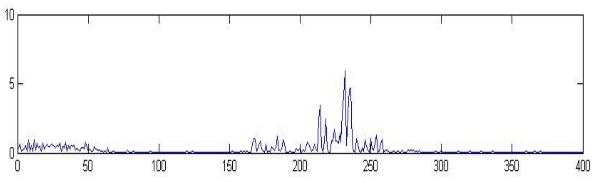


(c) Protected for 3500 power traces (zoomed)

Figure 11. The correlation coefficient curves for implementations with proposed coding and *equivalent-AND* method-2

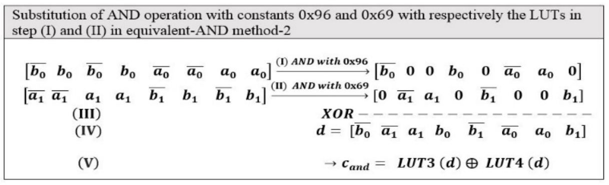


(a) Unprotected



(b) Protected

Figure 12. SNR for a round of SIMON32/64 implementations with proposed coding and *equivalent-AND* method-2


Substitution of AND operation with constants 0x96 and 0x69 with respectively the LUTs in step (I) and (II) in equivalent-AND method-2

$$[\overline{b_0}\ b_0\ \overline{b_0}\ b_0\ \overline{a_0}\ \overline{a_0}\ a_0\ a_0] \xrightarrow{\text{(I) } AND \text{ with } 0\text{x}96} [\overline{b_0}\ 0\ 0\ b_0\ 0\ \overline{a_0}\ a_0\ 0]$$

$$[\overline{a_1}\ \overline{a_1}\ a_1\ a_1\ \overline{b_1}\ b_1\ \overline{b_1}\ b_1] \xrightarrow{\text{(II) } AND \text{ with } 0\text{x}69} [0\ \overline{a_1}\ a_1\ 0\ \overline{b_1}\ 0\ 0\ b_1]$$

(III) XOR ----------------

(IV) $d = [\overline{b_0}\ \overline{a_1}\ a_1\ b_0\ \overline{b_1}\ \overline{a_0}\ a_0\ b_1]$

(V) $\rightarrow c_{and} = LUT3\,(d) \oplus LUT4\,(d)$

Figure 13. Substitution of *AND* with constants $0x96$ and $0x69$ with respectively the LUTs in step (I) and (II) in *equivalent-AND* method-2

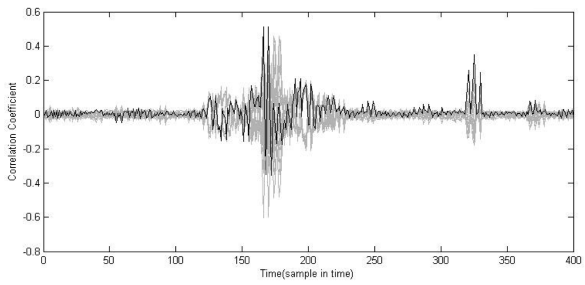


Figure 14. The correlation coefficient curves for new implementation *AND* was substituted with LUTs, for 3000 power traces

	Simple	Method-1	method-2
Clock Cycles	6681	14894	16238
Memory (byte)	32	716	794

Table 4. The number of clock cycles and memory usage of simple (no coding system), proposed coding with *equivalent-AND* method-1 and method-2 implementations of SIMON32/64 algorithm.

more meaningful between these methods, also, the methods implemented on a same microprocessor. The improvement of the security against CPA for proposed methods in this paper is more than the improvement of the security in [8]. Also, proposed methods in this paper have better ratio of speed (ratio of needed clock cycle for protected implementation to simple and without any coding system implementation) in comparison to methods in [8] but methods in [8] have less additional memory usage.

Table 5 presents the security improvement and speed reduction of each methods proportional to the unprotected implementation of that specific cryptographic algorithms. Actually, they are compared with themselves and the amounts of improvements or reductions, compared among various methods. If we use the same algorithm as the previous work, still different implementation would change the result a little. We try to use an implementation of SIMON that is more efficient and compatible with our proposed methods which shows the advantage of them. Second, the additional memory row, shows the required additional memory not overall memory usage. Therefore,

	Method-1 (SIMON32/64)	Method-2 (SIMON32/64)	[8] first 2 bit (PRESENT)	[8] bit 2 and 3 (PRESENT)
Ratio of security improvement	45	350	10	34
Speed reduction ratio	2.23	2.43	3	3
Additional memory (byte)	+684	+762	+64	+64

Table 5. The results of the proposed methods and the method studied in [8].

it is independent of used cryptographic algorithms.

6 Conclusion

We proposed a new method for software implementation of symmetric cipher algorithms based on the idea of the DPL circuits. Our method preserves the hamming weight and the hamming distance by introducing a new encoding scheme and defining equivalent operations. This method makes the consumed power smoother. The main advantage of this method is that it can be applied on any symmetric block cipher and it is not limited to a specific cryptographic algorithm. Also, it leads to lower SNR that would help the security of the system. Masking methods are vulnerable to higher order attacks when SNR is relatively high. Combining proposed method as a way for reducing SNR with masking would be an effective solution.

In the proposed method, we used only one coding system. Consequently, it avoids leakage from changing between different coding systems like the problem arises in [5]. Another advantage of the proposed method is that we could use it to make a program to get the assembly code of a bit-sliced implemented cryptographic algorithm and automatically, apply the proposed method on it to produce a secure implementation assembly code.

In order to evaluate the improvement of proposed method, we implemented the cryptographic algorithm SIMON with proposed methods in smart card with microprocessor ATmega163 for a sufficient number of tests. The results show that the *AND* operation has high leakage even when the hamming weight and the hamming distance is constant in each step. Therefore,*AND* operation is not used in *equivalent-AND* method-1 and 2, instead a lookup table is replaced with *AND* operations. With the strict aspect that the correlation coefficient of the correct key should not be more than the correlation coefficient of all the other hypothetical keys at any moment of time, the security improvement against CPA for the proposed coding system with *equivalent-AND* method-1 is 45 times and method-2 is 350 times in comparison with the corresponding unprotected implementations.

It should be mentioned that as stated in [8], leakage characterization in not the same for each of the eight bits in ATmega163 microprocessor. Whereas the same leakage characterization for different bits are assumed for proposing these methods. Hence, it is expected that the proposed coding system with two *equivalent-AND* methods have better results if they are implemented on another microprocessor with the same leakage characteristics for all the 8 bits. One flaw of our proposed schemes is that we do not consider status flags in our calculation for making the hamming weight and the hamming distance constant. It has bad effects on both of the proposed methods. For instance XOR of the LUT4(d) and LUT5(d) in some cases can set the “N flag in the status register which leads to more leakage. This could be avoided by using only 4 bits of eight bits of 8-bit registers and consequently more overheads.

References

[1] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual International Cryptology Conference*, pages 388–397. Springer, 1999.

[2] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*, volume 31. Springer Science & Business Media, 2008.

[3] Suresh Chari, Charanjit S Jutla, Josyula R Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *Annual International Cryptology Conference*, pages 398–412. Springer, 1999.

[4] Louis Goubin and Jacques Patarin. Des and differential power analysis the duplication method. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 158–172. Springer, 1999.

[5] Zhimin Chen, Ambuj Sinha, and Patrick Schaumont. Implementing virtual secure circuit using a custom-instruction approach. In *Proceedings of the 2010 international conference on Compilers, architectures and synthesis for embedded systems*, pages 57–66. ACM, 2010.

[6] Mohammad Tehranipoor and Cliff Wang. *Introduction to hardware security and trust*. Springer Science & Business Media, 2011.

[7] Philippe Hoogvorst, Guillaume Duc, and Jean-Luc Danger. Software implementation of dual-rail representation. *COSADE, February*, pages 24–25, 2011.

[8] Pablo Rauzy, Sylvain Guilley, and Zakaria Najm.

Formally proved security of assembly code against power analysis. *Journal of Cryptographic Engineering*, pages 1–16, 2015.

[9] Eli Biham. A fast new des implementation in software. In *International Workshop on Fast Software Encryption*, pages 260–272. Springer, 1997.

[10] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. Present: An ultra-lightweight block cipher. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 450–466. Springer, 2007.

[11] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. Nicv: normalized inter-class variance for detection of side-channel leakage. In *Electromagnetic Compatibility, Tokyo (EMC'14/Tokyo), 2014 International Symposium on*, pages 310–313. IEEE, 2014.

[12] Yang Han, Yongbin Zhou, and Jiye Liu. Securing lightweight block cipher against power analysis attacks. In *Future Wireless Networks and Information Systems*, pages 379–390. Springer, 2012.

[13] Victor Servant, Nicolas Debande, Houssem Maghrebi, and Julien Bringer. Study of a novel software constant weight implementation. In *International Conference on Smart Card Research and Advanced Applications*, pages 35–48. Springer, 2014.

[14] Wenling Wu and Lei Zhang. Lblock: a lightweight block cipher. In *International Conference on Applied Cryptography and Network Security*, pages 327–344. Springer, 2011.

[15] Cong Chen, Thomas Eisenbarth, Aria Shahverdi, and Xin Ye. Balanced encoding to mitigate power analysis: a case study. In *International Conference on Smart Card Research and Advanced Applications*, pages 49–63. Springer, 2014.

[16] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, et al. Prince–a low-latency block cipher for pervasive computing applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 208–225. Springer, 2012.

[17] B Ray, S Douglas, S Jason, TC Stefan, W Bryan, and W Louis. The simon and speck families of lightweight block ciphers. Technical report, Cryptology ePrint Archive, Report./404, 2013.

Morteza Safaei Pour received his B.S. and M.S. in electrical engineering from Sharif University of Technology, Tehran, Iran, in 2013 and 2016, respectively. His research interests include communication networks, computer science and information security.

Mahmoud Salmasizadeh received the B.S. and M.S. degrees in electrical engineering from Sharif University of Technology, Tehran, Iran, in 1972 and 1989, respectively. He also received the Ph.D. degree in information technology from Queensland University of Technology, Australia, in 1997. Currently, he is an associate professor in the Electronics Research Institute and adjunct associate professor in the Electrical Engineering Department, Sharif University of Technology. His research interests include design and cryptanalysis of cryptographic algorithms and protocols, e-commerce security, and information theoretic secrecy. He is a founding member of Iranian Society of Cryptology.

Appendix

A Experimental Setup

With the aim of evaluation of security, features and efficacy of a proposed method, it is required to be implemented in a cryptographic device, data are collected, attack is imposed and the results are compared with the unprotected mode. In this paper we used smart card which is a usual cryptographic device. This was a contact smart card with ATmega163 microprocessor, 16kB internal flash memory, 2k external EEPROM memory and clock frequency of 8MHz. In [8], authors measured the leakage characterization of all the eight bits of the ATmega163 microprocessor. The results showed that the leakage characterization for bit one is not the same as other seven bits. Therefore, this would aggravate the results of our proposed methods because we assume that all the processed bits are homogeneous with the leakage perspective. However, we decided to use this smart card with ATmega163 processor for two reasons. First, this smart card is widely used for side channel security evaluation in many of the previous papers and research. Second, because our proposed methods are not limited to some specific type of microprocessors, it could show us the results of the worst case that even with this property, still the proposed methods are effective.

In order to establish a communication properly with smart card, the intended assembly code and

No.	Operation	Changed Register	Proof	Step
1	$ldi\ ZH, 0x0d$			
2	$mov\ ZL, A1$	$ZL = [\bar{b_0}\, b_0\, \bar{b_0}\, b_0\, \bar{a_0}\, \bar{a_0}\, a_0\, a_0]$		
3	$lpm\ A1, Z$	$A1 = [\bar{b_0}\, 0\, 0\, b_0\, 0\, \bar{a_0}\, a_0\, 0]$	$HD(A1_{new}, A1_{old}) = 2$	$LUT1$
4	$mov\ ZL, A2$	$ZL = [\bar{a_1}\, \bar{a_1}\, a_1\, a_1\, \bar{b_1}\, b_1\, \bar{b_1}\, b_1]$	$ZL_{old} \in [B\|A],\ ZL_{new} \in [A\|B] \rightarrow HD(ZL_{new}, ZL_{old}) = 4$	
5	$lpm\ ZL, Z$	$ZL = [0\, \bar{a_1}\, a_1\, 0\, \bar{b_1}\, 0\, 0\, b_1]$	$HD(ZL_{new}, ZL_{old}) = 2$	$LUT2$
6	$eor\ ZL, A1$	$ZL = [\bar{b_0}\, \bar{a_1}\, a_1\, b_0\, \bar{b_1}\, \bar{a_0}\, a_0\, b_1]$	$HD(ZL_{new}, ZL_{old}) = 2$	**d**
7	$mov\ A1, ZL$	$A1 = [\bar{b_0}\, \bar{a_1}\, a_1\, b_0\, \bar{b_1}\, \bar{a_0}\, a_0\, b_1]$	$HD(A1_{new}, A1_{old}) = 2$	
8	$ldi\ ZH, 0x0b$			
9	$lpm\ A1, Z$	$A1 = LUT4(\mathbf{d})$	$HD(A1_{new}, A1_{old}) = HD(\mathbf{d}, LUT4(\mathbf{d})) = 2$	$LUT4$
10	$ldi\ ZH, 0x0c$			
11	$lpm\ ZL, Z$	$ZL = LUT5(\mathbf{d})$	$HD(A1_{new}, A1_{old}) = HD(\mathbf{d}, LUT5(\mathbf{d})) = 2$	$LUT5$
12	$eor\ A1, ZL$	$A1 = LUT4(\mathbf{d}) \oplus LUT5(\mathbf{d})$	$HD(A1_{new}, A1_{old}) = HW(LUT5(\mathbf{d})) = 6,\ A1_{new} \in [C\|C]$	
13	$mov\ ZL, A1$	$ZL = LUT4(\mathbf{d}) \oplus LUT5(\mathbf{d})$	$HD(ZL_{new}, ZL_{old}) = HW(LUT4(\mathbf{d})) = 6,\ ZL_{new} \in [C\|C]$	
14	$swap\ A2$	$A2 = [\bar{b_1}\, b_1\, \bar{b_1}\, b_1\, \bar{a_1}\, \bar{a_1}\, a_1\, a_1]$	$A2_{new} \in [B\|A]$	

Table A.1. Assembly code for implementing equivalent-AND method 2 of $A1 = [\bar{b_0}\, b_0\, \bar{b_0}\, b_0\, \bar{a_0}\, \bar{a_0}\, a_0\, a_0] \in [B|A]$ and $A2 = [\bar{a_1}\, \bar{a_1}\, a_1\, a_1\, \bar{b_1}\, b_1\, \bar{b_1}\, b_1] \in [A|B]$

the open source operation system "SOSSE" are compiled and then they are programmed in smart card. SOSSE controls the interactions and packet transmission/receive between smart card and computer based on ISO 7816 standard.

Oscilloscope could not measure device's consumed power or electrical current, therefore, proportional voltage should be measured instead. For this purpose, usually, a small electrical resistance is putted before ground or power supply pins, and its voltage is measured because it is proportional to the consumed power assuming a constant voltage source [2]. For that reason, we put a 10Ω resistance in the way of the card reader's electrical current to ground pin and measure and save its difference voltage during encryption process. We used Agilent U2702A oscilloscope which have two channels, with 1GS/sec sampling rate and bandwidth of 200MHz. Figure A.1 shows the used experimental setup.

Another point is that lookup tables related to proposed methods are required to be placed in a specific memory address (Section 3). We looked into the Hex file and found related addresses after compiling the code. We used the ".org assembly code to specify intended memory address and put the proper value in it, and used "LPM assembly code to read that memory address later. We generated the round keys based on key scheduling of SIMON, without any coding or mapping, but later on that they are required for some operation with coded them based on proposed methods.

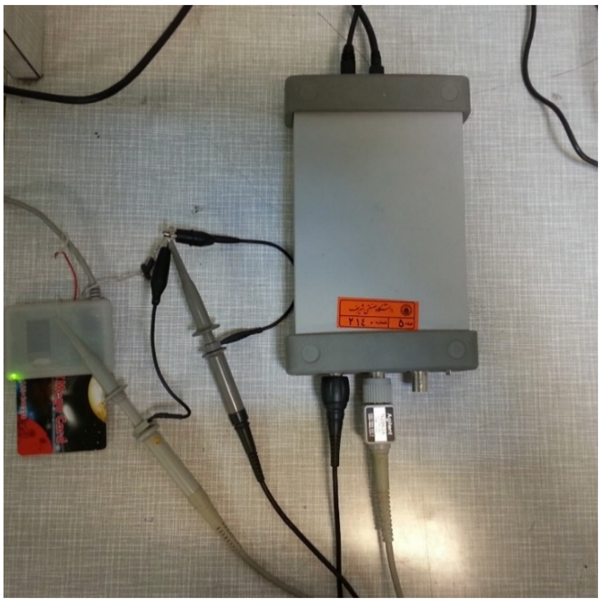


Figure A.1. Experimental setup: data acquisition during encryption

B Assembly Code

The assembly implementation of *equivalent-AND* method-2 is provided as a sample in Table A.1. This table explains all the steps in detail. It shows the changed register after the operation and the proof related to that register's hamming distance. ZL and ZH are two 8-bit registers ($r30$ and $r31$, respectively), together they form the Z register which is used for pointing to an address in memory and retrieving its content.