Differential Fault Intensity Analysis on PRESENT and LED Block Ciphers

Nahid Farhady Ghalaty, Bilgiday Yuce, Patrick Schaumont

Bradley Department of Electrical and Computer Engineering,
Virginia Polytechnic Institute and State University, Blacksburg, VA, USA
{farhady,bilgiday,schaum}@vt.edu

Abstract. Differential Fault Intensity Analysis (DFIA) is a recently introduced fault analysis technique. This technique is based on the observation that faults are biased and thus are non-uniformly distributed over the cipher state variables. The adversary uses the fault bias as a source of leakage by controlling the intensity of fault injection. DFIA exploits statistical analysis to correlate the secret key to the biased fault behavior. In this work, we show a DFIA attack on two lightweight block ciphers: PRESENT and LED. For each algorithm, our research analyzes the efficiency of DFIA on a round-serial implementation and on a nibble-serial implementation.We show that all algorithms and all implementation variants can be broken with 10 to 36 faults, depending on the case. We also analyze the factors that affect the convergence of DFIA. We show that there is a trade-off between the number of required plaintexts, and the resolution of the fault-injection equipment. Thus, an adversary with lower-quality fault-injection equipment may still be as effective as an adversary with high-quality fault-injection equipment, simply by using additional encryptions. This confirms that DFIA is effective against a range of algorithms using a range of fault injection techniques.

Keywords: Differential Attack, Fault Intensity, Light-weight Block Cipher, PRESENT, LED

1 Introduction

Nowadays, lightweight cryptographic primitives are recommended to be used to secure various resource-constrained systems such as RFID tags and sensor networks [1] [2]. The security of a cryptographic primitive relies on both its algorithmic features and on its physical implementation.

Physical attacks are divided into two groups. Side Channel Attacks retrieve the secret key by using statistical tests on the information leaked from the cryptographic device during its execution [3]. Fault attacks, first, intentionally disturb a cryptographic device by means of fault injection to induce errors in the output of the device. Then, they exploit the erroneous outputs to mathematically reverse-engineer the secret key [4].

Differential Fault Intensity Analysis (DFIA) is a recently introduced fault analysis technique [5]. In DFIA, the attacker injects faults by means of intentional variation of the fault intensity. Using this fault injection technique, he induces biased faults in the intermediate state of the cryptographic algorithm. Under the biased fault model, a gradual change in the fault intensity will cause a small change in the faulty state variable. Using the faulty ciphertext, the attacker computes the key-dependent secret state variable under each key hypothesis. Finally, he performs statistical tests on each of the computed state variables and selects the key guess that is most likely under the biased fault model. Due to the non-linear transformations of the cipher, the correct key hypothesis shows only small changes on the variable,while the wrong key guesses show a random behavior. This attack combines the principles of Differential Power Analysis and fault injection.

In this paper, we demonstrate a DFIA attack on two lightweight cryptographic algorithms: PRESENT [6] and LED [7]. In contrast to AES, PRESENT and LED are nibble-oriented (4-bit). This makes the observation and exploitation of biased faults more difficult. We therefore investigate the feasibility of DFIA on both nibble-serial and round-serial implementations of PRESENT and LED. We evaluate the practicality of the biased fault model and the attack strategies on both nibble-serial and round-serial implementations of the algorithms.

Our results show that a single plaintext and 10 fault injections are sufficient to extract the key of a nibble-serial PRESENT-80 design. We also show that 12 fault injections are sufficient to extract the key of a round-serial PRESENT-80 design. Besides a DFIA on PRESENT-80, the paper also provides the attack results for PRESENT-128, LED-80, and LED-128. We confirm that all these designs can be broken.

We also demonstrate that DFIA [5] can be easily extended over multiple plaintexts, and that this increases the efficiency of the attack in narrowing down the key search space. We show that using multiple plaintexts can compensate for the low-resolution fault injection equipments. Therefore, DFIA can still retrieve the correct key efficiently, even if the attacker is not in possession of a high-quality fault injection tool.

The paper is organized as follows. Section 2 describes the DFIA and its fault model requirements. In this section, we also explain the PRESENT and LED algorithms and the nibble-serial and round-serial implementations of these algorithms. Section 3 explains the DFIA attack procedure on the PRESENT and LED algorithms. Section 4 shows the required number of fault injections for a DFIA attack on the PRESENT and LED. In this section, we also show the efficiency of the extended version of the DFIA. Section 5 covers the previous work that relies on fault bias. Section 6 concludes the paper.

2 Background and Notation

This section explains the principles of the DFIA method. We will first explain the concept of biased fault and an easy way to control it.

Table 1. Symbols of DFIA Attack Procedure

P	Plaintext
Q	Total number of injected faults with different intensities
q	A specific fault intensity
C'_q	Faulty ciphertext under fault intensity q
S	Correct state
k	Key hypothesis
$S'_{k,q,P}$	Faulty state under hypothesis $K = k$, fault q, $S'_{k,q,P} = f(C'_q, k)$ and input P

2.1 Fault Model

The fault model is a combination of three factors. These factors are fault location, fault timing and fault type. Fault location and fault timing define the spatial and temporal location of the fault in a hardware circuit, respectively. The fault type describes the behavior of the injected fault, and can be stuck-at, set-reset, random bit-flip, or biased fault respectively. Throughout this paper, we refer to following terms and definitions:

- Fault Intensity: Fault intensity is the strength by which a circuit is pushed outside of its nominal operating conditions with the intent of inducing a fault. For example, when faults are introduced using clock glitches, then the fault intensity corresponds to the shortened clock cycle that is obtained as a result of the glitches.
- Fault Sensitivity: The fault sensitivity is the fault intensity at which a hardware circuit reflects faulty behavior [8]. For example, when faults are injected by means of clock glitches, then fault sensitivity generally corresponds to the critical path of the circuit.
- Biased Fault: A biased fault is the incremental fault behavior obtained as a result of gradual increase in fault intensity. For DFIA, we are especially interested in using minimal fault bias (e.g. changes of one or two bits in a state variable), although other authors have shown that any fault bias is a source of leakage [9].

One of the cheapest and most convenient methods of injecting biased faults into a hardware device is clock glitching. In this method, the attacker creates biased faults via injecting glitches into the clock signal of the device. To make a circuit fail its timing constraints,the attacker gradually increases the fault intensity by decreasing the clock period via glitch injection. As a result, he can obtain biased faults because of the existing non-uniformity in the path delays of the circuit.

2.2 Differential Fault Intensity Analysis using Multiple Plaintexts

This section summarizes Differential Fault Intensity Analysis. Algorithm 1 describes the attack procedure, and Table 1 lists the symbols used in this paper.

Algorithm 1: DFIA Attack Procedure using Multiple Plaintext

```
Assume Cryptographic Algorithm, Fault Injection Tool;
Result Correct Key Guess ;
foreach Plaintext $P$ do
    foreach Faultintensity $q$, $1 \leq q \leq Q$ do
        Obtain faulty ciphertext $C'_q$;
        foreach Key Hypothesis $k$ do
            Compute faulty state hypothesis $S'_{k,q,P} = f(C'_q, k)$;
//Post-processing phase ;
foreach Key Hypothesis $k$ do
    Calculate $\rho_k = \sum_P \sum_{n=1}^{Q} \sum_{m=1}^{n-1} HD(S^{'}_{k,n,P}, S^{'}_{k,m,P})$;
$K = \min \rho_k$;
```

DFIA starts by applying a fault intensity q into an intermediate value S. The attacker next observes the faulty ciphertext C'_q, and derives the faulty intermediate value $S'_{k,q,P} = f(C'_q, k)$ under a key hypothesis k. The attacker repeats these two steps for Q different fault intensities by gradually increasing the fault intensity each time. In the post-processing step, for each key hypothesis, he computes the cumulative Hamming Distance among all faulty intermediate values. Finally, the attacker selects the key hypothesis that corresponds to the minimum cumulative Hamming Distance. The reason of looking for minimum is that for the correct key hypothesis, the cumulative Hamming Distance is correlated with the fault intensity, and thus, it is minimal. A wrong key hypothesis infers a larger, random cumulative Hamming Distance due to the non-linear diffusion and confusion properties of the attacked cipher. Hence, the correct key results in the minimum cumulative Hamming Distance as long as the applied fault intensities induce biased faults. Ghalaty et al. [5] show this behavior on AES for different biased fault injection scenarios. They draw two conclusions. First, DFIA converges for any given set of biased faults. Second, DFIA converges faster for strongly-biased faults (e.g. 1-bit faults) than it does for weakly-biased faults (e.g. 4-bit faults).

The original DFIA is applied using a single plaintext value [5]. However, DFIA can be easily extended to multiple plaintexts, by repeating the above steps for each plaintext, and by accumulating the resulting Hamming Distance values for each key hypothesis. Again, the global minimum will be obtained only under the correct key hypothesis. In this paper, we make use of this feature, and we show that it can be used to improve the efficiency of DFIA when few biased faults are available, or when the fault injection equipment has limited precision.

2.3 PRESENT Block Cipher

We make a brief overview of PRESENT and our implementations of it. PRESENT is a lightweight block cipher that was recently standardized by IEEE [6]. It uses an SP-network structure, and loosely follows the structure of AES, with the fol-

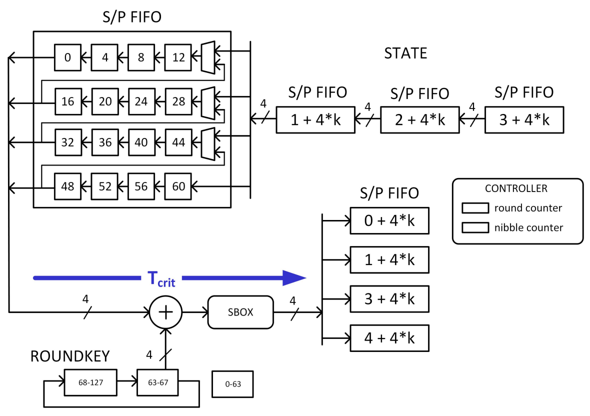


Fig. 1. Nibble-serial Implementation of PRESENT

lowing important differences. It has 31 rounds, and uses a block-length of 64 bits. It uses a selectable key size of 80 bit or 128 bit, and both versions are distinguished through their name (PRESENT-80 or PRESENT-128). Each round consists of three steps, including a roundkey addition layer, a nonlinear substitution layer with sixteen 4-bit Sbox, and a permutation layer. After the last round, an additional post-whitening step is included by adding a final roundkey.

The 64-bit roundkey is extracted from the upper part of the key register, and each round the key is updated with a key-size dependent key scheduling algorithm. The key schedule for PRESENT-80 is shown in Equations (1a) through (1c). The key schedule for PRESENT-128 is slightly more complex, and can be consulted in [6].

$$K_{79}K_{78}....K_0 = K_{18}K_{17}....K_{19} \tag{1a}$$
$$K_{79}K_{78}K_{77}K_{76} = Sbox[K_{79}K_{78}K_{77}K_{76}] \tag{1b}$$
$$K_{19}K_{18}K_{17}K_{16}K_{15} = K_{19}K_{18}K_{17}K_{16}K_{15} \oplus round\_counter \tag{1c}$$

In this work, we studied both a round-serial and a nibble-serial implementation. The reason for this is to show the feasibility of DFIA on different implementations of the same cipher. The round-serial implementation computes an entire round of a complete block in a single clock cycle. This implementation is straightforward and follows the design of the original PRESENT paper [6]. We also developed a nibble-serial design, as shown in Fig. 1. In this case, one round for a single nibble (4 bits) from a block is computed in a single clock cycle, and this requires sequentialization of the round operations. This is easy to achieve for the roundkey addition and the Sbox substitution. For the permutation layer, we make use of the property that PRESENT's permutation is a 4-bit by 16-bit transpose operation: 4 bits of the permutation output are taken from a column of

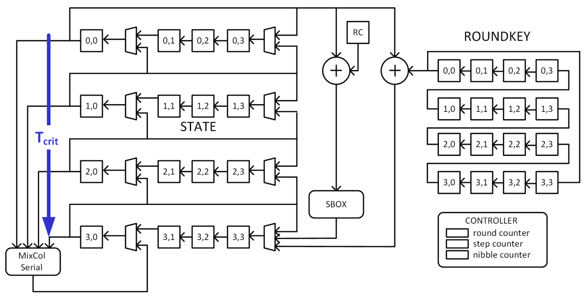


Fig. 2. Nibble-serial Implementation of LED

4-bits of an input block, when the block is arranged as a 4-bit by 16-bit matrix. In Fig. 1, we implement the permutation using serial/parallel FIFO modules, which consist of four 4-bit FIFO's that either operate as a single 16-bit FIFO (serial mode) or else as four parallel 4-bit FIFO's (parallel mode). A complete block is stored in four serial/parallel FIFOs. Using two such structures, which store either the odd or even round states, a compact nibble-serial version of PRESENT is obtained.

Of particular note for our fault analysis is the critical path in these structures. The critical path runs through the Sbox and roundkey addition operations. For the round-serial design, all Sbox operations will be in the critical path in a given clock cycle. For the nibble-serial design, on the other hand, only a single Sbox operation will be in the critical path in a given clock cycle.

2.4 LED Block Cipher

The Light Encryption Device (LED) is a compact block cipher that was developed after PRESENT, and that integrates further insight into the lightweight cipher design process [7]. This block cipher, too, is an SPN structure, with a 64-bit block size. It supports two different key sizes, 64-bit or 128-bit, and the notation LED-64 and LED-128 is used to distinguish these cases. LED-64 has 8 steps of 4 rounds each, for a total of 32 rounds. In between steps, roundkeys are added. Each of the rounds includes operations similar to AES (AddConstants(AC), SubCells(Sbox), ShiftRows(SR), MixColumnSerial(MC)), but each of these steps is specifically optimized towards lightweight encryption. LED organizes the state as a four by four matrix of nibbles, and the round operations operate on these nibbles.

The LED cipher does not use a key scheduling algorithm. Rather, it reuses the same key for every step. In the case of 128-bit key, the key bits are divided into two groups and each round uses one of them alternatively. LED includes a post-whitening step with a final addroundkey.

As with PRESENT, we developed a round-serial and a nibble serial version of LED for DFIA analysis. Fig. 2 shows the architecture of the nibble-serial design. It follows the design guidelines of the original LED paper [7]. The State is organized in a FIFO-like structure of 16 nibbles. The structure can rotate the first column to compute MixColumnSerial, and it can rotate rows to compute ShiftRows. SubCells and AddRoundKey rotate the entire matrix through an Sbox and round-key addition respectively. The critical path runs through the MixColumnSerial. This is true for either the nibble-serial as well as the round-serial design. Fault injection using glitches will directly affect the variables computed in the critical path.

2.5 Implementations of the Block Ciphers

We wrote Verilog codes for our block cipher designs, namely, round-serial LED (LED-rs), nibble-serial LED (LED-ns), round-serial PRESENT (PRE-rs), and nibble-serial PRESENT (PRE-ns). We choose the key size as 128-bit in our implementations. We also generated gate-level netlist files for an Altera Cyclone IV FPGA (60nm Technology). We use these netlists for gate-level simulations, which are carried out using Modelsim-Altera 10.1d [10] software, to verify our claims throughout the paper.

3 DFIA Attack on PRESENT and LED

In this section, we explain the DFIA attack on nibble-serial and round-serial implementations of PRESENT and LED block ciphers. To get the full key, the attacker must perform DFIA for the last two rounds of PRESENT-80 (i.e. round 30 and 31) and the last three rounds of PRESENT-128 [11]. The LED cipher has a very simple key scheduling method, and thus, we can retrieve the key by attacking the last round of LED-64. For LED-128, we have to attack the last two rounds to retrieve the key.

DFIA has two phases: Injecting biased faults into the intermediate state of the block cipher and post-processing the faulty ciphertexts to retrieve the key. The biased fault injection is nibble-wise (i.e., 4-bit) for nibble-serial implementations, while it is state-wise (i.e., 64-bit) for round-serial implementations. Regardless of DFIA on round-serial or nibble-serial designs, the post-processing is always applied on a single key nibble at a time.

3.1 Biased Fault Injection in PRESENT and LED

The proposed DFIA attacks build upon injecting biased faults in the inputs of Sbox blocks. One can use a clock glitch injection method such as in Figure 3(a) for this purpose. This method generates an input clock signal for the circuit as a combination of two clock signals, namely, glitch clock (clk_g) and nominal clock (clk_o). As it is seen in Figure 3(b), we inject glitches in the clk_o via an enable signal (g_en). To inject a biased fault in the input of an Sbox, we set the g_en signal just before the clock cycle, in which the Sbox is employed. Such a glitch injection makes some timing paths fail and causes a biased fault in the input of

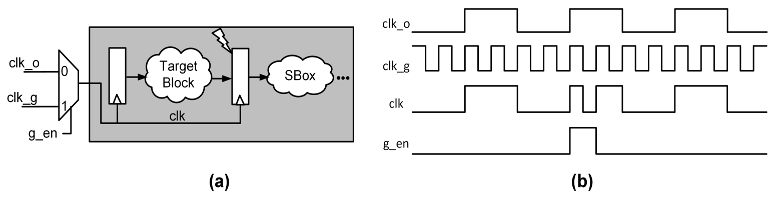


Fig. 3. (a)Block Diagram of Experimental Setup (b)Timing Diagram of Experimental Setup

X: Faulty Value
-: Correct Value

Glitch width (ns)	2.0	1.9	1.8	1.6	1.5	0.6
Bit 3	-	-	-	-	X	X
Bit 2	-	-	X	X	X	X
Bit 1	-	X	X	X	X	X
Bit 0	-	-	-	X	-	X

(a) Nibble-serial PRESENT

X: Faulty Value
-: Correct Value

Glitch width (ns)	1.6	1.4	1.3	1.2	1.1	0.8	0.6
Bit 3	-	-	-	-	-	X	X
Bit 2	-	-	-	-	X	X	X
Bit 1	-	-	-	X	X	X	X
Bit 0	-	X	-	X	X	X	-

(b) Round-serial PRESENT

X: Faulty Value
-: Correct Value

Glitch width (ns)	2.6	1.9	1.5	0.6
Bit 3	-	-	-	X
Bit 2	-	-	-	-
Bit 1	-	X	X	X
Bit 0	-	-	X	X

(c) Nibble-serial LED

X: Faulty Value
-: Correct Value

Glitch width (ns)	3.8	3.7	3.5	3.4	3.2	3.1	2.8	2.7	2.5	2.3	0.6
Bit 3	-	X	X	X	X	X	-	-	-	-	-
Bit 2	-	-	-	-	-	-	-	-	X	-	-
Bit 1	-	-	-	-	X	X	X	X	-	-	X
Bit 0	-	X	-	X	X	-	-	X	X	X	X

(d) Round-serial LED

Fig. 4. Biased Fault in the PRESENT and LED Implementations

the Sbox. We control the fault intensity by increasing/decreasing the frequency of the *clk_g* signal.

The target block of DFIA is different for each implementation. For LED-ns, the target block is MixColumnSerial (MC) logic. We create biased faults in the outputs of the MC logic by violating its timing paths. Then, the biased faults are transferred to the inputs of Sbox blocks via linear AddConstants (AC) layer. The target block of PRE-ns is the roundkey addition and substitution blocks. For LED-rs and PRE-rs the target blocks are the whole round logic of the corresponding algorithms.

3.2 Biased Faults in PRESENT and LED Exist

In this section, we present a set of experimental results to verify that fault bias is a feasible fault source. We demonstrated biased faults through gate-level (post-

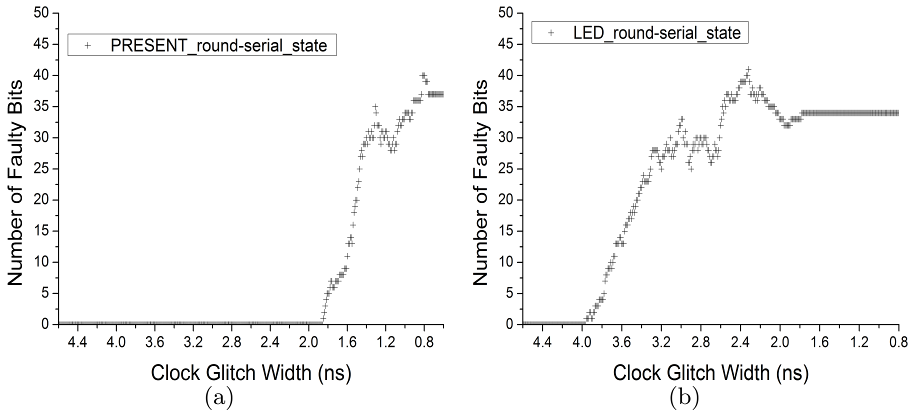


Fig. 5. Biased Fault Injection on State of (a) PRESENT and (b) LED

place-and-route) simulation of the four block cipher implementations. In Fig. 4, we present the results for Sbox of the four implementations for a single plaintext.

Fig. 4 shows the relationship between the clock glitch width and obtained faults in the Sbox inputs at the last round of the corresponding implementation. For each subgraph of Fig. 4, the horizontal axis is the clock glitch width and the vertical axis is the bit position. We mark a faulty bit position with the symbol (X) and mark a fault-free position with the symbol (-). In each subgraph of Fig. 4, we observe a minimal Hamming Distance between two neighbor columns. This behavior verifies the existence of fault bias in our implementations.

In Fig. 5(a) and Fig. 5(b), we show the number of faulty bits that are induced in the 64-bit state with respect to the clock glitch width for PRE-rs and LED-rs, respectively. These two graphs show that the fault bias exists for the 64-bit state as well. The PRE-rs will fail at higher fault intensity (i.e. at a narrower glitch width) than LED-rs. The reason is that the critical path of PRE-rs is shorter compared to the LED-rs. Thus, the attacker needs higher-capability fault injection tool to inject fault into PRE-rs.

3.3 Post-Processing of DFIA on PRESENT

In this section, we describe the procedure to retrieve the key for PRE-ns and PRE-rs implementations. To obtain the 80-bit key of PRESENT-80, we first retrieve the round key of round 31 to get the 64 most significant bits of the key. Then, to retrieve the remaining key bits, we retrieve the round key of round 30. Similarly, for PRESENT-128, the attacker must retrieve the round keys of rounds 31, 30, and 29.

We can retrieve each nibble of a round key separately. Therefore, the key retrieval procedure for nibble-serial and round-serial implementations is the same. Following is the procedure to retrieve the 80-bit key for PRESENT-80. We as-

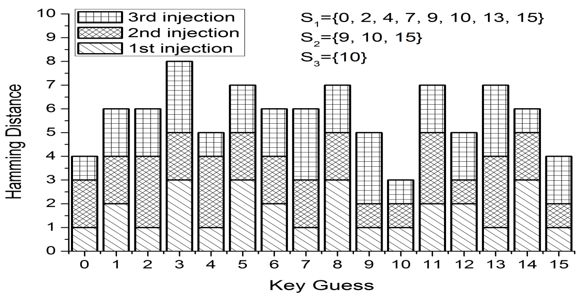


Fig. 6. DFIA Steps to Retrieve 4-bit Key of PRESENT

sume that the attacker has already collected the required amount of faulty ciphertexts to retrieve the key (using the method in Section 3.1).

The DFIA attack on PRESENT-80 follows Algorithm 1 as explained earlier. The faulty state variable is computed using Equation 2. By repeating this process for all nibbles of round 31, the attacker can retrieve the correct value for $K_{79}K_{78}....K_{16}$. In order to retrieve $K_{16}K_{15}....K_0$, the attacker has to process 4 least significant nibbles of round 30 as well.

$$S'_{k,C'} = PlayerInv(SboxInv(C' \oplus K)) \tag{2}$$

Fig. 6 shows an example DFIA attack to guess a nibble of a key. In this figure, the attacker injects four fault intensities: no injection, 1-bit fault injection, 2-bit fault injection and 3-bit fault injection. In this example, we retrieve one nibble of the round key with three fault injections. The bottom section of the bar chart shows the Hamming Distance between the first two intensities. The candidates for the correct key guess are the key guesses that show the minimum Hamming Distance, which is the set $G_1 = \{0, 2, 4, 7, 9, 10, 13, 15\}$ after the 1-bit fault injection. The middle section of the bar chart shows the Hamming Distance between 1-bit fault injection and 2-bit fault injection. As it is seen, the set of key candidates for correct key guess reduces to the set $G_2 = \{9, 10, 15\}$ after the 2-bit fault injection. The top section of the bar chart shows the Hamming Distance between 2-bit fault injection and 3-bit fault injection. The last fault injection gives us the unique key guess, which is $G_3 = \{10\}$.

3.4 Post-processing of DFIA on LED

In this section, we describe the procedure to retrieve the key for LED algorithm for nibble-serial and round-serial implementations. As the LED uses a very simple key scheduling method, the key can be retrieved by attacking the last round of LED-64. For LED-128, the attacker can retrieve the most significant 64-bit of the key by attacking the round 31. Then, he can retrieve the remaining bits by attacking the round 30.

The post-processing step of LED is different than the post-processing of PRESENT because LED includes a MixColumnSerial operation in its last round.

The MixColumnSerial operation spreads the single faulty nibble in the intermediate state (S) to four nibbles of the faulty ciphertext (C'). Therefore, the reconstruction of the hypothesized faulty intermediate state now requires a hypothesis on 16 key bits, which means that we have 2^{16} different hypotheses. We can solve this problem via a method proposed by Jeong et al. [12]. The solution relies on peeling off the MixColumnSerial operation of the last round by using an equivalent ciphertext ($C^{'*}$), and retrieving an equivalent key (K^*) instead of the actual key (K). The equivalent key and ciphertext satisfy the Equation 3a and 3b, respectively.

$$K^* = MCInv(K) \tag{3a}$$

$$C^{'*} = MCInv(C') \tag{3b}$$

As Equation 3b removes the effect of MixColumnSerial operation on C', one faulty nibble in S corresponds to one faulty nibble in $C^{'*}$. Therefore, we can use the $C^{'*}$ to retrieve each nibble of the K^* with 4-bit key hypotheses. Using S' and $C^{'*}$, we can perform DFIA (Algorithm 1) to retrieve four bits of the K^* using Equation 4. By repeating the procedure for 16 nibbles of the $C^{'*}$, we retrieve all 16 nibbles of the K^*. Then, we apply the MixColumnSerial operation on the K^* to retrieve the actual key K.

$$S^{'} = SboxInv(SRInv(C^{'*} \oplus K^{'*})) \tag{4}$$

The validity of the described solution can be seen from Equations 5a through 5d. The faulty ciphertext C' is computed by Equation 5a. Equation 5b is obtained by applying the MixColumnsInverse operation to both sides of Equation 5a. Using the distributive property of the MixColumnsSerial over the XOR operation, we obtain Equation 5b. Using Equations 3b and 3a we obtain Equation 5d.

$$C' = MC(SR(Sbox(S'))) \oplus K \tag{5a}$$

$$MCInv(C') = MCInv(MC(SR(Sbox(S'))) \oplus K) \tag{5b}$$

$$MCInv(C') = MCInv(MC(SR(Sbox(S')))) \oplus MCInv(K) \tag{5c}$$

$$C^{'*} = SR(Sbox(S')) \oplus K^* \tag{5d}$$

4 Results

We evaluated the proposed DFIA attacks using gate-level simulation. In our gate-level simulations, we first generated 50 random plaintexts. Then, for each of the four implementations, we obtained the ciphertexts for different clock glitch widths. In this experiment, we gradually decreased the clock glitch width from

Table 2. Required Number of Physical Fault Injections for DFIA Attack on PRESENT and LED with 100ps Fault Injection Precision

	Nibble-serial	Round-Serial
PRESENT-80	10	12
PRESENT-128	16	18
LED-64	14	18
LED-128	28	36

$4.6ns$ to $0.6ns$ with $100ps$ step size. At the end, we obtained 40 ciphertexts for each plaintext and each implementation. As can be seen in [13], the selected step size is a reasonable value. We present the analysis of our results in the following subsections. We also study the trade-off between glitch resolution and using multiple plaintexts in DFIA.

4.1 Results of DFIA on PRESENT and LED

Table 2 show the results of DFIA attack on PRESENT and LED implementations. We present the number of required steps, for each implementation, to retrieve the whole unique secret key by using a single plaintext.

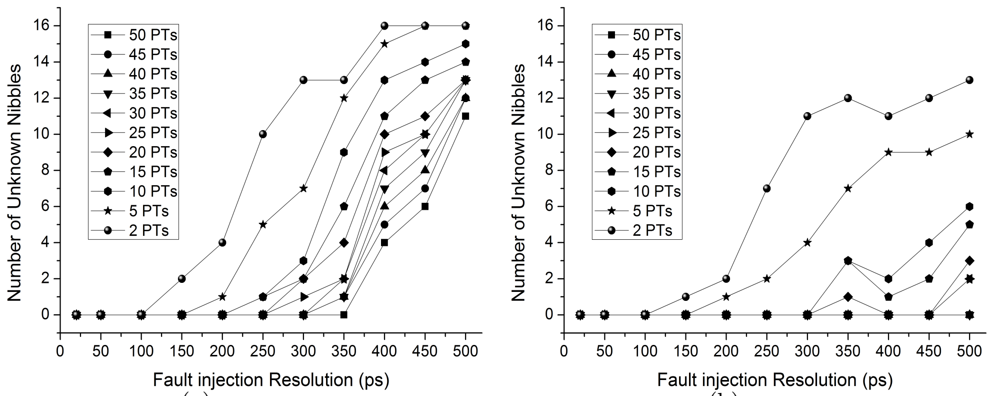
As it can be seen from Table 2, the number of required fault injections to retrieve the correct key for nibble-serial and round-serial implementations are close to each other. Thus, we can conclude that the round-serial and nibble-serial implementations are equally complex for DFIA. DFIA can attack them with the same efficiency. Table 2 also shows the effect of the key size on the number of required fault injections. For example, round-serial LED-128 requires 36 fault injections while round-serial LED-64 requires 18 fault injections.

Compared to the previous fault attacks on PRESENT, we inject fewer faults. The attack in [11] needs up to 150 faulty ciphertexts to retrieve the unique key. The attack proposed in [14] require 48 faulty ciphertext to retrieve the last round key of the algorithm. While the number of required fault injections in the DFIA attack is smaller compared to the mentioned previous works, we also provide a practical fault model.

The previous DFA attacks on LED [12], requires a random faulty nibble to decrease the key search space to 2^8 candidates. Also, the methodology proposed in [15] is based on algebraic equations and injects a single fault to reduce the key search space to $26 \sim 217$ key guesses. The proposed DFIA attack on LED finds the unique correct key guess using additional fault injections of fault injections. However, the biased fault model is practical and easy to achieve for the attacker.

4.2 Trade-off between Fault Injection Resolution and Number of Plaintexts

In this section, we provide the experimental results to verify the efficiency of the extended version of DFIA. We investigate the relationship between fault injection



(a) PRESENT (b) LED

Fig. 7. Trade-off Between Fault Injection Resolution and Number of Plaintexts used for (a)PRESENT and (b)LED

resolution and the number of plaintexts that DFIA needs to retrieve the key. As our fault injection means is clock glitching, our fault injection resolution is the minimum increment or decrement in the clock glitch width that we can achieve. In this experiment, we apply DFIA attacks (Algorithm 1) on our PRE-rs and LED-ns implementations for different fault injection resolutions, from $20ps$ to $500ps$, and for different number of plaintexts, which ranges from 2 to 50. Then, we count the number of the key nibbles that DFIA cannot retrieve under a given fault injection resolution and a given number of plaintexts. We call such nibbles as unknown nibbles throughout this section.

Fig. 7(a) and Fig. 7(b) present the results for PRE-rs and LED-rs implementations, respectively. In these figures, the Y axis shows the number of unknown nibbles out of 16 nibbles and the X axis shows clock glitch resolutions. Each data line in the graphs corresponds to a different number of applied plaintexts (PTs). Fig. 7(a) and Fig. 7(b) show two important behaviors for both LED-rs and PRE-rs. For a given fault injection resolution, using more plaintexts decreases the number of unknown nibbles. For a fixed number of plaintexts, the number of unknown nibbles decreases as the fault injection resolution increases (i.e., clock glitch step size decreases). An adversary can decrease the number of unknown nibbles either by increasing the fault injection resolution or by increasing the number of plaintexts (i.e., encryptions). Therefore, we can conclude that there is a trade-off between the fault injection resolution and the number of required plaintexts. Due to this trade-off, DFIA can still efficiently retrieve the key when the fault injection equipment has a low resolution or when few biased faults are available.

5 Related Work

Ghalaty et al. [5] have a discussion on the differences of the DFIA attack with other types of attack such as DFA [4], FSA [8] and DPA [3]. In this section, we will talk about the previous types of fault attacks that use the concept of biased fault and explain their differences with DFIA. Although DFIA [5] is not the first work that utilizes the fault bias as a fault model [9], [16], [17], [11], it is the first work that defines biased fault just beyond the fault sensitivity.

Lashermes et al. [9] assume a biased fault model and use the concept of hypothesis test on a distinguisher (i.e. Shannon entropy). However, in order for their method to converge in a practical time, they need a method to quantify the characteristic error distribution of the fault injection means. For this purpose, they need to profile the device under attack for different data sets. On the other hand, DFIA does not require any profiling phase.

Sakiyama et al. [16] and De Santis et al. [11] apply similar methodology to AES and PRESENT algorithms. They assume that faults will cause bias in the intermediate values, for example because of stuck-at faults in the intermediate value. In contrast, DFIA assumes that the fault itself is biased: The faulty intermediate values must differ in a small number of bits from the non-faulty value. But the faulty intermediate values do not have to be biased.

Jarvinen et al. [18] also propose DFA attack based on biased fault injection model. However, their method is more similar to the DFA attacks. Their definition of fault model is also different from DFIA's fault model. In this paper, the author defines the biased fault as the fact that the probability of stuck-at-1 or stuck-at-0 is higher compared to the other one. They assume that based on the fault injection method, the attacker knows the value of faulty bit and can use mathematical equation to reverse the faulty ciphertext and get the key.

6 Conclusion

In this paper, we propose a DFIA on round-serial and nibble-serial implementations of PRESENT and LED. Based on our result, we can retrieve the unique key guess for each algorithm with a reasonable number of fault injections. Our method of fault injection is the clock glitching in this paper which is a very cheap and easy way of attacking for the adversary. We also study the relation between the number of plaintexts (encryptions) used, and the precision of the fault injection equipment.

7 Acknowledgment

This research was supported through the National Science Foundation Grant 1441710, Grant 1115839, and through the Semiconductor Research Corporation.

References

1. ISO: Information Technology–Security Techniques–Lightweight Cryptography–Part 2: Block Ciphers. ISO/IEC 29192-2:2012, International Organization for Standardization (2012)
2. Atzori, L., Iera, A., Morabito, G.: The Internet of Things: A Survey. Computer networks **54** (2010) 2787–2805
3. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. In: Advances in Cryptology (CRYPTO), Springer (1999) 388–397
4. Biham, E., Shamir, A.: Differential Fault Analysis of Secret Key Cryptosystems. In: Advances in Cryptology (CRYPTO). Springer (1997) 513–525
5. Farhady Ghalaty, N., Yuce, B., Taha, M., Schaumont, P.: Differential Fault Intensity Analysis. In: 2014 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), IEEE (2014) 34–43
6. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-lightweight Block Cipher. Springer (2007)
7. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED Block Cipher. In: Cryptographic Hardware and Embedded Systems–CHES. Springer (2011) 326–341
8. Li, Y., Sakiyama, K., Gomisawa, S., Fukunaga, T., Takahashi, J., Ohta, K.: Fault Sensitivity Analysis. In: Cryptographic Hardware and Embedded Systems-CHES. Springer (2010) 320–334
9. Lashermes, R., Reymond, G., Dutertre, J., Fournier, J., Robisson, B., Tria, A.: A DFA on AES based on the Entropy of Error Distributions. In: 2012 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), IEEE (2012) 34–43
10. Altera Corporation: ModelSim Altera Starter Edition. (Available:www.altera.com)
11. De Santis, F., Guillen, O., Sakic, E., Sigl, G.: Ciphertext-Only Fault Attacks on PRESENT. Third International Workshop on Lightweight Cryptography for Security and Privacy (2014) 84–105
12. Jeong, K., Lee, C.: Differential Fault Analysis on Block Cipher LED-64. In: Future Information Technology, Application, and Service. Springer (2012) 747–755
13. Endo, S., Sugawara, T., Homma, N., Aoki, T., Satoh, A.: An On-chip Glitchy-clock Generator for Testing Fault Injection Attacks. Journal of Cryptographic Engineering **1** (2011) 265–270
14. Bagheri, N., Ebrahimpour, R., Ghaedi, N.: New Differential Fault Analysis on PRESENT. EURASIP Journal on Advances in Signal Processing **2013** (2013)
15. Zhao, X.j., Guo, S., Zhang, F., Wang, T., Shi, Z., Ji, K.: Algebraic Differential Fault Attacks on LED Using a Single Fault Injection. IACR Cryptology ePrint Archive **2012** (2012) 347
16. Sakiyama, K., Yang, L., Shigeto, G., Mitsugu, I., Naofumi, H., Takafumi, A., Kazuo, O., et al.: Practical DFA Strategy for AES Under Limited-access Conditions (Preprint). **55** (2014)
17. Fuhr, T., Jaulmes, E., Lomné, V., Thillard, A.: Fault Attacks on AES with Faulty Ciphertexts Only. In: 2013 IEEE Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), IEEE (2013) 108–118
18. Jarvinen, K., Blondeau, C., Page, D., Tunstall, M.: Harnessing Biased Faults in Attacks on ECC-Based Signature Schemes. In: 2012 IEEE Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), IEEE (2012) 72–82