Differential Fault Attack on LEA

Dirmanto Jap[1] and Jakub Breier[2]

[1]School of Physical and Mathematical Sciences
Nanyang Technological University, Singapore
dirm0002@e.ntu.edu.sg
[2]Physical Analysis and Cryptographic Engineering
Temasek Laboratories at Nanyang Technological University, Singapore
jbreier@ntu.edu.sg

Abstract. LEA is a symmetric block cipher proposed in 2014. It uses ARX design and its main advantage is the possibility of a fast software implementation on common computing platforms.
In this paper we propose a Differential Fault Analysis attack on LEA. By injecting random bit faults in the last round and in the penultimate round, we were able to recover the secret key by using 258 faulty encryptions in average. If the position of faults is known, then only 62 faulty encryptions are needed in order to recover the key which surpasses the results achieved so far.

Keywords: LEA, Fault Attack, DFA

1 Introduction

Today's applications require efficient ciphers that can run on small devices with constrained computing power. Recent trends show an increasing number of services intended for Internet of Things [8, 5] requiring both high level of security and fast running speed on embedded devices. For such applications, lightweight cryptography is an ideal choice.

LEA [6] is a symmetric block cipher, using the ARX design (modular Addition, bitwise Rotation, and bitwise XOR). It offers fast software encryption, comparable to lightweight ciphers, and comes in the same key size variants as AES. There is an exhaustive security analysis report published by Bogdanov et al. [2], stating that the cipher is secure against known cryptanalysis attacks. So far, there was only one attempt to break the cipher using fault analysis method, which requires 300 chosen fault injections for recovering the 128-bit secret key.

In this paper we present a Differential Fault Analysis attack on LEA. We exploit properties of a non-linearity of modular addition operation used in a round function. To recover the key, our attack requires two different positions of fault injections - in the last round and in the penultimate round. By using a random bit-flip model, we were able to recover a 128-bit secret key by using $\approx$258 faulty ciphertexts in average. If the precise fault position is known, our attack requires only $\approx$62 faulty ciphertexts in average. Thus, our method overcomes the fault attack on LEA published so far.

This paper is organized as follows. First, we provide an overview of related work in Section 2. LEA cipher is described in details in Section 3. Section 4 provides methodology of our fault attack, following by Section 5 which summarizes our simulation results. Finally, Section 6 concludes this work and provides motivation for further work.

2 Related Work

Since the first publication proposing a fault analysis as a method to retrieve the secret information from the encryption process proposed by Boneh, DeMillo, and Lipton in 1997 [3], this technique became very popular and testing of implementations against information leakage from fault attack has become a standard procedure.

Differential Fault Analysis technique was first introduced by Biham and Shamir [1], using this powerful technique for revealing the DES secret key. Many other techniques using fault injection have been proposed so far, e.g. Collision Fault Analysis (CFA), Ineffective Fault Analysis (IFA), and Safe-Error Analysis (SEA) [4].

There is only one fault attack on LEA published so far, using the Differential Fault Analysis technique. Myungseo and Jongsun [7] used 300 chosen fault injections in order to recover 128-bit secret key. They inject faults into three out of four words in the intermediate state at round 24.

From the attack methodology point of view, the closest attack proposal to this paper is the attak proposed by Tupsamudre et al. [9], aiming at SPECK cipher. Since SPECK uses the ARX structure as well, authors aimed at the only non-linear operation, at the modular addition. They were able to recover the n-bit secret key by using $n/3$ bit faults on average.

3 LEA Cipher

In this section we will describe a symmetric block cipher LEA, introduced by Hong et al.[6]. According to security evaluation report [2], LEA is secure against state-of-the-art cryptographic attacks.

The block size of this cipher is 128 bits, key sizes can be 128, 192, or 256 bits. Number of rounds for each key size is 24, 28, and 32, respectively. It is a pure ARX cipher, consisting of modular Addition, bitwise Rotation, and bitwise XOR operations on 32-bit words. We will further describe the design of the 128-bit key size version of the cipher.

3.1 Encryption Procedure

First, the 128-bit intermediate value X_0 is set to the plaintext P. Then, a key schedule process creates r round keys. The 128-bit output $X_{i+1} = (X_{i+1}[0], \ldots, X_{i+1}[3])$ of i-th round is computed as:

$$X_{i+1}[0] \leftarrow ROL_9((X_i[0] \oplus RK_i[0]) \boxplus (X_i[1] \oplus RK_i[1]))$$
$$X_{i+1}[1] \leftarrow ROR_5((X_i[1] \oplus RK_i[2]) \boxplus (X_i[2] \oplus RK_i[3]))$$
$$X_{i+1}[2] \leftarrow ROR_3((X_i[2] \oplus RK_i[4]) \boxplus (X_i[3] \oplus RK_i[5]))$$
$$X_{i+1}[3] \leftarrow X_i[0]$$

The resulting ciphertext is then obtained after r rounds in the following way:

$$C[0] \leftarrow X_r[0], C[1] \leftarrow X_r[1], C[2] \leftarrow X_r[2], C[3] \leftarrow X_r[3].$$

The whole process is depicted in Figure 1.

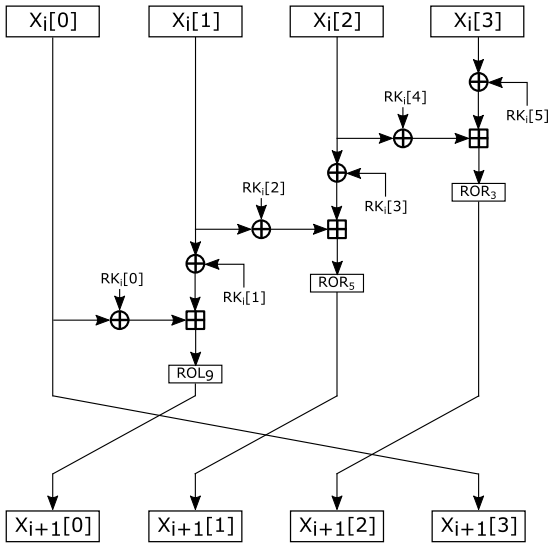


Fig. 1: i-th round function.

3.2 Key Schedule

Let $K = (K[0], K[1], K[2], K[3])$ be a 128-bit key. We set $T[i] = K[i]$ for $0 \leq i < 4$. Round keys $RK_i = (RK_i[0], \ldots, RK_i[5])$ for $0 \leq i < 24$ are then computed as follows:

$$T[0] \leftarrow ROL_1(T[0] \boxplus ROL_i(\delta[i \bmod 4]),$$
$$T[1] \leftarrow ROL_3(T[1] \boxplus ROL_{i+1}(\delta[i \bmod 4]),$$
$$T[2] \leftarrow ROL_6(T[2] \boxplus ROL_{i+2}(\delta[i \bmod 4]),$$
$$T[3] \leftarrow ROL_1 1(T[3] \boxplus ROL_{i+3}(\delta[i \bmod 4]),$$
$$RK_i \leftarrow (T[0], T[1], T[2], T[1], T[3], T[1]),$$

where $\delta[i]$ for $0 \leq i < 8$ are key generating constants, obtained from a hexadecimal expression of $\sqrt{766995}$, where 76, 69, and 95 are ASCII codes of 'L,' 'E,' and 'A.'

4 Attack Methodology

To perform a differential fault attack on LEA, we propose using two single bit flip faults, at $X_{22}[0]$ and at $X_{23}[2]$. The propagation of the faults can be observed in Figure 2. We then retrieve the key by exploiting the modular addition operation.

To describe the proposed method, we first show how it works on a normal modular addition. First, let us assume that the operation is done on 32-bit values, A and B. The modular addition could then be expressed as:

$$D = (A \boxplus B),$$
$$D_j = (A_j \oplus B_j \oplus c_j),$$

where $j \in \{0, ..., 31\}$, and c_j is a carry bit from the previous addition with $c_0 = 0$. The idea is, that the fault could be injected at A, and by observing $(A \oplus A^*)$ and the pair of correct-faulty outputs (D, D^*), the attacker could retrieve the value of A and B. Here, A^* denotes the faulty value of A, where $A^*_k \neq A_k$ for $k \in \{j, j+1, ..., j+n\}$ and $n \geq 0$. The value k denotes the position of fault(s). If $n = 0$, then it is a single bit fault, otherwise, it is a multiple consecutive bit fault.

From the output value D and D^*, the attacker could also observe $D \oplus D^*$. The attacker then checks how many N bit difference(s) are in $D \oplus D^*$. First, we consider the case with only 1 bit difference in $(A \oplus A^*)$, more specifically, at bit j. Starting from the location of fault j, the attacker calculates N. If the value of $N = 1$, it can be concluded that the carry bit at bit $j+1$ is not flipped and hence, from the left part of the Table 1, we can conclude that the value of $B_j = c_j$ (highlighted with red color). However, if $N_1 > 1$, it can be concluded that the carry bit at $j+1$ is flipped, and thus, $B_j \neq c_j$. Note that this attack requires that the carry bit c_j is known and thus, it relies on the assumption that the 0-th bit could be faulted in the process (since $c_0 = 0$ is the only known carry from the beginning). Once the values of B_j, c_j and D_j are known, the value A_j could be determined ($A_j = D_j \oplus B_j \oplus c_j$), as well as the value c_{j+1} (by observing the carry bit of $A_j + B_j + c_j$).

Next, we consider the case when there are multiple consecutive bit differences at $(A \oplus A^*)$, in bits $j, ..., j+n$. First, the attacker needs to determine if the carry bit at the bit $j+i$ ($i \in \{0, ..., n\}$) is flipped or not. The attacker observes $(D \oplus D^*)_{j+i}$.

$$(D \oplus D^*)_{j+i} = (A_{j+i} \oplus B_{j+i} \oplus c_{j+i}) \oplus (A^*_{j+1} \oplus B_{j+1} \oplus c^*_{j+1})$$
$$(D \oplus D^*)_{j+i} = 1 \oplus c_{j+i} \oplus c^*_{j+i}$$

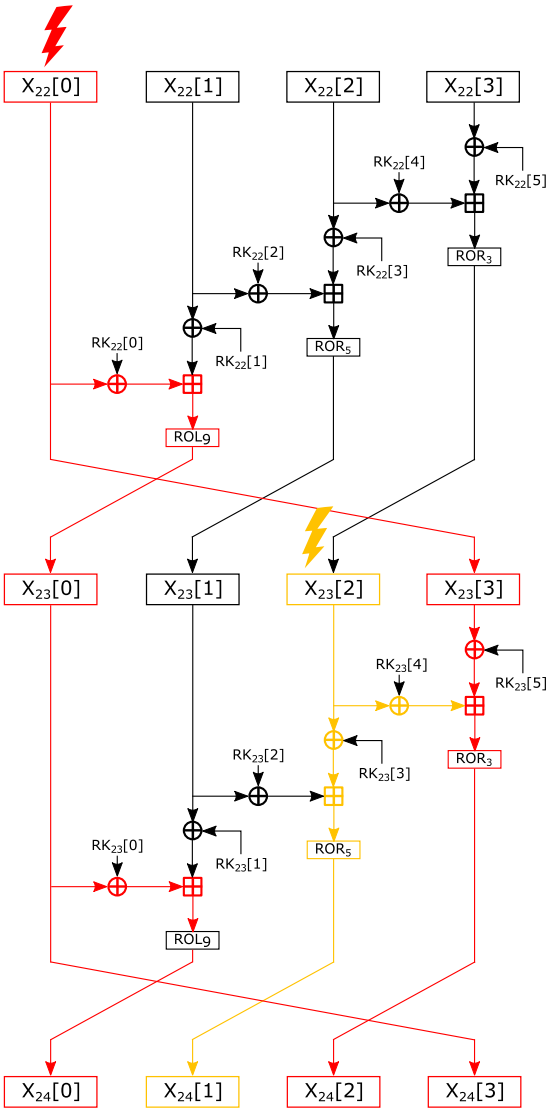


Fig. 2: Fault propagation in rounds 23 and 24.

So, if the value of $(D \oplus D^*)_{j+i} = 1$, it means that $c_{j+i} = c^*_{j+i}$, and similarly, if $((D \oplus D^*)_{j+i} \neq 1) \Rightarrow (c_{j+i} \neq c^*_{j+i})$. Then, the attacker determines the value of the carry bit in the next bit $(j + i + 1)$ using similar approach. The Table 1 summarizes the scenario when the carry values for the current bit and the consecutive bit are flipped or not. To highlight, if the value of the current bit is not flipped and if the value in the next carry bit is not flipped as well, then $B_{j+i} = c_{j+i}$ (left part, highlighted with red color). If the value of the current bit

is not flipped and if the value in the next carry bit is flipped, then $B_{j+i} \neq c_{j+i}$ (left part). On the contrary, if the value of the current carry bit is flipped, and also the value in the next carry bit is flipped, then $A_{j+i} = c_{j+i}$ (right part). If the value of the current bit is flipped and if the value in the next carry bit is not flipped then $A_{j+i} \neq c_{j+i}$ (right part, highlighted with yellow color). Once the value of either A_{j+i}, or B_{j+i} is known, the other value and the next carry bit c_{j+i+1} could be determined.

Table 1: Changes of values in modular addition after a bit flip.

First faulty bit

A_j	A_j^*	B_j	c_j	c_{j+1}	c_{j+1}^*
0	1	0	0	0	0
0	1	0	1	0	1
0	1	1	0	1	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	1	1	1

Consecutive faulty bit

A_j	A_j^*	B_j	c_j	c_j^*	c_{j+1}	c_{j+1}^*
0	1	0	0	1	0	1
0	1	0	1	0	0	0
0	1	1	0	1	0	1
0	1	1	1	0	1	1
1	0	0	0	1	0	0
1	0	0	1	0	1	0
1	0	1	0	1	1	1
1	0	1	1	0	1	0

5 Experiments

We tested the proposed method by conducting an experiment using simulated faults. Here, the attacker uses one fixed plaintext, measured multiple times, and injects fault at different bit locations. Then, he collects all the faulty ciphertexts. As mentioned earlier, the attacker has to inject faults at two different positions of different rounds. The main target is LEA-128, due to its key structure. Each of the round keys can be segmented into 6 parts, each part consists of 32 bits. In LEA-128, the second, the fourth and the last part ($RK_i[1], RK_i[3]$, and $RK_i[5]$) are identical, based on the key scheduling. In previous section, it is shown that the attack exploits the modular addition $D = (A \boxplus B)$.

5.1 Phase 1

First, the attacker injects single bit flip fault at $X_{22}[0]$, and the fault will propagate to $X_{24}[0]$, $X_{24}[2]$ and $X_{24}[3]$. The fault propagation will not be affected by the XOR operation with the round keys.

1. The attacker first exploits the difference $(X_{24}[3] \oplus X_{24}^*[3]) = (X_{23}[0] \oplus X_{23}^*[0])$. Then, he can pinpoint the location of initial fault injected by taking the last bit difference in $ROR_9(X_{23}[0]) \oplus ROR_9(X_{23}^*[0])$, since the attacker knows that the fault is a single bit flip and the locations of faults have been changed due to a rotation.

2. Then, the attacker can construct $(X_{22}[0] \oplus X^*_{22}[0])$. Here, the difference $(X_{24}[3] \oplus X^*_{24}[3])$ can be considered as caused by multiple consecutive bit flips injected at $X_{23}[0]$.
   Input: $(X_{23}[0] \oplus X^*_{23}[0])$, $X_{24}[0]$, $X^*_{24}[0]$,
   Output: estimated $(X_{23}[0] \oplus RK_{23}[0])$, $(X_{23}[1] \oplus RK_{23}[1])$.
   However, since $X_{23}[0] = X_{24}[3]$, the round key $RK_{23}[0]$ can be calculated, and using the key schedule algorithm, $RK_{22}[0]$ can be determined as well from $RK_{23}[0]$.
3. Using the knowledge from the previous step, attacker can continue with revealing partial information.
   Input: $(X_{22}[0] \oplus X^*_{22}[0])$, $X_{23}[0]$, $X^*_{23}[0]$,
   Output: estimated $(X_{22}[0] \oplus RK_{22}[0])$, $(X_{22}[1] \oplus RK_{22}[1])$.
   Since the attacker already knows $RK_{22}[0]$, he can reveal $X_{22}[0]$ based on the estimated output, which in turn reveals $X_{23}[3]$.
4. Here, $(X_{22}[0] \oplus X^*_{22}[0])$ is the same as $(X_{23}[3] \oplus X^*_{23}[3])$.
   Input: $(X_{23}[3] \oplus X^*_{23}[3])$, $X_{24}[2]$, $X^*_{24}[2]$,
   Output: estimated $(X_{23}[3] \oplus RK_{23}[5])$, $(X_{23}[2] \oplus RK_{23}[4])$.
   However, as the $X_{23}[3]$ is known by previous step, the attacker obtains $RK_{23}[5]$, which, by key scheduling, is the same as obtaining $RK_{23}[1]$ and $RK_{23}[3]$.

By the end of phase 1, the attacker has obtained the values of $X_{22}[0](= X_{23}[3])$, $RK_{22}[0]$, $X_{23}[0]$, $RK_{23}[0]$, $X_{23}[1] \oplus RK_{23}[1]$, $RK_{23}[1]$, $X_{23}[2] \oplus RK_{23}[4]$, $RK_{23}[3]$ and $RK_{23}[5]$.

5.2 Phase 2

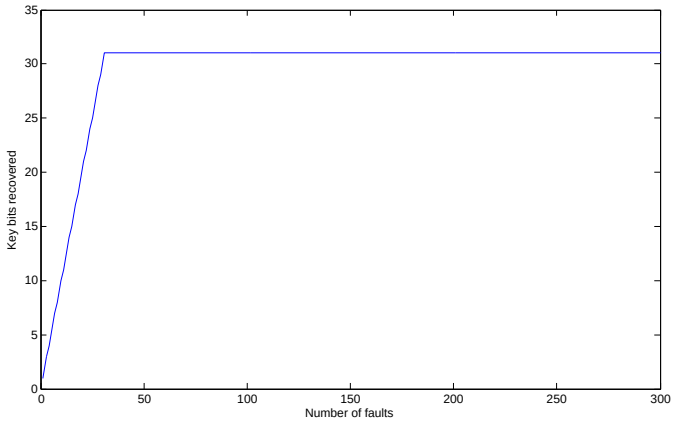
With the partial knowledge of the key and the intermediate states from phase 1, the attacker injects another single bit flip fault at $X_{23}[2]$.

1. By observing $(X_{24}[2] \oplus X^*_{24}[2])$, he can determine $(X_{23}[2] \oplus X^*_{23}[2])$, since it is a single bit flip fault.
   Input: $(X_{23}[2] \oplus X^*_{23}[2])$, $X_{24}[1]$, $X^*_{24}[1]$,
   Output: estimated $(X_{23}[1] \oplus RK_{23}[2])$, $(X_{23}[2] \oplus RK_{23}[3])$.
   From the previous phase, $X_{23}[1]$ could be determined since $X_{23}[1] \oplus RK_{23}[1]$ and $RK_{23}[1]$ are known. Then, $RK_{23}[2]$ could be determined from $(X_{23}[1] \oplus RK_{23}[2])$ and $X_{23}[1]$.
2. Similarly, since $RK_{23}[3]$ is known from the previous phase, $X_{23}[2]$ can be calculated with $(X_{23}[2] \oplus RK_{23}[3])$, obtained in previous step.
3. Since $(X_{23}[3] \oplus RK_{23}[5])$ has been determined in previous phase, together with $X_{23}[2]$ from the previous step and $X_{24}[2]$, which is the output, the attacker can determine $RK_{23}[4]$.

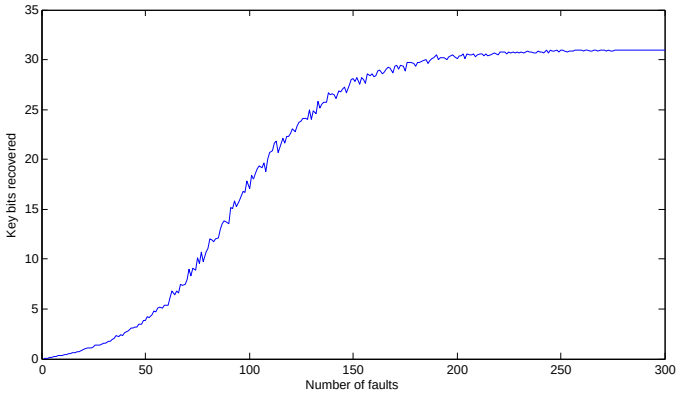
Thus, the remaining of last round key, $RK_{23}[2]$ and $RK_{23}[4]$, can be retrieved in this phase.

5.3 Observations

One observation regarding the attack is that the last bit (MSB) could not be obtained because the next carry bit cannot be determined. This condition holds for each part of the key and hence, in total, 4 bits of the key could not be determined. Thus, these remaining key bits have to be brute-forced. An-



(a) Known position of faulty bits.



(b) Unknown position of faulty bits.

Fig. 3: Number of key bits retrieved.

other problem could also occur due to the rotation of the bits. Due to rotation ROL_9, the resulting bit difference might not be consecutive, for example, $ROL_9(0\ldots01\ldots1) = (110\ldots01\ldots1)$. To mitigate this problem, it could be

considered as separate multiple consecutive faults. From the example, the fault can be considered as $(110\ldots0)$ and $(0\ldots01\ldots1)$. Since the attack only considers bits of D and D^* which coincide with the faulty bits in A^*, the previous method still works.

In Figure 3, we show the number of faults required to retrieve the value of the input of the modular addition. In the case where each bit could be flipped precisely, around 30 faults are required. However, if the bit could be flipped only randomly, more faults are required.

Based on the experiment, if the attacker cannot determine the location of a single bit flip fault, and hence, could only inject faults randomly at single bit, it requires ≈ 130 and ≈ 128 for the first and the second fault respectively in order to determine the last round key, minus the 4 bits mentioned earlier.

However, if the attacker could determine the precise location of a single bit flip, it requires only ≈ 31 and ≈ 31 faults for the first and the second fault respectively. This is because each fault reveals one bit of the key and in each attack, 2 parts of the last round key could be determined. For the previous attack model, more faults are required in order to obtain the value of each bit, similar to the coupon collector problem ($\Theta(n\log n)$). These results were verified by simulating 500 independent experiments on different scenarios.

6 Conclusions

In this paper we have proposed a differential fault attack on LEA. We used a random bit flip model, aiming at the last two rounds of the cipher. Using this model, we were able to retrieve the secret key with around 258 faulty encryptions in average. However, if the attacker can determine the position of faults, number of faults can be reduced to only 62 faulty encryptions in average. This result overcomes the only known fault attack on LEA published so far which needs 300 chosen faulty encryptions in order to reveal the key.

In the future, we would like to aquire practical results for our fault model, using the software implementation of LEA on a microcontroller, with the laser fault injection technique.

References

1. E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In J. Kaliski, BurtonS., editor, *Advances in Cryptology CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer Berlin Heidelberg, 1997.
2. A. Bogdanov, K. Varici, N. Mouha, V. Velichkov, E. Tischhauser, M. Wang, D. Toz, Q. Wang, and V. Rijmen. Security Evaluation of the Block Cipher LEA. Technical report, July 2011.
3. D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults. In *Proceedings of the 16th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'97, pages 37–51, Berlin, Heidelberg, 1997. Springer-Verlag.

4. C. Clavier. Attacking Block Ciphers. In M. Joye and M. Tunstall, editors, *Fault Analysis in Cryptography*, Information Security and Cryptography, pages 19–35. Springer Berlin Heidelberg, 2012.
5. N. K. Giang, J. Im, D. Kim, M. Jung, and K. Wolfgang. Integrating the EPCIS and Building Automation System into the Internet of Things: a Lightweight and Interoperable Approach. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 6(1):56–73, 2015.
6. D. Hong, J.-K. Lee, D.-C. Kim, D. Kwon, K. Ryu, and D.-G. Lee. Lea: A 128-bit block cipher for fast encryption on common processors. In Y. Kim, H. Lee, and A. Perrig, editors, *Information Security Applications*, volume 8267 of *Lecture Notes in Computer Science*, pages 3–27. Springer International Publishing, 2014.
7. P. Myungseo and K. Jongsung. Differential Fault Analysis of the Block Cipher LEA. *Journal of the Korea Institute of Information Security and Cryptology*, 24(6):1117–1127, 2014.
8. T. Robles, R. Alcarria, D. Martín, M. Navarro, R. Calero, S. Iglesias, and M. López. An IoT based reference architecture for smart water management processes . *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 6(1):4–23, 2015.
9. H. Tupsamudre, S. Bisht, and D. Mukhopadhyay. Differential fault analysis on the families of simon and speck ciphers. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2014 Workshop on*, pages 40–48, Sept 2014.