Analyzing the Efficiency of Biased-Fault Based Attacks

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. The traditional fault analysis techniques developed over the past decade rely on a fault model, a rigid assumption about the nature of the fault. A practical challenge for all faults attacks is to identify a fault injection method that achieves the presumed fault model. In this paper, we analyze a class of more recently proposed fault analysis techniques, which adopt a biased fault model. Biased fault attacks enable a more flexible fault model, and are therefore easier to adopt to practice. The purpose of our analysis is to evaluate the relative efficiency of several recently proposed biased-fault attacks, including Fault Sensitivity Analysis (FSA), Non-Uniform Error Value Analysis (NUEVA), Non-Uniform Faulty Value Analysis (NUFVA), and Differential Fault Intensity Analysis (DFIA). We compare the relative performance of each technique in a common framework, using a common circuit and using a common fault injection method. We show that, for an identical circuit and an identical fault injection method, the number of faults per attack greatly varies according with the analysis technique. In particular, DFIA is more efficient than FSA, and FSA is more efficient than both NUEVA and NUFVA. In terms of number of fault injections until full key disclosure, for a typical case, FSA uses 8x more faults than DFIA, and NUEVA uses 33x more faults than DFIA. Hence, the post-processing technique selected in a biased-fault attack has a significant impact on the probability of a successful attack.

Keywords: Differential Attack, Fault Intensity, Biased Fault, Fault Intensity

1 Introduction

Secure cryptographic circuits are subject to a wide variety of cryptanalytic techniques, at the level of the algorithm as well as at the level of the implementation. Fault analysis is a class of implementation-oriented attacks. They analyze the response of a circuit to a fault injection, with the objective of accurately estimating an internally stored secret such as a key or a secret variable. Known since over a decade, fault analysis has grown into an advanced and refined cryptanalytic technique that handles public-key - as well as secret-key cryptographic implementations [1, 2]. Faults can be obtained by pushing the circuit outside of

its nominal operating conditions. Some common techniques include overclocking it, voltage-starving it [3], heating it up [4], creating spurious charges using optical means [5], or causing eddy currents through electromagnetic induction. Faults are only limited by the creativity of the adversary.

On the other hand, fault attacks face a recurring challenge. The adversary needs to ensure that the actual physical manifestation of the fault corresponds to the specific assumptions made during fault analysis. Such assumptions include, for example, the location of the fault in the circuit, the precise time at which a fault must occur, and the specific value of a faulty variable. These conditions are summarized in a *fault model*, the set of properties that describe a given fault. Some of the well known fault types assumed by fault attacks are random-byte errors, bit-flip errors, or stuck-at errors. The attacks further assume a specified time precision that can range from a complete encryption period, to one cryptographic round, down to a precise clock cycle.

The efficiency of a fault attack is inversely proportional to the number of fault injections required to learn an internal secret. In well-known fault attacks such as differential fault analysis, a more precise fault model typically requires fewer faults [6]. Hence, a more precise fault model is therefore a desirable objective in fault analysis.

The challenge to the cryptographic engineer is to ensure that the available fault injection techniques for the circuit under consideration, will provide the fault model required for the selected fault analysis. That is a challenge, for various reasons. First, the resolution of fault injection techniques varies greatly. Some fault injection techniques only enable time control, but are imprecise in terms of location. Glitch injection and electromagnetic-pulse injection fall in this category. Some fault injection techniques only have a global effect. Temperature and voltage fall in this category. On the other hand, precise fault injection may be too expensive or too complicated. For example, the use of a laser to trigger setup effects in a selected register, requires partial disassembly of a chip package.

Recently, a series of fault analysis techniques have been introduced that are based on *fault bias*. In this context, fault bias is the proportion of a circuit that experiences a fault under a given fault injection. Attacks based on fault bias can use relaxed fault models, compared to classic fault attacks.

We will discuss and compare several recently proposed biased-fault attacks, including Fault Sensitivity Analysis (FSA, [7]), Non-Uniform Error Value Analysis (NEUVA, [8]), Non-Uniform Faulty Value Analysis (NUFVA, [9]) and Differential Fault Intensity Analsysis (DFIA, [10]). We will show that these four attacks share common ideas, and hence their performance can be compared. Performance, in the context of this paper, is defined as the number of $(faults, plaintext)$ pairs needed to extract (fully or partially) the secret key of a cipher. For the experiments, we have selected *setup time violation* as the source of faults, and a hardware implementation of the Advanced Encryption Standard as the reference design. The faults are injected by controlled clock glitches. Our results are obtained from gate-level simulations, such that we are able to determine the exact

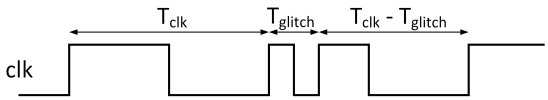


Fig. 1. The effect of the glitch injection on the clock signal

cause of each fault. These are pragmatic choices that make our results verifiable and comparable with other efforts.

The remainder of this paper is organized as follows. In Section 2, we provide a more systematic description of fault bias, and we present a tentative definition of the concept. We show that, when faults are caused through setup time violations, fault bias is a property of the static timing of a circuit. Section 3 shows the effects of fault bias on the circuit behavior. These effects will later be used by the attacks. In Section 4, we describe a generic framework for a systematic comparison of biased-fault attacks, and in Section 5, we apply it to FSA, NUEVA, NUFVA and DFIA. Section 6 describes our experimental setup, and in Section 7, we present the results of biased fault attacks. We determine the efficiency of each fault attack under fault injection with varying precision. In Section 8, we conclude the paper.

2 Causes of Biased Faults

In this section, we first explain the mechanism of clock glitch injection, and then, describe the causes of fault bias.

2.1 Setup Time Violation

We inject faults into the operation of a circuit by violating its setup time constraints. *Setup time violation* is a widely-used low-cost fault injection mechanism [11]. In the following paragraphs, we explain setup time constraints of a circuit and their use as the fault injection means.

In synchronous circuits the data is processed by combinational blocks, which are surrounded by input/output registers. The data is captured when the sampling edge of the clock signal arrives at the registers. Each combinational block requires a certain *propagation delay* (T_{pd}) to compute its output value. For the correct operation of the circuit, combinational block outputs must settle to their final values and remain stable at least some *setup time* (t_{su}) before the sampling clock edge. Therefore, the *clock period* (T_{clk}) must satisfy the following equation for all paths from input registers to output registers:

$$T_{clk} \geq T_{pd} + T_{su} \tag{1}$$

This equation specifies the *setup time constraints* of a circuit. The setup time constraint of the longest (i.e, critical) path determines the minimum clock period

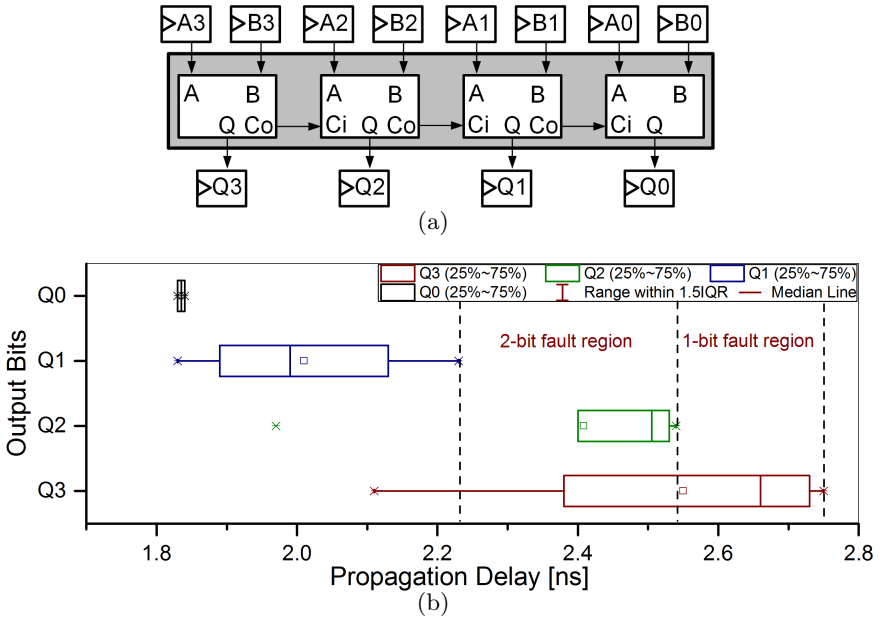


Fig. 2. (a) A block diagram of a 4-bit ripple-carry adder implemented on an FPGA. **(b)** The distribution of path delays within fan-in cone of each output bit. Using the non-uniform path delay distribution, we can create biased faults (e.g, 1-bit, 2-bit, etc.)

for the circuit. Applying a shorter clock period than this value will fail the setup time constraints.

An adversary can cause setup time violation via clock glitches. Figure 1 shows the effect of a glitch on the clock signal. A clock glitch will temporarily shorten the clock cycle period from T_{clk} to T_{glitch}, thereby causing timing violation of the digital logic. When the *glitch period* (T_{glitch}) violates timing constraint of a path, the output value of this path is captured before its computation is completed. Therefore, the captured value is very likely to be faulty.

2.2 Causes of Fault Bias

The fault behavior of a circuit under clock glitch injection is determined by its *path delay distribution* and the applied *fault intensity*. For clock glitching, we define the *fault intensity (FI)* as the the inverse of glitch period, T_{glitch} (Fig. 1), and quantify it with Equation 2. The following paragraphs explain how these two factors can be combined to inject biased faults.

$$FI = \frac{1}{T_{glitch}} \tag{2}$$

Figure 2(a) shows the block diagram of a 4-bit ripple-carry adder, which computes sum (Q) of two input numbers (A, B). Figure 2(b) shows the (static)

path delay distribution of the adder in the form of box-whisker plots. Each box-whisker plot in Figure 2(b) shows the delay distribution of paths within the fan-in cone of a different output bit. We extracted the path delays from a post-place-and-route netlist generated for a Xilinx Spartan 3E FPGA. As it is seen, the path delay distribution is non-uniform. For example, more than 50% of the path delays within the fan-in cone of bit $Q3$ are greater than all of the path delays within the fan-in cone of bit $Q2$. Similarly, the path delays within the fan-in cone of $Q0$ are the smallest ones. This observation promises a biased (i.e, non-uniform) fault behavior. For example, it is possible to inject a fault that affects only a few bits of a targeted variable. This can be achieved by applying a fault intensity that violates only a few paths. If the applied fault intensity is greater than $0.364GHz$ (i.e, $T_{glitch} > 2.75ns$), no faults can be injected into the ripple-carry adder. However, single-bit faults can be injected on the bit Q_3 when the fault intensity is between $0.364GHz$ and $0.392GHz$ (i.e, $2.55ns < T_{glitch} < 2.75ns$). Additional faults can be induced in the other bits only if the fault intensity value is increased further.

As a consequence, the non-uniform path delay distribution enables an adversary to obtain a biased fault behavior in proportion to the fault intensity. The next section provides a definition for the fault bias, which allows us to quantify it.

2.3 Quantifying Fault Bias

The fault bias is a property of the circuit architecture, which expose the potential of a circuit to experience a setup time violation at a given fault intensity. Therefore, we can define the *fault bias (FB)* as the proportion of the violated paths for a given fault intensity. This definition enables us to quantify the fault bias as a number between 0 and 1 via Equation 3.

$$FB(f) = \frac{Number\ of\ violated\ paths}{Number\ of\ all\ paths}\Big|_{FI=f} \tag{3}$$

Figure 3 illustrates this concept for our ripple-carry adder example. It shows the number of violated paths with respect to the fault intensity and the corresponding clock glitch period. The bottom horizontal axis shows the fault intensity values and the top horizontal axis shows the corresponding glitch period values. As it is seen, the number of violated paths increases with the fault intensity. As an example, we will explain two fault bias values for two different fault intensities. The first fault intensity is $0.370GHz$ and the corresponding glitch period is $2.70ns$. The fault bias for this fault intensity value is 0.22. This means that only 22% of the all paths can contribute the fault behavior. As it is shown in the ripple-carry example, this fault intensity can induce only 1-bit faults (Fig. 2(b)). The second fault intensity and the corresponding glitch period are $0.526GHz$ and $1.90ns$, respectively. At this fault intensity, the fault bias is 0.91. In this case, only the bit $Q0$ of the adder computes the correct result. Depending on the attack strategy, both of these cases can be exploited to retrieve the key.

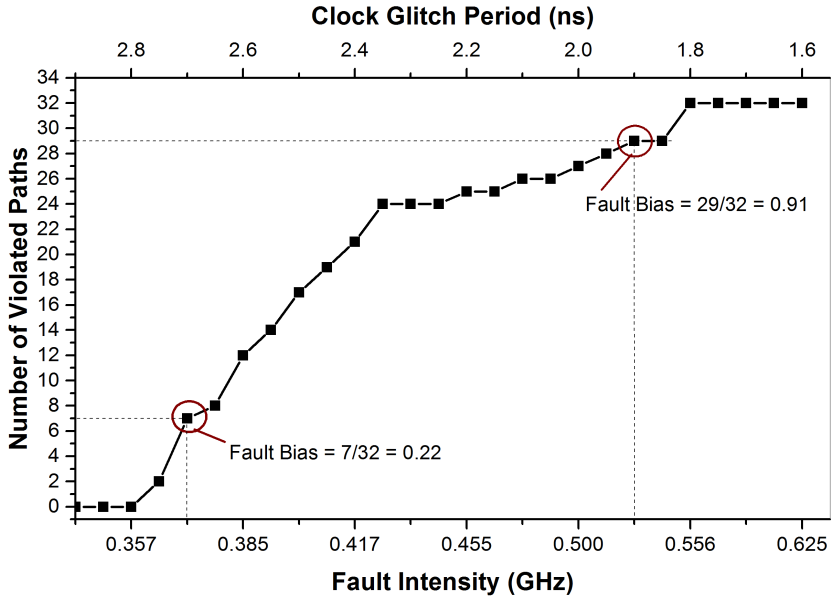


Fig. 3. Number of violated paths with respect to the applied fault intensity and the corresponding glitch period for the ripple-carry adder.

Equation 3 reveals three important properties of fault bias. First, the fault bias is a property of circuit architecture. Therefore, an adversary can accurately model the fault behavior in terms of circuit architecture. Second, an adversary can control the severity of fault effects on a circuit by controlling the fault intensity. Higher fault intensity values induce more severe fault effects. Thus, each point in Figure 3 provides additional information that can be used as a source of leakage. Third, cryptographic hardware designers can evaluate the vulnerability of their designs to setup time time violation, and develop more efficient countermeasures. Next, we will demonstrate other parameters that affect the fault bias.

2.4 Effects of Operating Conditions and Data on Fault Bias

During an attack, the adversary can influence the dynamic path delay distribution through *the input data* and *the operating conditions*. This will affect the fault bias. In the following paragraphs, we will demonstrate their effects on the path delays.

Effects of Operating Conditions: An adversary can influence the path delays of a circuit by varying the operating conditions such as the supply voltage and the operating temperature. In Figure 4, we provide an example to illustrate the effects of varying the supply voltage and the operating temperature on the

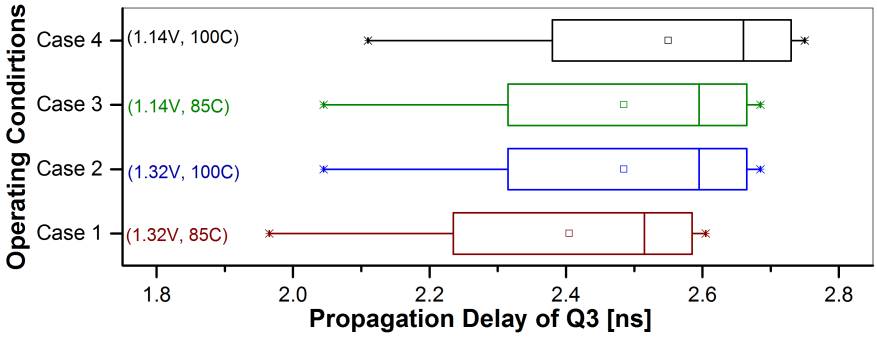


Fig. 4. The effect of varying the supply voltage and the operating temparature on the path delays. Increasing the temperature and decreasing the supply voltage increase the path delays.

path delays of the ripple carry adder. We investigate the path delay distribution within the fan-in cone of bit Q_3 with four different (*supply voltage, operating temperature*) combinations. We provide a separate box-whisker plot for each case. We obtained these data using Xilinx's static timing analyzer tool.

The top two box-whisker plots of Figure 4 show the effect of increasing the operating temperature from $85C$ to $100C$, while applying a constant supply voltage of $1.14V$. The bottom two box-whisker plots show the same effect for the supply voltage of 1.32. As it is seen, the path delays increase with the increasing temperature.

The *Case 1* and *Case 3* shows the effect of decreasing the supply voltage from $1.32V$ to $1.14V$ for a temperature of $85C$. The *Case 2* and *Case 4* illustrate the same effect for a temperature of $100C$. As it is shown, the path delays increases with the decreasing supply voltage. These behaviors are compatible with the experimental results provided by Zussa et al. [12].

Effects of the Processed Data: An adversary can also influence the path delays via processed data since path delays are data-dependent.

In Figure 5, we provide post-place-and-route simulation results for our ripple-carry adder example to illustrate the data-dependency of its path delays. In the simulation, we first initialized the adder ouputs to *logic-1*. Then, we applied two different input sets and observed the timing of the output signals during their transition from *logic-1* to *logic-0*. As it is seen, path delays have a distribution of $T_{Q3} > T_{Q2} > T_{Q1} > T_{Q0}$ for the first input set (Fig. 5(a)). The output bit $Q3$ settles down later than the other bits because of the ripple effect. On the other hand, the path delay distribution is $T_{Q3} > T_{Q1} > T_{Q2} > T_{Q0}$ for the second input set (Fig. 5(b)). As there is no carry propagation in this case, the bit $Q1$ can settle down before the bit $Q2$. This shows that modifying the processed data changes the distribution of the path delays.

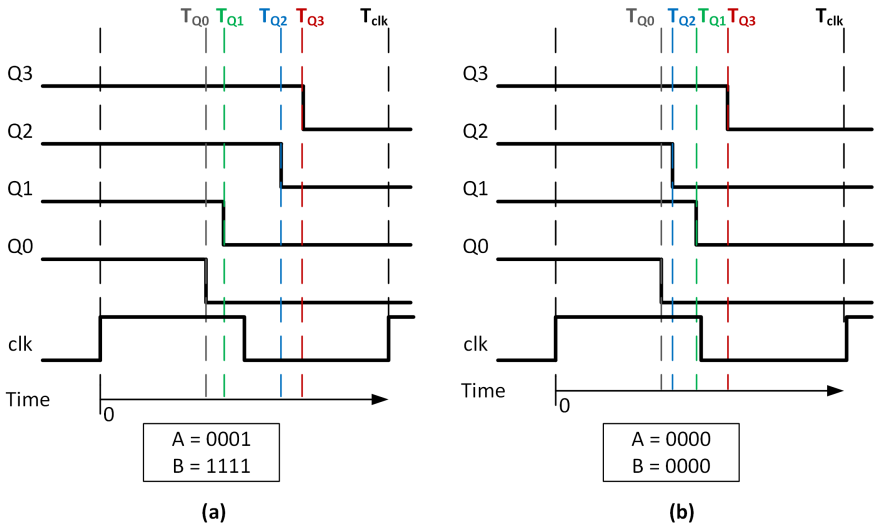


Fig. 5. Illustration of data-dependency of paths delays on the ripple-carry adder: **(a)** $T_{Q3} > T_{Q2} > T_{Q1} > T_{Q0}$. **(b)** $T_{Q3} > T_{Q1} > T_{Q2} > T_{Q0}$.

3 Effects of Fault Bias on Circuit Behavior

In this section, we demonstrate the different effects of the fault bias on the circuit behavior, which are exploited by FSA, NUEVA, NUFVA, and DFIA. For this purpose, we provide some experimental results for two AES SBOX designs, namely, PPRM1-SBOX [13] and Comp-SBOX [14]. PPRM1-SBOX is a low-power SBOX design, which is based on a AND-XOR logic array. Comp-SBOX is a compact composite-field-based design, which decomposes the SBOX operation into smaller, lower-level finite-field operations.

Before proceeding further, we need the following definitions. We can model the effects of the fault injection on the targeted signal with an XOR operation:

$$s^* = s \oplus e \tag{4}$$

In Equation 4, the *correct value*, s, is the value of the targeted signal without fault injection. The *faulty value*, s^*, is the value of the targeted signal after fault injection. The *error value*, e, denotes the value of the injected fault itself. If *i-th* bit of the error value (e) is 1, the *i-th* bit of the correct value (s) is flipped and the faulty value (s^*) is obtained. Next, we will show different effects of fault bias on these values.

Data-Dependency of the Fault Sensitivity: If an adversary gradually increases the fault intensity, a circuit can reach a point at which the output of the circuit becomes faulty. This threshold point is called *fault sensitivity*. For setup

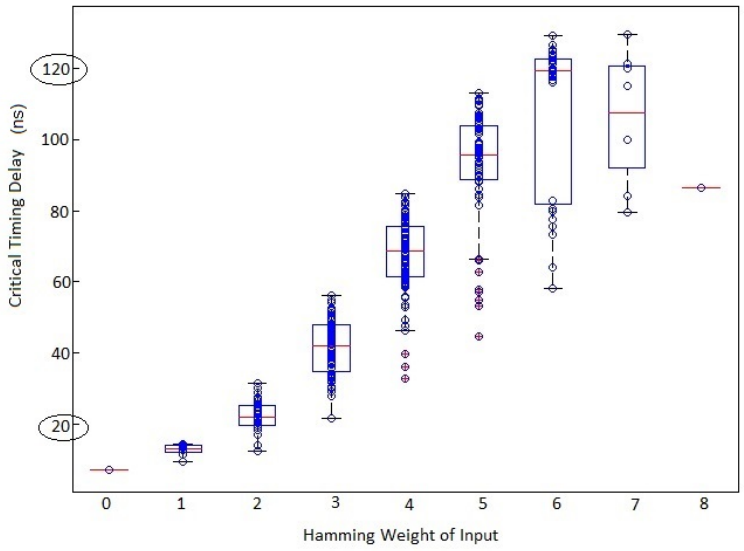


Fig. 6. The relationship between the critical timing delay and the Hamming weight of the input of PPRM1-SBOX [15].

time violation, the critical path delay determines the fault sensitivity point. As the path delay distribution of a circuit is data-dependent, the fault sensitivity of the circuit is also data-dependent.

Ghalaty et al. experimentally demonstrated this effect on an FPGA implementation of PPRM1-SBOX architecture [15]. They obtained critical path delay value for each possible SBOX input in their experiment, where the initial values of SBOX outputs are *logic-0*. Figure 6 shows their critical path delay results with respect to the Hamming weight of the SBOX input values. As it is seen, the critical path delay of PPRM1-SBOX is proportional to the Hamming weight of the SBOX inputs.

Non-uniform Error Value Distribution: Fault bias can also cause a non-uniform distribution in the error value (i.e, fault pattern).

To illustrate this effect, we obtained error value distribution for a PPRM1-SBOX design. For this purpose, we applied a gate-level simulation for a post-place-and-route netlist, which is generated for a Xilinx Spartan6 FPGA. During the simulation, we applied all possible input transitions to the SBOX inputs and observed the error values injected into the SBOX outputs. We used a fault intensity of $0.238GHz$ in our experiment. Figure 7 shows the obtained error value distribution at the output of the SBOX. The horizontal axis shows the error values and the vertical axis shows the their frequency of occurrence. As it

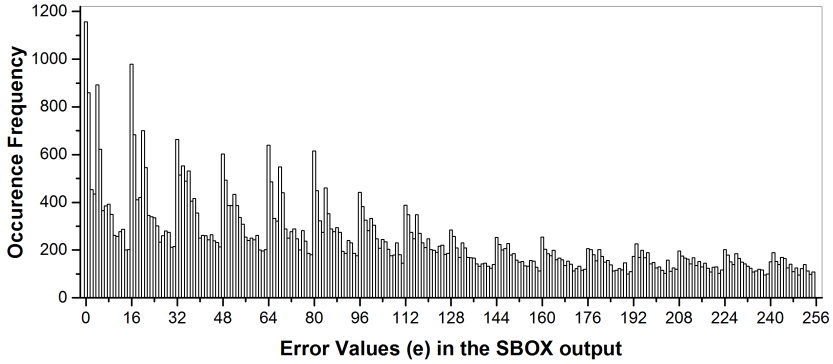


Fig. 7. The distribution of injected error values in the output of PPRM1-SBOX ($Fault\ Intensity = 0.238GHz$).

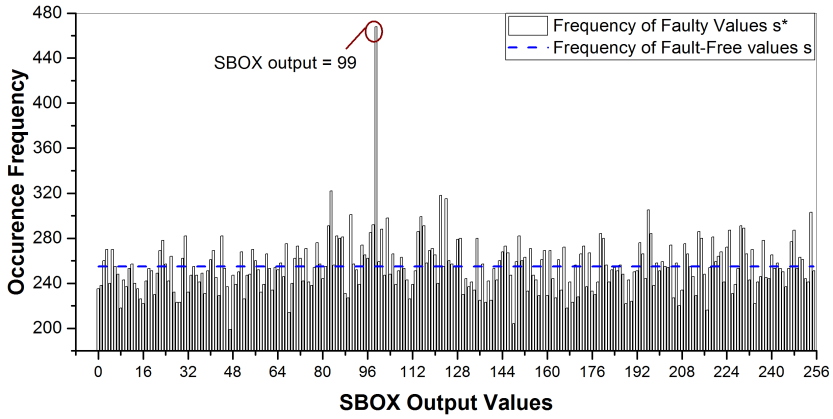


Fig. 8. The distribution of faulty values in the output of Comp-SBOX ($Fault\ Intensity = 0.250GHz$).

is seen, the fault bias causes a non-uniform error value distribution. Also, error values that have low Hamming-weight are more likely to occur for this fault intensity value.

Non-uniform Faulty Value Distribution: Fault bias might also cause a non-uniform distribution in faulty values. By controlling the fault intensity, an adversary can make a circuit produce some faulty values more than the other faulty values.

Figure 8 demonstrates this effect for a gate-level netlist of Comp-SBOX, which is generated for a Xilinx Spartan6 FPGA. To obtain this figure, we applied a gate-level simulation with all possible SBOX input transitions and observed the faulty SBOX ouput values. Our fault intensity was $0.250GHz$ in our sim-

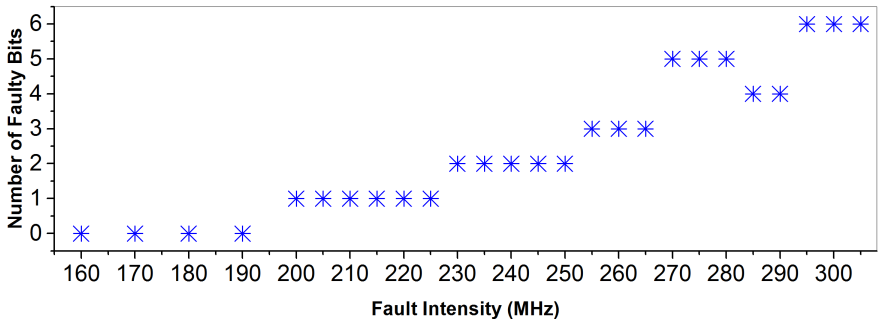


Fig. 9. The relationship between the critical timing delay and the Hamming weight of the input of PPRM1-SBOX [10]

.

ulation. The figure shows the distribution of faulty SBOX output values. The horizontal axis shows the faulty output values and the vertical axis shows the their frequency of occurrence. It also shows the distribution of fault-free SBOX output values. As it is shown (blue line of Fig. 8), the correct outputs of the SBOX have a uniform distribution. However, faulty outputs have a non-uniform distribution, and we see the output value 99 more than the other faulty values because of fault bias. This effect also experimentally demonstrated on an ASIC implementation of Comp-SBOX by Li et al. [16].

Small Changes in Fault Behavior: The effects of fault bias on a circuit's operation has also a differential character: A small change in the fault intensity will cause a small change in the fault bias. Therefore, an adversary can gradually increase the number of induced faults by gradually increasing the fault intensity. This enables the adversary to combine the information obtained at different fault intensities.

Ghalaty et al. experimentally demonstrated this effect on an FPGA implementation of PPRM1-SBOX architecture [10]. In their experiment, they collected the faulty SBOX output values, generated by a certain SBOX input value for 33 different clock frequencies. Figure 9 shows the number of faulty output bits, which they observed in the experiment, with respect to the applied clock frequency (i.e, fault intensity). As it is seen, the number of faulty output bits increases with the increasing fault intensity. As a consequence, this figure experimentally demonstrates the gradual fault behavior in proportion to the fault intensity.

Next section provides a framework to build biased fault attacks, which rely on the effects of fault bias. Then, in Section 6, we will explain how FSA, NUEVA, NUFVA, and DFIA utilize the demonstrated fault bias effects.

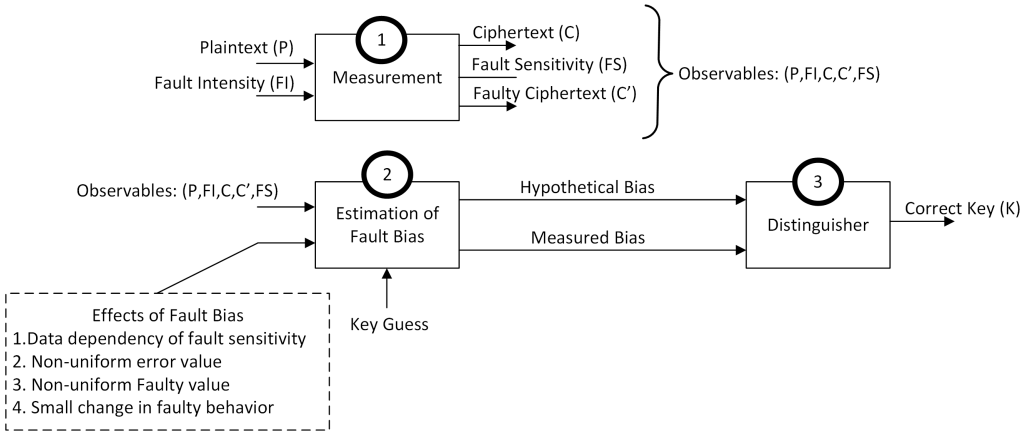


Fig. 10. Generic Fault Attack Framework

4 Building a Biased Fault Attack

In the previous sections, we discussed a systematic description of fault bias, its causes and effects on the circuit behavior. In this section, we describe a generic framework for a systematic comparison of the fault attacks. Following are the steps of a fault attack. These steps are shown in Figure 10 as well.

- **Step 1: Measurement**: In this step, the adversary applies several plaintexts and all possible fault intensities to inject biased fault into the block cipher. Then, he collects the observables, namely, plaintexts, faulty ciphertexts, correct ciphertexts and fault intensities for further analysis. The collected observables contain fault bias information of the circuit.
- **Step 2: Estimating the Effects of Fault Bias on Secret Intermediate Variable**: The target of biased fault injection is a secret intermediate variable. Therefore, an adversary should first invert the observables back to the intermediate variables using key guesses. Then, the adversary estimates the effect of fault bias on hypothesized intermediate variable. Since the effects of fault bias are different, each attack strategy requires a different estimation function on the intermediate variable.
- **Step 3: Distinguisher**: Only for the correct key guess, the estimated fault bias values in Step 2 correspond to the observed fault bias in Step 1. In this step, we distinguish the correct key guess from the wrong key guesses. For this purpose, the distinguisher first assigns a number for the strength of the correlation between the collected observables (Step 1) and the fault bias estimations (Step 2). Then, it selects the key guess which corresponds to the maximum strength.

5 Biased Fault Attacks

In this section, we explain FSA, NUEVA, NUFVA and DFIA on AES algorithm.

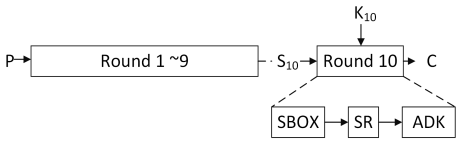


Fig. 11. AES Structure

5.1 Advanced Encryption Standard Algorithm

This section provides some preliminary information about AES algorithm and the notations used throughout this section. The AES algorithm consists of 10 rounds. The first 9 rounds have 4 main operations, SBOX, ShiftRows(SR), MixColumn(MC) and AddRoundKey(ADK). Round 10 omits the MixColumn operation. Figure 11 shows the structure of the AES algorithm. In this figure, P is the applied plaintext to the AES algorithm, S_{10} is the intermediate state variable for round 10. K_{10} is the key for round 10 and C represents the ciphertext. The faulty value of the variable x is shown by $x', x'', ...$

All of the considered attacks aim at retrieving the last round key (K_{10}). However, the target round for fault injection might differ based on the attack strategy. Next, we will give the details of each attack.

5.2 Fault Sensitivity Analysis (FSA)

Fault Sensitivity Analysis (FSA) attack is proposed by Li et. al, in CHES 2010 [7]. Realizing that fault bias can have a data dependency on a secret value in a circuit, Li used it to demonstrate a fault attack with side-channel-like properties. The steps of FSA are as below.

- Step 1: In this attack, the target of the fault injection is round 10. The observables are plaintext, ciphertext and fault sensitivity points.
- Step 2: For FSA, the effect of fault bias is the data dependency of fault sensitivity. The adversary first inverts the ciphertext to round 10 input (S_{10}) using a key guess. Then, he estimates the effect of fault bias as the Hamming Weight of round 10 input $HW(S_{10})$.
- Step 3: In this step the attacker uses the Pearson Correlation Coefficient to find the key guess for which the fault sensitivity is strongly correlated to $HW(S_{10})$ for all inputs.

5.3 Non-Uniform Error Value Analysis (NUEVA)

Lashermes proposed a technique now abbreviated as Non Uniform Error Value Analysis, or NUEVA [8]. NUEVA relies on a biased distribution of error values. Lashermes used NUEVA in a differential evaluation technique. He evaluated the Shannon Entropy in the distribution of a secret error value under a given key hypothesis, and was thus able to distinguish a correct hypothesis from a wrong hypothesis. The steps of the attack are as below.

- Step 1: In this step, the target of fault injection is the output of round 9. The observables are correct and faulty ciphertexts.
- Step 2: Since, the effect of the fault bias in NUEVA is non-uniformity in error value, the adversary must estimate the fault bias on the error value. For this purpose, he inverts the faulty and correct ciphertexts for each key guess, and obtains the input of round 10 (S_{10}, S'_{10}). Then, the error value is estimated by XORing the S_{10} and S'_{10} values.
- Step 3: In this step, using the error values computed in Step 2, the adversary generates the error value distribution for each key guess. The adversary uses the Shanon Entropy to differentiate a key guess that has the strongest bias in the error value distribution.

5.4 Non-Uniform Faulty Value Attack (NUFVA)

Fuhr generalized the NUEVA technique, by directly considering the distribution of the faulty secret variable separately. His technique therefore is called Non Uniform Faulty Value Analysis (NUFVA), to indicate that the bias is present in the fault value itself, rather than in the error pattern [9]. He also proposed several distinguisher techniques. The NUFVA attack can be performed in different rounds of AES including round 7, 8 and 9.

- Step 1: We take the round 9 as the fault injection target. Since, this attack is a faulty ciphertext only attack, the observables include the fault intensity and the faulty ciphertext.
- Step 2: The effect of fault bias for NUFVA is the non-uniform faulty value distribution. To extract the fault bias, the attacker must estimate the effect of fault bias on the faulty value. Therefore, he inverts the faulty ciphertext using a key hypothesis and obtain faulty inputs for round 10 (S'_{10}).
- Step 3: Using the values of faulty intermediate state, the attacker computes the distribution of faulty values for each key guess. Then, for each key guess, the attacker applies the Maximum likelihood function to distinguish the correct key guess from the wrong ones.

5.5 Differential Fault Intensity Analysis (DFIA)

The fourth technique that builds on fault analysis is Differential Fault Intensity Analysis (DFIA), proposed by Ghalaty [10]. The steps and properties of the attack are explained in the following steps.

- Step 1: The target of fault injection for DFIA is the output of round 9. The observables for this attack are the fault intensity and the faulty ciphertexts.
- Step 2: Unlike the previous techniques, DFIA does not assume that the fault distribution or the faulty value is biased. Rather, the fundamental difference with the previous techniques is that DFIA relies on small change in fault behavior as a result of small change in fault intensity. To estimate the small change, the adversary computes the input of round 10 (S'_{10}, S''_{10},...), by

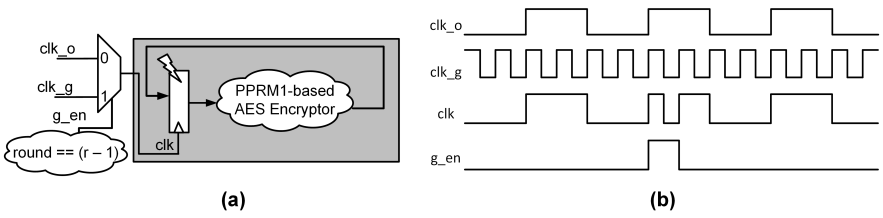


Fig. 12. (a) Block diagram for the experimental setup. **(b)** Timing diagram for the experimental setup.

inverting the faulty ciphertexts and key guess for several fault intensity levels. Then, he computes the distance between the hypothesized intermediate variables by using the Hamming Distance function.

– Step 3: The fault bias assumption for DFIA enables the use of a distinguisher that looks for the smallest change. Unlike the previous techniques, DFIA can combine fault behaviors collected at multiple fault intensities. Hence, the complete fault bias characteristic of a circuit can be exploited. Based on the assumption of fault attack, the error values are close to each other for the correct key guess. For wrong key guesses, the distance between injected error values will be random due to the non-uniform behavior of the Sbox module. Therefore, the distinguisher function simply chooses the key that shows the minimal distance between intermediate variables.

6 Experimental Setup

In this work, we injected biased faults into a device under test (DUT) through gate-level simulation. As the DUT, we use two AES-128 designs: PPRM1-SBOX-based AES-128 (PPRM1-AES) and Comp-SBOX-based AES-128 (Comp-AES). We generated the gate-level netlists of these designs for a Xilinx Spartan6 FPGA ($45nm$ technology). Both DUTs compute each round of AES in a separate clock cycle. We use clock glitches as the fault injection means (Fig. 12(a)). This method generates a 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 12(b), we inject glitches in the clk_o via an *enable signal* (*g_en*). To inject a biased fault in the input of the *r-th round* of AES, we set the *g_en* signal just before the clock cycle, in which the *r-th round* is computed. Such a glitch injection makes some timing paths fail during $(r-1)$*-th round* and causes a biased fault in the input of *r-th round*. We control the fault intensity by increasing/decreasing the period of the *clk_g* signal.

As each considered attack has different requirements for the fault injection, we collected a large set of fault injections results to compare their performances. We repeated the following steps in our gate-level simulations for each DUT. We first generated 1000 random plaintexts. Then, for the *rounds 6-10* of AES, we obtained the ciphertexts for different clock glitch periods. In this experiment,

we gradually decreased the clock glitch period from $16ns$ to $0.6ns$ with $100ps$ step size. At the end, we obtained 154 ciphertexts for each plaintext and 154000 ciphertexts for each round. In Section 8, we will use these ciphertexts to evaluate the performances of the considered biased fault attacks.

7 Results

In this section, we show our results for two fault injection conditions. The first case is the ideal condition, in which the target of the fault injection is a specific round of the AES algorithm.

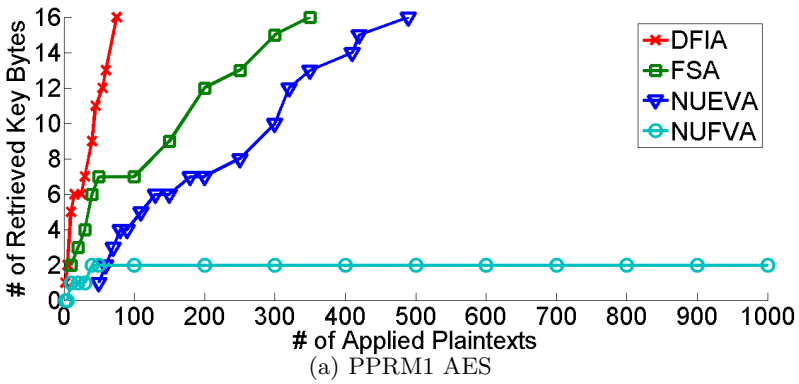
In the second condition, we assume that the fault injection is in a noisy environment or the adversary is not able to control the timing of the glitch injection precisely [17]. Some previous works show that in case of using other injection tools such as Electromagnetic pulse injection, the attacker might not be able to specifically inject fault into one round [17]. In this case, we assume that the faults might occur in other rounds of the AES algorithm. In this case, we randomly choose the faulty results from several rounds of AES and study the effect of noise in the performance of the attacks.

7.1 Results for Ideal Fault Injection

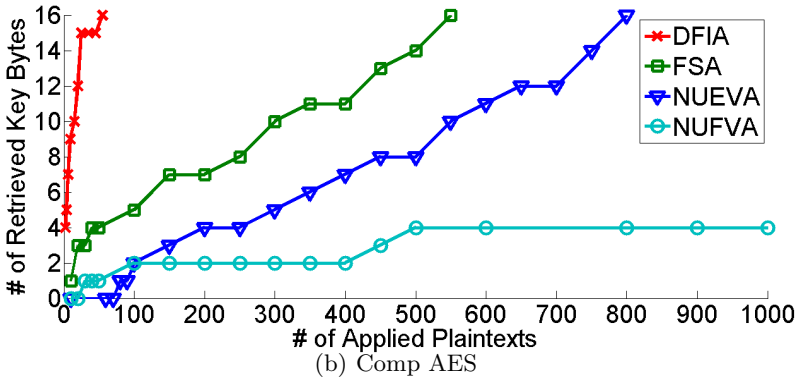
In the ideal condition, we assume that the fault injection tool is based on the clock glitching and the attacker is able to identify the location of the fault in the AES algorithm. Based on the requirements of the discussed attacks, we inject faults in Round 9 for DFIA, NUEVA and NUFVA, and in round 10 for the FSA algorithm, in order to retrieve the key of the last round. Figure 13 shows the results of applying different attack strategies on two implementations of AES algorithm (PPRM1-AES and Comp-AES).

The first fault injection strategy is by starting from the correct ciphertext. Then, we gradually increase the fault intensity until we observe the first faulty ciphertext at the fault sensitivity point. We captured the faulty ciphertext with this method for 1000 plaintext. Then, we applied four attack strategies to the set of faulty ciphertexts. The results in Figure 13(a) and 13(b) show the required number of plaintext for retrieving the key. In this case, the required number of plaintexts simply shows the number of fault injection attempts, since there s one fault sensitivity point for each plaintext. As shown, in this case, even if we do not have multiple fault intensities, the DFIA attack works with less number of plaintexts compared to FSA and NUEVA. The NUFVA attack is not able to retrieve all bytes of the key as observing stuck-at or biased faulty value with this method of fault injection is very difficult.

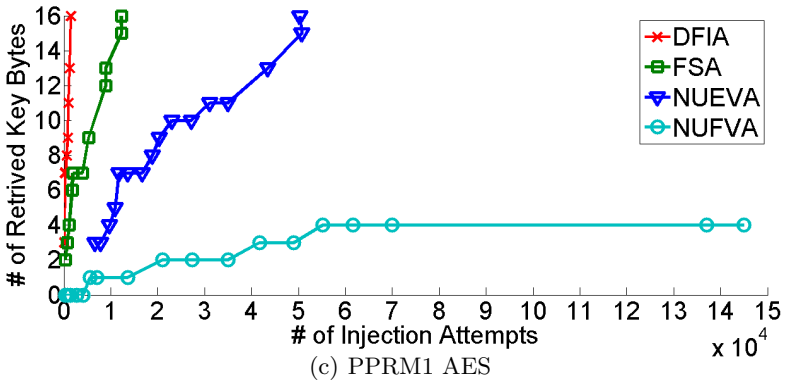
The second fault injection methodology is the extension of the first one. Starting from the correct ciphertext, we gradually increase the fault intensity for each plaintext and keep different faulty ciphertexts for each plaintext. We injected 154 levels of fault intensity for 1000 plaintext. Then, we applied four different attacks on these faulty ciphertexts. Each attack uses a certain amount



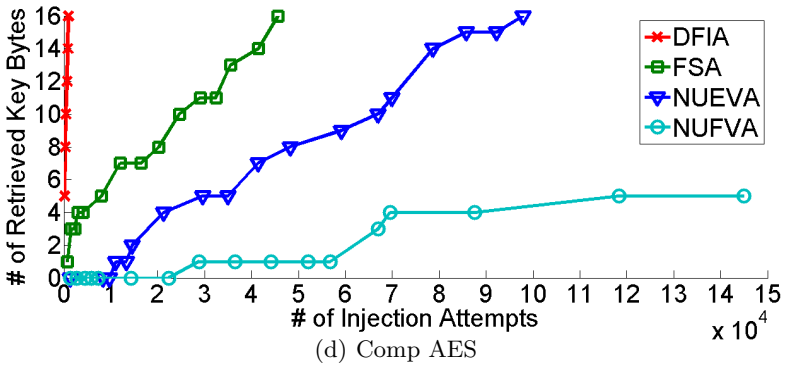
(a) PPRM1 AES



(b) Comp AES



(c) PPRM1 AES



(d) Comp AES

Fig. 13. Number of Required Fault Injection Attempts to Retrieve the Key with Different Attack Strategies in Ideal Condition (a)(b)with Fault Sensitivity Information (c)(d)with All Possible Fault Intensities

of the injected faults based on its requirements. For example, in FSA attack, the attacker only requires the fault injections up to the fault sensitivity point. Or for DFIA, the adversary requires increasing the fault intensity up to the point of generating the last faulty byte. For each attack, we count the number of useful fault intensity levels associated with each plaintext to find the total number of fault injection attempts.

Figure 13(c) and 13(d) show the results for the second fault injection methodology. The results show that the DFIA attack can retrieve the key efficiently with using less than 2000 fault injection attempts. The number of fault injection attempts used by the FSA in Comp-AES increases because observing the effect of fault bias is more difficult in Comp-SBOX implementation. NUFVA attack can only retrieve less than 5 bytes of the key. The reason is that we cannot observe biased faulty output or stuck-at faults in the intermediate state.

7.2 Results for Noisy Fault Injection

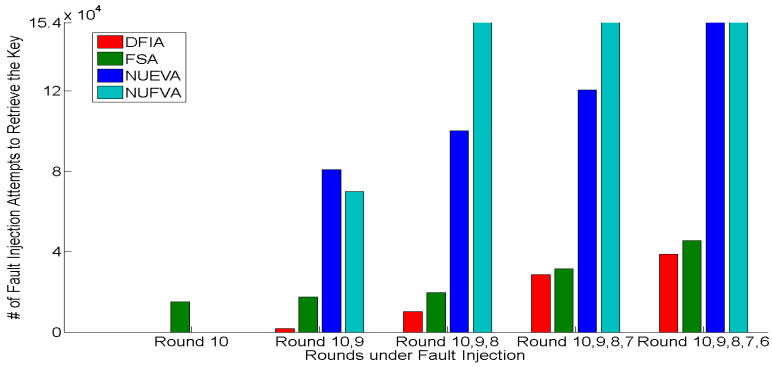
Assuming a noisy fault injection environment, we injected faults into round 6, 7, 8, 9 and 10. Then, as the set of faulty values, we choose faults from each round randomly. Then, we apply the four fault attacks, and count the number of fault injection attempts it requires to find the key. Figure 14(a) and 14(b) shows the number of required fault injections for each case. Since NUFVA attack cannot fully retrieve the key, the number of fault injection attempts shown is for the maximum number of key bytes that it can retrieve. As shown, DFIA is able to retrieve the key by less number of attempts compared to other attacks. The number of fault injection attempts increases exponentially for FSA attack. The reason is that due to the noise in the induced fault, the fault sensitivity point associated with each plaintext is for the data of the noisy rounds, rather than round 10. The NUEVA attack, is still successful when the noise is in round 8, however, in the last case, NUEVA can only retrieve 9 bytes of the key using all available plaintexts.

Since, NUFVA is not able to fully retrieve the key, to provide a fair comparison of the attacks, Figure 14(d) and 14(d) show the number of required fault injection attempts to retrieve only one byte of the key. As shown in these figures, the required number of attempts for DFIA is much less compared to other attacks.

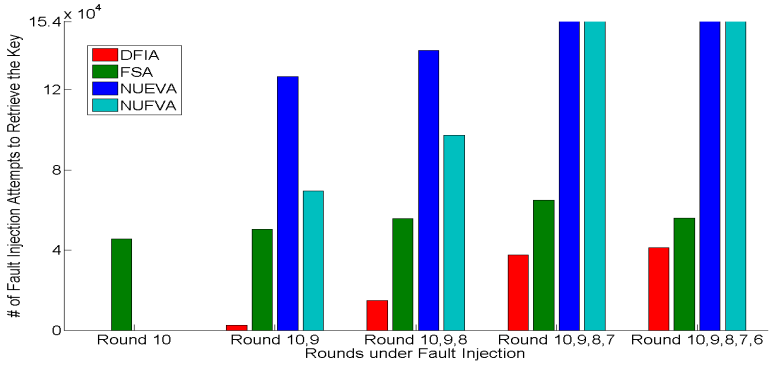
7.3 Attack Efficiency

In this section, we intend to compare the efficiency of the biased fault based attacks. The cost of an attack can be defined by the number of fault injections and the number of applied plaintexts used for the attack. As mentioned in previous sections, for each applied plaintext, we count the number of useful fault injection attempts. The attack efficiency in this paper is defined with the Equation 5.

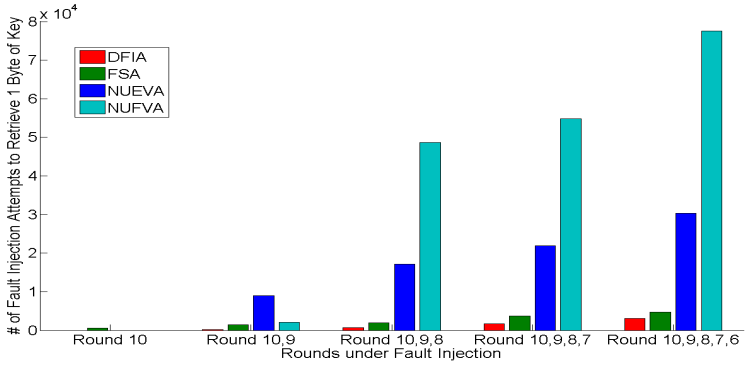
$$Attack\ Efficiency = (\#Plaintexts \times \#Fault\ Injection\ Attempts\ per\ Plaintext)^{-1} \tag{5}$$



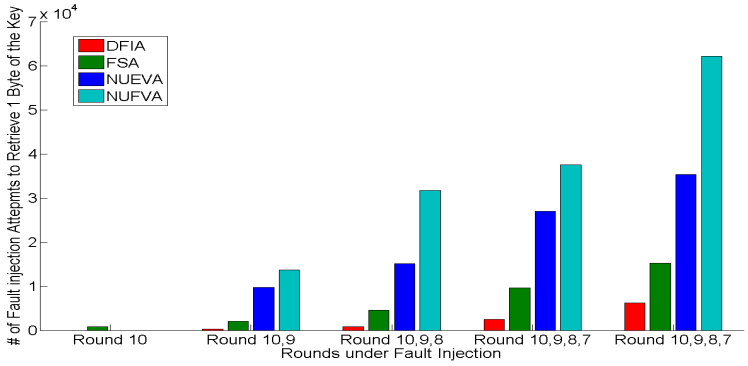
(a) PPRM1 AES



(b) Comp AES



(c) PPRM1 AES



(d) Comp AES

Fig. 14. Number of Required Fault Injection Attempts to Retrieve (a)(b) the Key (c)(d) one Byte of the Key with Different Attack Strategies in Noisy Environment. Each group shows the rounds that are affected by the fault injection.

Table 1. Cost of Fault Attacks for Noisy and Ideal Fault Injection Conditions

	DFIA	FSA	NUEVA	NUFVA
Ideal-PPRM1	1518^{-1}	12250^{-1}	50320^{-1}	69000^{-1}
Noisy-PPRM1	10346^{-1}	19800^{-1}	100100^{-1}	154000^{-1}
Ideal-Comp	980^{-1}	45650^{-1}	91650^{-1}	112800^{-1}
Noisy-Comp	14800^{-1}	55610^{-1}	139650^{-1}	154000^{-1}

To compare the attacks, we counted the number of fault injections and number of applied plaintext for each attack in two conditions. First is the ideal condition that we injected fault only in the target location and second is the noisy condition in which we injected the fault in rounds 8, 9 and 10. Table 1 shows the efficiency of different fault attacks, in these two conditions for two implementations of AES. Since, NUFVA cannot completely retrieve the key, we provided the minimum number of fault injection attempts that it uses for partial key retrieval. Based on the results, DFIA needs less number of fault injection attempts compared to other attacks. The reason is that, the assumption on the effects of fault bias for the DFIA attack benefits from combining multiple fault intensity levels per plaintext. This property maximizes the information we can obtain from each biased fault injection and hence, helps to improve the efficiency of the DFIA attack.

8 Conclusion

In this paper, we presented a comparison of four recently published attacks that use fault bias, the non-uniform response of digital systems towards fault injection. By investigating the common elements of these attacks,we were able to build a single framework that supports a systematic comparison. Our main conclusion is that, even though all of the investigated attacks use the same test case of glitch injection on an AES design, their performance differs greatly. Using the number of fault injections as the cost metric, we found that DFIA performs best, followed by FSA, NUEVA and NUFVA. The main reason for the better performance of DFIA is the differential nature of the analysis mechanism. This makes DFIA more tolerant against estimation mistakes and noise effects. Also, DFIA can use the entire fault characteristic for a given input stimulus, in contrast to other fault injection techniques, which uses only a single fault per input stimulus. Overall, fault-bias based techniques are effective as an implementation attack, and they show that fault-injection based attacks can be applied in a generic setting with minimal assumptions on the underlying cryptographic implementation. It seems reasonable to conclude that there is a rapidly increasing need to develop countermeasures against fault bias in digital hardware, including firmware-driven embedded systems.

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

References

1. Joye, M., Tunstall, M., eds.: Fault Analysis in Cryptography. Information Security and Cryptography. Springer Berlin Heidelberg (2012)
2. Karaklajic, D., Schmidt, J.M., Verbauwhede, I.: Hardware Designer's Guide to Fault Attacks. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on **21** (2013) 2295–2306
3. Barenghi, A., Bertoni, G., Parrinello, E., Pelosi, G.: Low Voltage Fault Attacks on the RSA Cryptosystem. In: IEEE Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC),, IEEE (2009) 23–31
4. Hutter, M., Schmidt, J.: The Temperature Side Channel and Heating Fault Attacks. In: Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers. (2013) 219–235
5. van Woudenberg, J., Witteman, M., Menarini, F.: Practical Optical Fault Injection on Secure Microcontrollers. In: IEEE Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC),. (2011) 91–99
6. Clavier, C.: Attacking block ciphers. In Joye, M., Tunstall, M., eds.: Fault Analysis in Cryptography. Information Security and Cryptography. Springer Berlin Heidelberg (2012) 19–35
7. Li, Y., Sakiyama, K., Gomisawa, S., Fukunaga, T., Takahashi, J., Ohta, K.: Fault Sensitivity Analysis. In: Cryptographic Hardware and Embedded Systems, CHES 2010. Springer (2010) 320–334
8. Lashermes, R., Reymond, G., Dutertre, J., Fournier, J., Robisson, B., Tria, A.: A DFA on AES Based on the Entropy of Error Distributions. In: IEEE Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC),, IEEE (2012) 34–43
9. Fuhr, T., Jaulmes, E., Lomné, V., Thillard, A.: Fault Attacks on AES with Faulty Ciphertexts Only. In: IEEE Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC),, IEEE (2013) 108–118
10. Ghalaty, N.F., Yuce, B., Taha, M., Schaumont, P.: Differential fault intensity analysis. In: IEEE Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC),, IEEE (2014) 49–58
11. Agoyan, M., Dutertre, J.M., Naccache, D., Robisson, B., Tria, A.: When Clocks Fail: On Critical Paths and Clock Faults. In: Smart Card Research and Advanced Application. Springer (2010) 182–193
12. Zussa, L., Dutertre, J.m., Clédiere, J., Robisson, B., Tria, A.: Investigation of Timing Constraints Violation as a Fault Injection Means. In: 27th Conference on Design of Circuits and Integrated Systems (DCIS). (2012)
13. Morioka, S., Satoh, A.: An Optimized S-Box Circuit Architecture for Low Power AES Design. In: Cryptographic Hardware and Embedded Systems-CHES 2002. Springer (2003) 172–186
14. Satoh, A., Morioka, S., Takano, K., Munetoh, S.: A compact Rijndael Hardware Architecture with S-box Optimization. In: Advances in CryptologyASIACRYPT 2001. Springer (2001) 239–254

15. Ghalaty, N.F., Aysu, A., Schaumont, P.: Analyzing and Eliminating the Causes of Fault Sensitivity Analysis. In: Proceedings of the conference on Design, Automation & Test in Europe, European Design and Automation Association (2014) 204
16. Li, Y., Hayashi, Y.i., Matsubara, A., Homma, N., Aoki, T., Ohta, K., Sakiyama, K.: Yet Another Fault-Based Leakage in Non-uniform Faulty Ciphertexts. In: Foundations and Practice of Security. Springer (2014) 272–287
17. Takahashi, J., Hayashi, Y.i., Homma, N., Fuji, H., Aoki, T.: Feasibility of Fault Analysis based on Intentional Electromagnetic Interference. In: Electromagnetic Compatibility (EMC), 2012 IEEE International Symposium on, IEEE (2012) 782–787