Differential Fault Intensity Analysis

Nahid Farhady Ghalaty, Bilgiday Yuce, Mostafa Taha, Patrick Schaumont
Bradley Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, USA
{farhady,bilgiday,mtaha,schaum}@vt.edu

***Abstract*—Recent research has demonstrated that there is no sharp distinction between passive attacks based on side-channel leakage and active attacks based on fault injection. Fault behavior can be processed as side-channel information, offering all the benefits of Differential Power Analysis including noise averaging and hypothesis testing by correlation. This paper introduces Differential Fault Intensity Analysis, which combines the principles of Differential Power Analysis and fault injection. We observe that most faults are biased - such as single-bit, two-bit, or three-bit errors in a byte - and that this property can reveal the secret key through a hypothesis test. Unlike Differential Fault Analysis, we do not require precise analysis of the fault propagation. Unlike Fault Sensitivity Analysis, we do not require a fault sensitivity profile for the device under attack. We demonstrate our method on an FPGA implementation of AES with a fault injection model. We find that with an average of 7 fault injections, we can reconstruct a full 128-bit AES key.**

***Index Terms*—Fault Analysis; AES; Fault Injection; Fault Intensity.**

I. Introduction

Traditionally, implementation attacks on secure embedded systems come in two flavors: passive side-channel analysis (SCA), and active fault analysis (FA). SCA observes the physical implementation of cryptography and uses implementation effects (time, power, etc) to reverse-engineer the secret key. The most powerful SCA, *differential power analysis*, enables the adversary to accumulate knowledge over many encryptions [1]. FA, on the other hand, injects faults into the cryptographic implementation, and analyzes the system response under the assumption of a specific fault model [2].

This paper introduces Differential Fault Intensity Analysis (DFIA), a fault attack that combines the principles of DPA and fault injection. DFIA is based on a differential analysis of the system response under fault injection with varying intensity. Changing the fault intensity causes a non-uniform fault response. For example, a biased fault on a full byte of a secret state variable would cause a single-bit, two-bit, or three-bit error with high probability [3], [4]. Under a biased fault model, small changes to a faulty state variable are therefore more likely than big, random changes. This is the basis for the differential technique. In DFIA, the attacker injects faults with different intensities into the cryptographic module to get faulty ciphertexts. Next, he derives an intermediate, key-dependent state variable under a given key hypothesis (similar to DPA). Finally, he will select the key hypothesis that is most likely under the biased fault model. Indeed, due to the non-linear properties of cryptographic modules, a wrong key hypothesis will always infer big, random changes as a result of a fault.

In comparison to earlier contributions, DFIA has three benefits. The first one is that DFIA does not require fault path analysis of the cipher, in the same manner as required for Differential Fault Analysis (DFA). In DFA, the adversary compares the response of a cipher with and without fault injection. DFA thus requires the adversary to obtain both faulty and fault-free ciphertext. The secret key is inferred by analyzing the fault propagation under assumption of a fault model [5]. DFA does not average over different fault injections, but treats each fault separately as part of a system of equations [6]–[8].

The second benefit of DFIA is that DFIA works with a looser fault model. For example, earlier work has assumed precise fault effects such as stuck-at-0 or stuck-at-1 faults [3], or precise control on the position where the fault is injected [6]. DFIA, on the other hand, only assumes that the fault model is biased.

The third advantage is that DFIA does not require a profiling phase. In contrast, the recently proposed Fault Sensitivity Analysis (FSA) is based on two phases [5]. The first phase does an exhaustive profiling of the circuit to characterize its response under varying fault intensity. This allows the adversary to associate the value of secret state variables to the fault sensitivity of the circuit. In the second phase, the attacker will measure the fault sensitivity of a circuit with an unknown key, and derive the key value based on the most likely secret state variable. The profiling phase is an important condition for FSA to work, since the attacker needs to associate fault sensitivity and secret state values. DFIA, on the other hand, does not need the association to fault sensitivity, and it can directly derive the most likely secret state.

The paper is organized as follows. In Section II, we describe the fault model. Section III explains the proposed attack procedure and provides an example of the proposed attack for the case of an AES implementation. Section IV describes the experimental setup that was used to derive results. The results prove the fault model existence, the key retrieval cost for AES round key byte and whole AES key. Section V gives a brief overview on the previous works. This section compares DFIA with other types of attacks. Finally Section VI concludes the paper.

978-1-4799-6292-1/14 $31.00 © 2014 IEEE
DOI 10.1109/FDTC.2014.15

II. Fault Model

This paper proposes DFIA. We will illustrate it for hardware implementations of cryptographic algorithms on FPGA. Like other fault attacks, DFIA starts with the injection of faults. There are two important factors to any form of fault injection.

1) **Control of the location of the fault:** In a software implementation, the fault location refers to the variable or instruction chosen for fault induction. Because software is sequential, the fault location is closely associated with the fault timing. In DFIA, we assume that the attacker can inject faults at the required timing and that he is aware of the fault location in the program [9]. This is a reasonable assumption, even for unknown architectures. The adversary can characterize implementation aspects such as encryption rounds using side-channel leakage. For example, in a typical software implementation of AES, it is usually easy to distinguish 10 different rounds [10].
2) **Fault Model:** Depending on the type of fault injection, different types of faults can occur, with different granularities. We can have multi-byte faults or single-byte faults. Furthermore, faults can be stuck-at faults, bit flips, random faults and sets or resets.
   One other aspect of the fault model is the number of faulty bits induced. Previous work has often considered the fault model a *random effect on one byte*, meaning that the fault injection can change the value of a register to any random value [11], [12] and [13]. However, we specifically assume that the fault in a single byte is biased. For example, in an analysis of the fault effect of radiation, more than 90% of faults are Single Bit Upsets (SBU) and the probability of inducing single-bit and two-bit fault successively is 70% [14], [15]. Also for directed fault injections (such as clock glitches), the number of faulty bits are mostly one-bit or two-bits [15]. Hence, we make the following assumptions for the DFIA fault model:
   - We assume that a random byte of the state will be affected by the fault injection.
   - The distribution of faults in this byte will be biased. The attacker can manually increase or decrease the number of faulty bits by controlling the fault intensity [4], [16].

III. Differential Fault Intensity Analysis

In this section, we describe our attack. First, we define a generic procedure, and then we apply it to AES.

A. The Attack Procedure

The problem definition is as follows: Given a cryptographic algorithm, the fault injection tool and enough pairs of faulty ciphertexts corresponding to the same input, we want to find the round key. We make the following assumptions:

- The adversary knows what cryptographic algorithm is executing.
- The adversary is able to inject a fault in an intermediate variable S that produces a key-dependent observable output. This is similar to the way in which DPA selects intermediate variables.
- The fault can be injected with varying intensity, with a limited change to the intermediate variable.

Table I
Symbols of DFIA Attack Procedure

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

Algorithm 1: DFIA Attack Procedure

Assume *Cryptographic Algorithm, Fault Injection Tool*;
Result *Correct Key Guess* ;
//Step1 *Fault Injection*;
forall the *Fault* q *such that* $1 \leq q \leq Q$ **do**
  Obtain faulty ciphertext C'_q;
  forall the *Key Hypothesis* k **do**
    Obtain faulty state $S'_k, q = f(C'_q, k)$;
//Step2 *Post-process*;
forall the *Key Hypothesis* k **do**
  Calculate $\rho_k = \sum_{n=1}^{Q} \sum_{m=1}^{n-1} HD(S'_{k,n}, S'_{k,m})$;
//Step3 *Key Hypothesis*;
$K = \min \rho_k$;

Table I defines the notations of the DFIA attack procedure. DFIA uses three steps. Algorithm 1 explains the DFIA attack procedure. This algorithm consists of the following steps.

Step 1: The attacker induces a fault into S while running the cryptographic algorithm with input P. The fault will change the value of S to a faulty value. The output of the algorithm will be C'_q.

The attacker is in possession of the values of C'_q. Under a key hypothesis k, he can compute $S'_{k,q}$ as a function of C'_q and k. The attacker will generate $S'_{k,q}$ for every possible key hypothesis.

Step 2: The attacker computes Equation 1 for each key hypothesis k. This equation adds up the Hamming Distance between the value of $S'_{k,q}$ for current fault injection and any previous one. The Hamming Distance between $S'_{k,n}$ and $S'_{k,m}$ is equal to the difference between the number of injected faults in fault injection n and fault injection m.

$$\rho_k = \sum_{n=1}^{Q} \sum_{m=1}^{n-1} HD(S'_{k,n}, S'_{k,m}) \tag{1}$$

Step 3: The attacker evaluates the Hamming Distance among faulty state variables testing for the distance that is

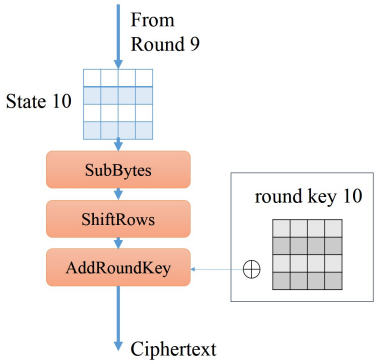


Figure 1. AES-128 encryption algorithm

most likely under the assumed biased fault model. To get to this point, the attacker should look for the minimum of cumulative Hamming Distance (ρ_k) among all key hypotheses. The reason behind looking for minimum is that only under the correct key guess the number of injected faults will correspond to the fault intensity, otherwise the Hamming Distance will be a random or unpredictable number. If no conclusion can be made on the most likely key, the attacker will increase Q and repeat the process.

Hence, similar to DPA, the DFIA attack tests the most likely model corresponding to an observation (measurement). Adding more observations improves the hypothesis test.

B. DFIA on AES

We now describe our algorithm as applied to the Advanced Encryption Standard (AES) [17]. Figure 1 shows the flow diagram of AES. We assume 128-bit AES (10 rounds).

We use the same notations as before, with the extension that S represents the AES state at round 10, and K represents the round key for round 10.

If a fault q is induced in S, it will only corrupt a single byte of C since the last AES round does not perform MixColumns diffusion. The adversary then collects multiple faulty ciphertexts C'_1, C'_2, ..., C'_Q under the same plaintext intput P. We know that

$$C'_q = k \oplus ShiftRows(SubBytes({S_q}'')) \tag{2}$$

Therefore, the faulty state can be computed as

$${S_q}' = SubBytesInverse(ShiftRowsInverse(k \oplus C'_q)) \tag{3}$$

We arrange the observations in a table such as Table II, which we call the key hypothesis table. Now, only one of the 256 key guesses is true. Because of the biased fault model, the true faulty state bytes must be close to each other in terms of Hamming Distance. Hence, under a given key hypothesis, we test the distance between faulty state bytes by calculating ρ_k. Only for a correct key guess will a minimum be found. As before, we may need multiple fault injections to uniquely pinpoint the most likely roundkey.

Table II
KEY HYPOTHESIS TABLE

		Key Hypothesis K			
		k_0	k_1	...	k_n
Fault Injection	Faulty Ciphertexts	Apply Equation 3 to get S'_q			
1	C'_1	$S'_{0,1}$	$S'_{1,1}$	...	$S'_{n,1}$
2	C'_2	$S'_{0,2}$	$S'_{1,2}$	...	$S'_{n,2}$
...	...	...	...	...	...
Q	C'_Q	$S'_{0,Q}$	$S'_{1,Q}$	...	$S'_{n,Q}$
Apply Equation 1 to get ρ_k		ρ_0	ρ_1	...	ρ_n

C. Methodology for Selecting Fault Set

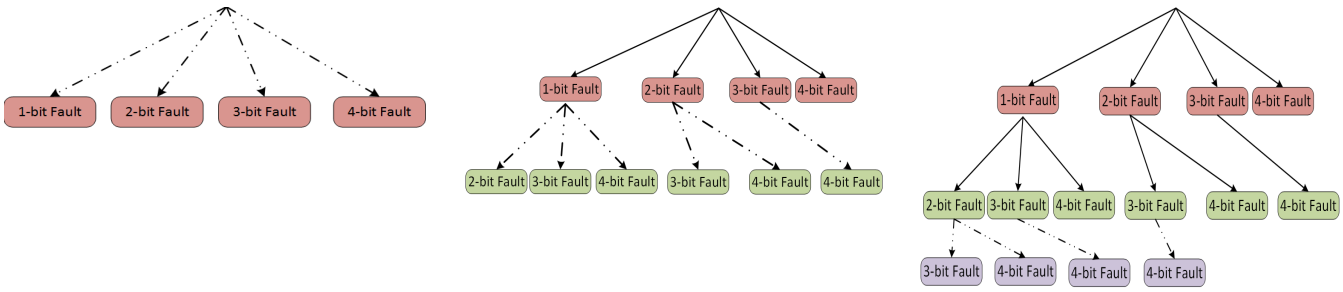
The previous section demonstrated how a set of biased faults can be used to construct a hypothesis test leading to the correct key. In this section, we elaborate how to construct the set of biased faults such that the hypothesis test shows the quickest convergence. From a practical point of view, the adversary is interested in using as few faults as possible, since every fault injection costs time and effort.

To build an understanding of what makes a good set of biased faults, we performed the following experiment. We wrote a simulator for the AES algorithm that supports the injection of biased faults in the last round. The simulator allows the injection of single-bit, two-bit, three-bit and four-bit errors in a state byte. For example, injection of a two-bit biased fault in a state byte means that the simulator randomly selects two bits from the state byte. Then, it flips the selected bits to random values of 0 or 1. Hence, two-bit fault injection means that at most two bits in a byte have faulty values. Observe that this fault model only gives us control over the *number* of bits that may be affected, not over their location or their final value.

We argue that the single-bit, two-bit, three-bit and four-bit biased fault models can be used to simulate variation of fault intensity. At low fault intensity, a single-bit fault is more likely, while at higher intensities, multi-bit faults are more likely. This effect has been observed by other authors as well [16], [18] , and we also confirmed it through practical experiments (See section IV.B).

Using the simulator, we can now explore different strategies for fault injection campaigns. The most simple set of biased faults is a set with one single fault at a given intensity level (single-bit, two-bit, three-bit or four-bit). After injection of a fault, the simulator computes the number of possible key bytes under the given fault model. For example, after injection of a two-bit fault, the simulator finds the number of key candidates that, according to Equation 3, predicts a faulty state with a Hamming Distance of 1, or 2 from the original state. We repeat this for 256 different plaintext inputs, each time finding the number of key candidates.

Figure 2(top) shows different scenarios of injecting faults into the state byte. Each level of the tree shows different fault intensities at each trial. Figure 2(a) shows the possible fault



(a) First Level of Fault Injection (b) Second Level of Fault Injection (c) Third Level of Fault Injection

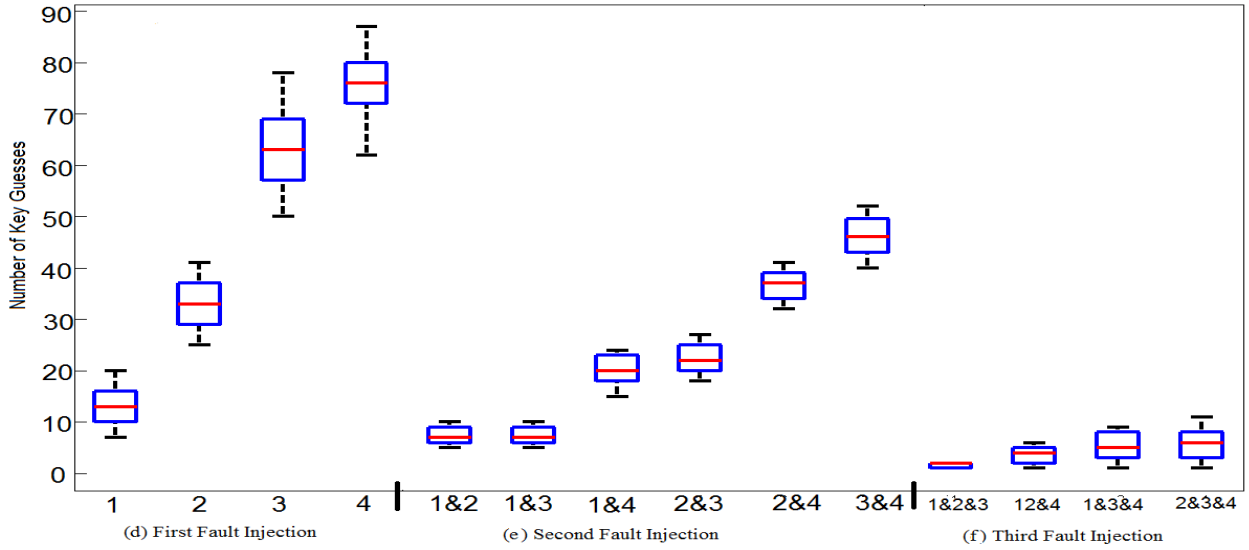


Figure 2. Effect of Choosing Fault Set on Key Recovery Convergence

intensities for the first fault injection trial. In the first trial, the attacker can inject either single-bit, two-bit, three-bit or four-bit faults into the state byte. Figure 2(d) shows how many key guesses could be the correct key after the first trial of fault injection. In this figure, the X axis shows the sequence of faults injected so far and each sequence of fault injections are labeled by the corresponding fault intensities. Y axis shows the number of possible correct key guesses corresponding to the each sequence of fault injections. The numerical data in Figure 2(bottom) is shown as a box-and-whisker diagram.

Figure 2(d) shows that strongly-biased faults (such as single-bit faults) leave less ambiguity about the correct key than weakly-biased faults (such as four-bit faults). The reason for this is simply a consequence of Hamming Distance computation over a byte: Under a single-bit error, a state byte can only map in 7 other byte values (of the possible 256). Under a two-bit error, however, a state byte can map in up to 55 other possible values.

In more elaborate fault injection campaigns, we will inject multiple faults, with varying intensity. Figure 2(b) shows the next level of the tree. In the second trial, the attacker can increase the fault intensity. Therefore, the second fault injection after single-bit could be two-bit, three bit or four-bit. Figure 2(e) shows the number of possible key guesses after post-processing the information of both trials. As an example, the number of possible key guesses after single-bit and two-bit fault injection is 7 in average. This number is smaller compared to injecting only single-bit (13 key guesses based on Figure 2(d)) or two-bit (33 key guesses based on Figure 2(d)) faults. The reason is that the attacker is now in possession of two sets of possible key guesses and can find the subscription of the two sets.

Finally, at the last level, the attacker injects the third fault (Figure 2(c)). Only one or two key guesses are left if the attacker has injected one, two and three faulty bits (Figure 2(f)). But there exists up to 11 key guesses if the attacker injects two, three and four bit faults. This demonstrates the fact that by injecting more accurate faults (single-bit), the key retrieval procedure takes less execution time and less trials. We also see that multiple fault injections quickly reduce the number of viable key candidates.

IV. Experimental Setup

In order to evaluate our claims, we implemented an experimental setup as shown in Figure 3. The experimental setup consists of an FPGA (Altera Cyclone 4 E), a computer, and an

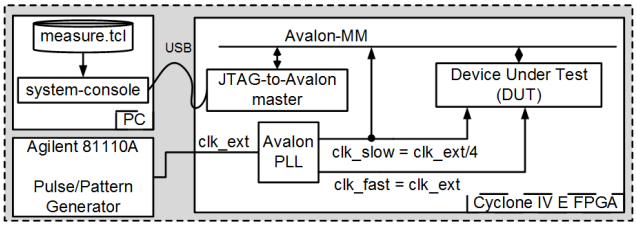


Figure 3. Block Diagram of the Experimental Setup

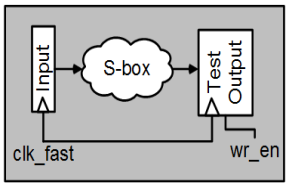


Figure 4. DUT Architecture for the Biased Fault Experiment

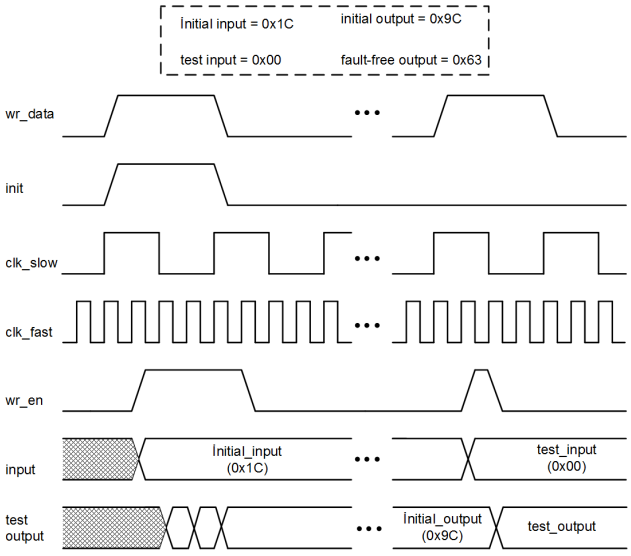


Figure 5. A High-level Timing Diagram for the Biased Fault Experiment

external clock signal generator (Agilent 81110A Pulse/Pattern Generator). On the FPGA, we have three blocks: Device under test (DUT), JTAG-to-Avalon master, and Avalon PLL. The DUT is connected to an Avalon-MM bus via Avalon memory-mapped slave interface. The DUT is controlled by a PC through a JTAG-to-Avalon master bridge. The PC uses an FPGA debug program with a Tcl script in order to create read/write transactions on the Avalon bus. In our setup, we use clock glitches for fault injection and control the fault intensity by increasing/decreasing the frequency of the clock signal. Hence, the system has two clock signals *clk_fast* and *clk_slow*, which are derived from an external clock signal (*clk_ext*) using a phased-locked loop block (Avalon PLL). The external clock signal is generated by the external clock signal generator and it can take frequency values up to 330 MHz. The *clk_slow* signal drives the Avalon-MM bus and it is used for fault-free operation of the DUT. The frequency of the *clk_slow* signal is one fourth of the external clock signal frequency. The *clk_fast* signal is used to create clock glitches for fault injection and its frequency is equal to the external clock signal frequency.

Using the experimental setup with two different DUT blocks, we carried out two experiments:

1) In the first experiment, we investigate the behavior of four different S-box implementations by using them as DUTs in order to evaluate our biased fault model.
2) In the second experiment, we evaluate the DFIA attack on AES proposed in Section III by using an FPGA implementation of AES block cipher.

The details of the experimental setups will be explained in the following subsections.

A. Biased Fault Experiment for S-Box Architectures

In this experiment, we used four different FPGA implementations of AES S-box block as the DUT: PPRM1 S-box [19], Boyar-Peralta S-box [20], Canright S-box [21], and LUT-based S-box. In the DUT block of Figure 3, we place a combinational S-box implementation between a set of input/output registers as it is shown in Figure 4.

Our approach in this experiment is, for different external clock frequencies, applying a test input to the DUT, and then, computing the Hamming distance between the fault-free output and the test output corresponding to this test input. The fault-free output can be easily obtained by looking up the AES S-box table. The test output is the value obtained in the test output register of Figure 4 for the applied test input and clock frequency. Before applying the test input, we reset the test output register to the bitwise inverse of the fault-free output value via an initial input value. Hence, the Hamming distance between the fault-free and the test outputs gives the number of faulty bits obtained for the applied test input and clock frequency. Similar to the fault-free output value, the initial output value can be easily determined by looking up the inverse AES S-box table.

In our experiment, we arbitrarily selected 0x00 as the test input, and thus, the corresponding fault-free output value is 0x63. In order to reset the test output register to 0x9C (i.e. the inverse of 0x63) before applying the test input, our initial input value is 0x1C. A high-level timing diagram for the experiment is shown in Figure 5. The experiment starts with setting the input register to an initial input value to obtain the initial output value in the test output register. Then, the test input is applied to obtain the test output. These steps are controlled by a data write (*wr_data*) and an initialization (init) signal, which are generated by the Tcl script (measure.tcl) and are transferred to the DUT via JTAG-to-Avalon master block. Finally, the Hamming distance between the test and the fault-free outputs are computed.

In this experiment, we applied the above steps for four aforementioned S-box implementations and 33 different external clock frequency values. As a result, we obtained 33

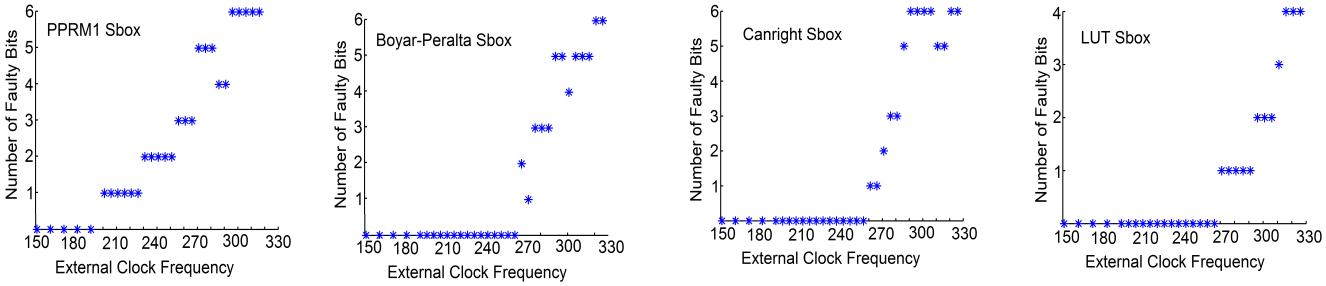


Figure 6. Biased Fault Behavior for Different S-box Implementations

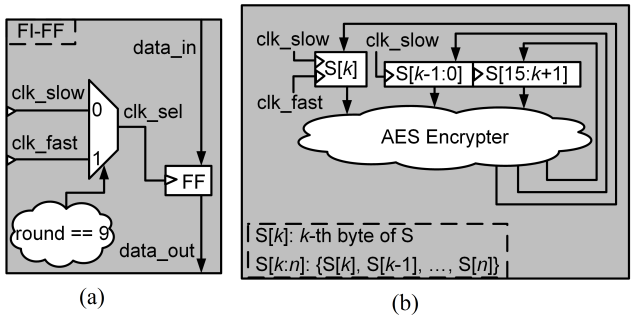


Figure 7. (a) Fault-injection Flip-flop (FI-FF) Architecture. (b) The DUT Architecture for Fault Injection into k-th Byte of the AES State

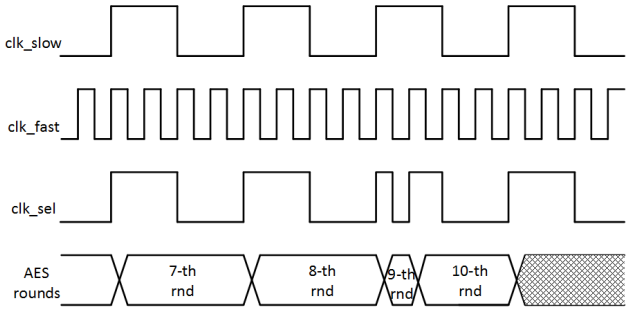


Figure 8. A High-level Timing Diagram for FI-FF

experimental results for each S-box architecture. The experimental results are given in Figure 6. For each graph of Figure 6, the X axis represents the external clock frequency and the Y axis represents the Hamming distance between the fault-free and the test outputs. As it can be seen from Figure 6, the Hamming distance between the fault-free and the test outputs (i.e. the number of faulty output bits) increases with the clock frequency.

B. DFIA Experiment on AES

In this experiment, we evaluated the proposed attack on AES. We used the experimental setup given in Figure 3 with an AES block cipher implementation as the DUT. Our AES implementation is a register-transfer level Verilog definition of AES encryption and uses PPRM1 S-box architecture. In this experiment, our method is injecting biased faults into the AES state at 10-th round. In order to inject a fault into a position of the AES state, we use a fault-injection flip-flop (FI-FF) for this position rather than using a regular flip-flop (FF). As it is shown in Figure 7(a), the FI-FF includes a multiplexer and control logic in addition to a regular FF to select between two different clock signals. An FI-FF uses a high-frequency clock signal (*clk_fast*) at 9-th round of the AES, while using a low-frequency clock signal (*clk_slow*) at the other rounds. In Figure 7(b), we show a DUT architecture for fault injection into k-th byte of the AES state. In Figure 7, S[k] denotes the k-th byte of the AES state S, while S[k:n] denotes all of the AES state bytes from k-th byte to n-th byte. In this architecture, we replaced regular FFs used for k-th byte of the AES state with FI-FFs. Using this architecture, we can inject a clock glitch into the k-th byte of the AES state as it can be seen from the high-level timing diagram provided in Figure 8. In the DUT design, our assumption is that the fault injection affects only the state registers of an AES implementation. A reasonable extension to our method would be considering the effects of the fault injection on the other registers of AES as well. Hence, this is left as a future work.

In order to evaluate a key retrieval process by DFIA, we applied the attack on AES algorithm by different fault models. Hence, our experiment consists of two parts:

1) In the first part, we first injected faults with different intensities (i.e. for different external clock frequencies) into the most significant byte of the AES state ($S[15]$) in order to show how we retrieve a byte of the key using DFIA. Then, we injected faults into the separate bytes of the AES state ($S[14], S[13], ..., S[0]$) to compute how many single-byte fault injections are required, on the average, to retrieve the key via fault injections with single-byte granularity. We used the architecture given Figure 7(b) with different values of k ($k \in [15, 0]$) in this part of the experiment.
2) In the second part, we injected faults with different intensities into the whole AES state by changing all of the regular AES state FFs into FI-FFs. Our purpose in this experiment was to see how many fault injections with 16-byte granularity are needed to retrieve the key value.

In our experiment, we selected one arbitrary plaintext and an

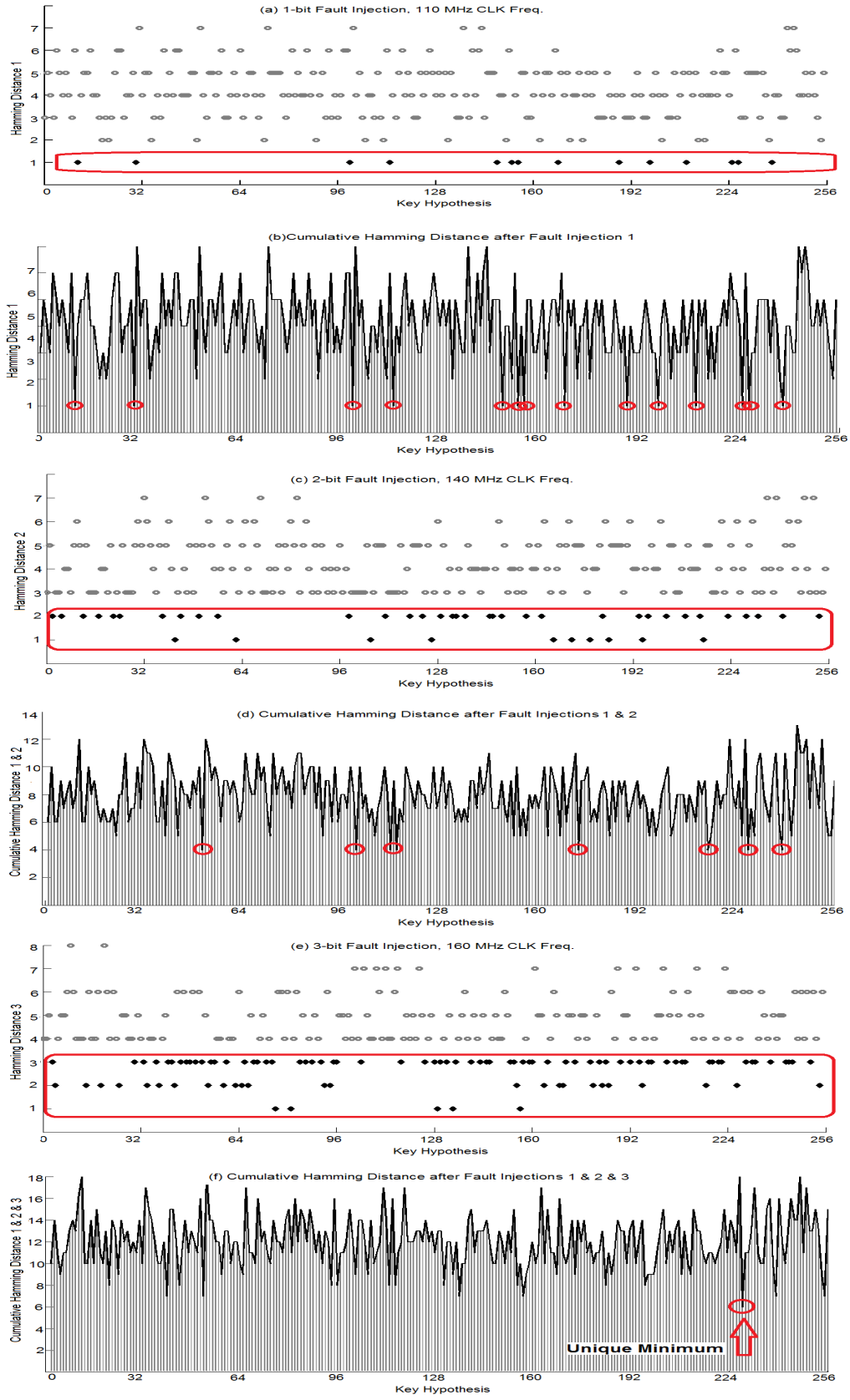


Figure 9. Results of a DFIA Attack Procedure on the Most Significant Byte of AES Algorithm

arbitrary key value. We obtained ciphertexts for fault injections with different intensities. Finally, we analyzed the results with a post-processing script executed in Matlab in order to find the total number of fault injections for a complete retrieval of the AES encryption key.

In this part, our assumption is that the attacker can affect only one byte of the AES state at 10-th round. In order to inject faults into separate bytes of the AES state, we used the architecture given Figure 7(b) with different values of k ($k \in [15, 0]$). The attacker can increase the external clock frequency and generate different faulty ciphertexts. Based on results shown in Figure 6, the attacker is aware that the number of faulty bits is in proportion with the clock frequency. For each byte k of the AES state, we injected faults at four different intensities using different external clock frequencies: faultless, single-bit errors, two-bit errors, and three-bit errors. Based on the fault models, the Hamming Distance from the correct state to the faulty state should be one, two and three for the single-bit, two-bit and three-bit errors respectively.

Before proceeding to the results, we provide an example to show how we retrieve a single byte of the key using the experimental results: Figure 9 shows the results of the attack applied to the most significant byte of the AES state ($S[15]$).

1) AES Key Retrieval Process with One Byte Fault Injection: Figure 9(a),(c),(e) plot the Hamming Distance between the fault-free state and the faulty state for one-bit, two-bit, and three-bit errors respectively. The X axis shows the key hypothesis, while the Y axis shows the Hamming Distance as computed with Table II. Figure 9(b),(d),(f) show the bar graph after post-processing the results of different trials of fault injection.

Because of the varying fault intensity, we know that the correct key in Figure 9(a) can only have a Hamming distance of 1. The black dots show the possible key values. After post-processing the results, there are 14 possible keys (Figure 9(b)). These key guesses are shown by red circles. Hence, we conclude that a single-bit fault injection is insufficient to reveal the key. We therefore increase the fault intensity, and inject a two-bit error. The Hamming Distance graph now looks as in Figure 9(c). We expect the correct key is among those who show a Hamming Distance of 2. We can also see that some key choices lead to a Hamming distance of only one. However, under two-bit fault injections, any wrong key choice will eventually end up at a Hamming Distances higher than 2, while the correct key will always remain at or lower than 2. We post-process the results of the one-bit fault injection and the two-bit fault injection by accumulating the Hamming Distance for each key guess. Again we find multiple candidates(Figure 9(d)). We increase the fault intensity once more, to three-bit errors. In this case, the correct key is among those with Hamming Distance at or lower than 3(Figure 9(e)).

We can now post-process the results of DFIA. The post-processing is simple: we accumulate the Hamming Distance for the respective keys for all graphs, and we look for the minimum. In this case, because we collected a graph at fault intensity 1, 2, and 3, we expect the minimum Hamming Distance in the overall graph to be at or lower than 6. Indeed, in the accumulated graph, we can find only a single key which shows a minimum Hamming Distance of 6, which allows us to conclude that this key is the correct one(Figure 9(f)).

It is important to note that the attacker might not be aware of the number of faulty bits in the state variable. Since the attacker increases the clock frequency for fault injection, he expects the number of faulty bits to increase. Therefore, for the key retrieval process, he still expects the correct key to be among the key guesses with the minimum summation of Hamming Distances.

Based on the results obtained for different bytes of the AES state, the average number of fault injections required for retrieving one byte of the key is 4.6. Furthermore, we need 68 fault injections to retrieve the whole AES key if the accuracy of the fault injection is one state byte.

2) AES Key Retrieval Process with Multiple Byte Fault Injection: In this part, we injected faults into the whole AES state by changing all of the regular state FFs with FI-FFs in order to find the total number of fault injection with multiple byte fault injection for a complete key retrieval of the AES algorithm. We obtained ciphertexts for 24 different external clock frequencies, which are uniformly distributed from 100 MHz to 330 MHz. The results of this part shows that the whole AES key can be retrieved with 7 fault injections if the accuracy of the fault injection is whole state (i.e. 16 state bytes).

V. Comparison with Previous Research

DFA attacks on AES algorithm can be classified into many categories [13]. Previous works have tried to reduce the number of fault injections, using different fault models and different rounds in the AES algorithm. DFA attacks that have been proposed so far, have a high complexity due to two reasons. The first one is that they require a very precise fault injection method, and the second reason is that they require a detailed understanding of the fault propagation in the algorithm, in order to identify the correct key.

Piret has shown that the number of faults required to retrieve the AES-128 key is two [6], but they inject the fault between round two and three of AES and consequently have very complex formulas for the key retrieval algorithm. On the other hand, methods like the ones proposed in [7] could reduce the number of fault injections to one byte while they require a exhaustive search among 2^8 candidates. Kim shows a method that can retrieve the key of AES-192 with two fault injections and AES-256 with three fault injections [8]. Fuhr proposes a new DFA attack that only requires faulty ciphertexts for the key retrieval. Their method seems most close to ours, however they rely on a specific stuck-at fault model [3].

Li introduces Fault Sensitivity Analysis (FSA) [5]. This attack requires an exhaustive profiling phase on the intermediate values to find the correlation graph. The correlation graph shows the connection between the fault sensitivity and the value of the input data. The attacker then measures the fault sensitivity for every unknown key guess and finds the best candidate by comparing the graphs to the golden model. Li

Table III
SIMILARITIES AND DIFFERENCES OF DIFFERENT ATTACKS AND DFIA

	Source of Leakage	Extraction of the Key	Model of Leakage	Profiling
DPA [1]	Side Channel (Power)	Model Testing	Power Model	No
DFA [22], [23], [6]	Fault	Exact Analysis	None (Actual Ciphertext)	No
FSA [5]	Fault Sensitivity	Model Testing	Fault Sensitivity Table	Yes
DFIA	Biased Faults	Model Testing	Fault Model	No

shows that 50 trials are sufficient to identify the key for the AES-128 algorithm.

Table III shows the differences and similarities of DFIA and other methods. Here is a brief description of the columns:

Source of Leakage is what the adversary uses as extra information to perform the attack. Besides classic side-channel leakage, the table lists faults and fault sensitivity and biased fault. The differences between the latter three is subtle but important. Faults leak information because they enable small difference to propagate through the implementation and reach the output. Fault Sensitivity is a point at which the output of the device becomes faulty by increasing the Fault Intensity. Fault Intensity, on the other hand, leaks information because it induces a biased fault. How the fault propagates to the output doesn't matter.

Extraction of the Key column shows the method that each algorithm exploits for key retrieval. Model-based methods (DPA, FSA, DFIA) construct a model (an abstracted version) of the implementation, that can be used to test if the actual observed leakage (measurement) matches the assumption (model). Exact methods (DFA), on the other hand, cannot make use of such models: They need to formulate the key search directly in terms of the algorithm under analysis.

Model of Leakage shows the different models used.

Profiling column shows whether the attack requires any exhaustive profiling phase. The only method that requires profiling on an exhaustive set of inputs is FSA. The reason is that FSA cannot use the raw data of source of leakage for performing the model test.

Table III shows that DFIA extends the concept of DPA to the fault attacks without causing the cost of profiling. The purpose of the proposed attack is to derive fault intensity information from the cryptographic algorithm and correlate this information to the fault type. Compared to the previously discussed DPA attacks, DFIA does not require a large number of experiments. Compared to DFA attacks, DFIA has lower restrictions on the fault model. It also does not require using the complex mathematical formulas for the key retrieval. Compared to the FSA attack, DFIA has no need to the profiling phase. Also, the source of leakage for FSA is Fault Sensitivity which is the property of the circuit under attack, but the source of leakage for DFIA is injecting biased faults which is under the control of the attacker.

VI. CONCLUSION AND FUTURE WORK

This paper presents a way to perform differential fault analysis based on the correlation of fault propagation probability and the value of faulty ciphertexts. The method requires reasonably more fault injections in comparison to previous works but has lower restrictions on fault injection scheme. We have injected faults in the 10th round of AES algorithm and shown that by inversing the value of the faulty ciphertext byte to an intermediate value, namely the state byte of round 10, the location of the state bytes are close to each other only for the correct key byte. We have tested our methodology on an FPGA implementation of AES algorithm. Our results show that DFIA can retrieve the 128-bit AES key by 7 fault injections.

VII. ACKNOWLEDGMENTS

This research was supported in part by NSF grant no 1115839, and in part in support of our planned contribution to the NSF/SRC STARRS program.

REFERENCES

[1] Paul Kocher, Joshua Jaffe, and Benjamin Jun, "Differential Power Analysis", in *Advances in Cryptology CRYPTO99*. Springer, 1999, pp. 388–397.

[2] Dan Boneh, Richard A De Millo, and Richard J Lipton, "On the Importance of Checking Cryptographic Protocols for Faults", in *Advances in Cryptology EUROCRYPT97*. Springer, 1997, pp. 37–51.

[3] Thomas Fuhr, Eliane Jaulmes, Victor Lomné, and Adrian Thillard, "Fault Attacks on AES with Faulty Ciphertexts Only", in *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC),*. IEEE, 2013, pp. 108–118.

[4] Sylvain Guilley, Laurent Sauvage, J-L Danger, Nidhal Selmane, and Renaud Pacalet, "Silicon-level Solutions to Counteract Passive and Active Attacks", in *5th Workshop on Fault Diagnosis and Tolerance in Cryptography, 2008. FDTC'08*. IEEE, 2008, pp. 3–17.

[5] Yang Li, Kazuo Sakiyama, Shigeto Gomisawa, Toshinori Fukunaga, Junko Takahashi, and Kazuo Ohta, "Fault Sensitivity Analysis", in *Cryptographic Hardware and Embedded Systems, CHES 2010*, pp. 320–334. Springer, 2010.

[6] Gilles Piret and Jean-Jacques Quisquater, "A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD", in *Cryptographic Hardware and Embedded Systems, CHES 2003*, pp. 77–88. Springer, 2003.

[7] Michael Tunstall, Debdeep Mukhopadhyay, and Subidh Ali, "Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault", in *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication*, pp. 224–233. Springer, 2011.

[8] Chong Hee Kim, "Differential Fault Analysis against AES-192 and AES-256 with Minimal Faults", in *2010 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC),*. IEEE, 2010, pp. 3–9.

[9] Oliver Kömmerling and Markus G Kuhn, "Design Principles for Tamper-Resistant Smartcard Processors", in *Proceedings of the USENIX Workshop on Smartcard Technology*. USENIX Association, 1999, pp. 2–2.

[10] Sıddıka Berna Örs, Elisabeth Oswald, and Bart Preneel, "Power-Analysis Attacks on an FPGA–First Experimental Results", in *Cryptographic Hardware and Embedded Systems, CHES 2003*, pp. 35–50. Springer, 2003.

[11] Ludger Hemme, "A Differential Fault Attack against Early Rounds of (triple-) DES", in *Cryptographic Hardware and Embedded Systems-CHES 2004*, pp. 254–267. Springer, 2004.

[12] Eli Biham and Adi Shamir, "Differential Fault Analysis of Secret Key Cryptosystems", in *Advances in Cryptology CRYPTO'97*, pp. 513–525. Springer, 1997.

[13] Chong Hee Kim, "Improved Differential Fault Analysis on AES Key Schedule", *IEEE Transactions on Information Forensics and Security,*, vol. 7, no. 1, pp. 41–50, 2012.

[14] Damien Giot, Philippe Roche, Gilles Gasiot, Jean-Luc Autran, and Reno Harboe-Sorensen, "Heavy ion Testing and 3D Simulations of Multiple Cell Upset in 65nm Standard SRAMs", in *9th European Conference on Radiation and Its Effects on Components and Systems, 2007. RADECS 2007.* IEEE, 2007, pp. 1–6.

[15] Michel Agoyan, Jean-Max Dutertre, David Naccache, Bruno Robisson, and Assia Tria, "When Clocks Fail: On Critical Paths and Clock Faults", in *Smart Card Research and Advanced Application*, pp. 182–193. Springer, 2010.

[16] Alessandro Barenghi, Guido Bertoni, Emanuele Parrinello, and Gerardo Pelosi, "Low Voltage Fault Attacks on the RSA Cryptosystem", in *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC),*. IEEE, 2009, pp. 23–31.

[17] PUB FIPS, "197, Advanced Encryption Standard (AES)", *National Institute of Standards and Technology*, 2001.

[18] Ronan Lashermes, Guillaume Reymond, J Dutertre, Jacques Fournier, Bruno Robisson, and Assia Tria, "A DFA on AES Based on the Entropy of Error Distributions", in *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC),*. IEEE, 2012, pp. 34–43.

[19] Sumio Morioka and Akashi Satoh, "An Optimized S-Box Circuit Architecture for Low Power AES Design", in *Cryptographic Hardware and Embedded Systems-CHES 2002*, pp. 172–186. Springer, 2003.

[20] Joan Boyar and René Peralta, "A New Combinational Logic Minimization Technique with Applications to Cryptology", in *Experimental Algorithms*, pp. 178–189. Springer, 2010.

[21] David Canright, "A Very Compact S-box for AES", in *Cryptographic Hardware and Embedded Systems–CHES 2005*, pp. 441–455. Springer, 2005.

[22] Pierre Dusart, Gilles Letourneux, and Olivier Vivolo, "Differential Fault Analysis on AES", in *Applied Cryptography and Network Security*. Springer, 2003, pp. 293–306.

[23] Christophe Giraud, "DFA on AES", in *Advanced Encryption Standard–AES*, pp. 27–41. Springer, 2005.