TVVF: Estimating the Vulnerability of Hardware Cryptosystems against Timing Violation Attacks

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

Abstract—Secure hardware designers require a method to evaluate the vulnerability of their systems against fault attacks to make proper security/cost trade-offs. To our knowledge, no systematic approach has been proposed for this purpose.

This paper introduces Timing Violation Vulnerability Factor (TVVF), which evaluates the vulnerability of a hardware structure to setup time violation attacks. TVVF, a probabilistic metric computed on a circuit’s netlist, is comprised of two factors: First, the probability of injecting a specific fault in the hardware structure, and second, the probability of propagating this fault to the output of the structure. TVVF aims at evaluating the security of designs against intentional faults caused by an adversary. In contrast, existing vulnerability metrics such as the Architecture Vulnerability Factor (AVF), evaluate the system reliability against random uncontrolled faults. To show the applicability of our metric, we compute the TVVF for two fault attacks on two AES netlists, which are generated for an FPGA.

Index Terms—Vulnerability Factor, Timing Violation Fault Attacks, Secure Hardware Design, Cryptography.

I. INTRODUCTION

Fault attacks pose a serious threat to the security of cryptosystems [1]. A design metric to assist cryptographic hardware designers in the evaluation of the fault attack threat is vital because of the huge pool of fault attacks and possible countermeasures. However, the bulk of the research on fault attacks makes abstraction of the implementation, and concentrates instead on the analysis of fault effects in cryptographic algorithms [2]. Furthermore, the research efforts that consider actual fault attack implementations typically ignore their generalization as a design method or metric [3]. In this paper, we investigate the feasibility of a metric that can be computed from a circuit’s netlist and express the circuit’s vulnerability to fault attacks.

Although each fault attack has different strategies and requirements, all of them share three requirements (Fig. 1). First, an attacker needs to inject an exploitable fault that results in the fault model (i.e., expected fault behavior of the attacked device). For this purpose, the attacker may use a variety of fault injection techniques such as overclocking and voltage-starving. Second, the attacker needs to propagate the effect of the injected fault to the output of the device. Third, the attacker needs to analyze the observed faults to find the secret key. An attacker can use several fault analysis methods (i.e., fault attack strategy) to exploit an observed fault such as Differential Fault Analysis [4], Fault Sensitivity Analysis [5], and Differential Fault Intensity analysis [6]. A successful attack is the one that meets these requirements.

To our knowledge, this is the first work to propose a metric for measuring the vulnerability of an architecture to fault attacks, the so-called Timing Violation Vulnerability Factor (TVVF). Given a hardware structure and a fault attack strategy, the TVVF quantifies the vulnerability of a circuit as a number between 0 and 1, representing non-vulnerable and completely vulnerable, respectively. TVVF estimates the feasibility of an attack in practice. Thus, it captures two crucial requirements of a fault attack.

- How easy is it to inject an exploitable fault?
- How easy is it to propagate the effect of the exploitable faults to the output?

The likelihood of injecting an exploitable fault depends on the fault injection mechanism and fault model. This work focuses on the fault mechanisms that are based on the violation of setup timing constraints of a device. Therefore, our method relies on Static Timing Analysis (STA) to estimate the likelihood of injecting an exploitable fault.

Timing violations can be obtained through different fault injection methods, including overclocking, voltage-starving, and heating of a device [7]. We note that there are also fault injection mechanisms that generate single-event upsets, such as when using lasers or EM pulses. For these mechanisms, other methods than STA can be used for the likelihood of injecting an exploitable fault.
that an exploitable fault affects the device output. We use the observability analysis as the proof-of-concept method because of its simplicity. More advanced methods (e.g., formal verification) can be used for more accurate results.

The proposed methodology for computing TVVF does not require test vectors and can be a part of the design phase. The only assumption in this method is that the evaluator is in possession of the netlist of the hardware structure. TVVF concentrates on timing faults. However, our method for computing TVVF can be applied to any fault injection mechanism by using an appropriate method for likelihood of exploitable fault injection for this specific fault mechanism.

The paper is organized as follows: Section II reviews the preliminaries. Section III explains the methodology to compute the TVVF. In Section IV, we select AES as a case study to evaluate our method. Section V presents related work and Section VI concludes the paper.

II. PRELIMINARIES

A fault attack strategy relies on three elements: (i) A known cryptographic algorithm, (ii) a fault injection tool and fault model, and (iii) a strategy for post processing the fault injection results and finding the key.

The fault model is arguably the most critical element in a fault attack and is comprised of three factors:

- **Fault Location**: It defines the hardware block into which the fault must be injected.
- **Fault Timing**: It defines the time instant at which the fault is injected into the specified fault location.
- **Fault Type**: It describes how a fault manifests itself at the fault location. Fault types can be stuck-at faults, set-reset faults, random bit flip, biased faults, etc [8].

As an example, for the Tunstall attack [2], the fault timing is round $i$ of AES, the fault location is the input of SubBytes, and the fault type is a random byte.

III. TIMING VIOLATION VULNERABILITY FACTOR

Timing Violation Vulnerability Factor (TVVF) of a hardware structure can be defined as the probability of injecting an exploitable fault and observing its effect in the output. We provide a method to compute TVVF of hardware structures for a specific group of fault attacks, which use violation of timing constraints as the fault injection mechanism. Throughout the paper, we use overclocking (i.e., clock glitching) as the fault injection means for simplicity.

To define the concepts in a more convenient way, we use a 2-bit Ripple Carry Adder (RCA) as an example hardware structure (Fig. 2). In this example, our hypothetical attack strategy is to inject a random fault into the output of Full Adder 1 ($C_1$ and $S_1$) and observe the fault in the output of the RCA ($C_2$ or $S_2$ or $S_1$).

The first requirement of a fault attack is injecting an exploitable fault. In a hardware design, there are multiple components that may contribute to a given fault. In a Setup Timing Violation attack on a register, for example, the entire logic cone at the data input can potentially affect the Setup Timing. We use the term **Scope of Vulnerability (SoV)** to describe the collection of hardware elements that contribute to an exploitable fault. Referring to the example in Fig. 2, the SoV for faults on $C_1$ or $S_1$ is the first full adder ($FA_1$).

The second requirement is the ability to observe the exploitable fault. We use the term **Scope of Propagation (SoP)** to describe the part of a netlist that contributes to the propagation of such an exploitable fault. Referring to the RCA example, the SoP for faults on $C_1$ or $S_1$ is the second full adder, and we observe the effect of these faults through $C_2$, $S_2$, or $S_1$. The SoP and SoV concepts apply equally well to cryptographic algorithms. For example, a given fault attack on AES requires the adversary to inject a fault into a state byte in the $j$th round [9]. The SoV for this attack is SubBytes of $9$th round and the SoP is the path from input of ShiftRows at the $9$th round to the output of $10$th round.

The concept of TVVF can be used for (i) comparing the vulnerability of two hardware implementations to a given fault attack and (ii) comparing the feasibility of two fault attacks on a specific hardware implementation. Therefore, each attack strategy $A$ on an implementation $C$ has a value for $TVVF(C, A)$. Each $TVVF(C, A)$ has an SoV and SoP component. In the next three subsections, we present a method to compute TVVF. The method includes analysis of SoV, analysis of SoP, and combining them to TVVF.

A. Timing characterization of the SoV

A timing violation attack is based on making a circuit fail its timing constraints. The attacker violates the timing constraints by controlling the clock period. For example, the injection of a glitch shortens the clock period. A fault manifests itself as a bit-flip on a violated path [3]. Hence, for a given clock period, the occurring fault type is the number of bit-flips at the SoV output.

We want to evaluate the likelihood of a bit-flip fault for a given clock period. We do this by analyzing the timing of each path in the circuit, and evaluating how many paths violate a given clock period constraint. We assume that the likelihood of a bit-flip fault is proportional to the number of violated paths. To characterize the circuit, we repeat this
analysis for different clock period values. For each clock period, we count the number of violated paths. This count is used in the computation of bit-flip fault probability for each clock period, which is described in Algorithm 1. Table I shows the notations that we use throughout this paper.

In our RCA example, the attacker aims to inject a random fault into FA1. Therefore, we perform a timing characterization for FA1 using Algorithm 1:

- **Step 1:** We assume \( T_{min} = 0.9ns \) and \( T_{step} = 0.1ns \), as it can be seen in the first column of Table II.
- **Step 2:** Using STA, we determine the critical path delay \( T_{max} \) of the RCA as 1.153ns. Therefore, we need to apply three different clock periods \( G_{max} = \lceil (1.153 - 0.9)/0.1 \rceil = 3 \) to characterize the SoV (i.e., FA1).
- **Step 3:** After analysis of the STA results, we obtained the path delays for each RCA output. The \( T_{k1} \) and \( T_{k2} \) columns of Table II show these delays for \( C1 \) and \( S1 \), respectively.
- **Step 4:** We discard two paths annotated with \( \times \) in Table II. The delay values of these paths are smaller than \( T_{min} \). Thus, the attacker is not able to violate these paths.
- **Step 5:** The columns \( N_{i1} \) and \( N_{i2} \) of Table II show the number of violated paths for each clock period. We compute the probability values \( P_{G1,O1} \) and \( P_{G1,O2} \) by using the values of \( N_{i1} \) and \( N_{i2} \), respectively. The computed values are shown in Table II.

### B. Propagation Probability Analysis of the SoP

The probability of propagating an exploitable fault from output of the SoV to the output of the cryptosystem is determined by the circuit topology and processed data. Thus, any method that considers these factors can be used to compute this probability.

In this work, we use a probability-based observability analysis method for the probability of propagating an exploitable fault to the output. Observability analysis, which is widely used in VLSI test area, reflects the difficulty of propagating the value of a signal to primary outputs [10]. We selected this method as the proof-of-concept method due to its simplicity. More advanced methods can be used for better results.

In our method, we compute three probability values for each signal \( s \) of the SoP: \( P_{S0}(s) \), \( P_{S1}(s) \), and \( P_{obs}(s) \). \( P_{S0}(s) \) and \( P_{S1}(s) \) are the probability of setting signal \( s \) to 0 and 1 by a random input vector, respectively. \( P_{obs}(s) \) is the probability of observing the value of signal \( s \) at primary outputs (i.e., observability of \( s \)). We first compute \( P_{S0}(s) \) and \( P_{S1}(s) \) for each signal of the SoP in a breadth-first manner from inputs of the SoP (i.e., outputs of SoV) to the outputs of the SoP. After computing all \( P_{S0}(s) \) and \( P_{S1}(s) \) values, we compute the observability of each signal in a breadth-first manner from outputs of the SoP to inputs of the SoP. In these computations, \( P_{S0}(s) \) and \( P_{S1}(s) \) values of the SoP inputs are assigned to 0.5, and \( P_{obs}(s) \) values of the SoP outputs are assigned to 1 as boundary conditions.

In Table III, we provide computation rules for AND, OR, and XOR gates as an example. By applying the rules in Table III, and starting from the given boundary condition, we can compute \( P_{S0}(s) \), \( P_{S1}(s) \), and \( P_{obs}(s) \) values for any circuit consisting of these gates. For instance, we show \( P_{S0}(s) \), \( P_{S1}(s) \), and \( P_{obs}(s) \) values of a full adder in Fig. 3. The three-value tuple \( v1v2v3 \) on each signal line corresponds to \( P_{S0}(s) \), \( P_{S1}(s) \), and \( P_{obs}(s) \) values of this line, respectively.

For our RCA example, the SoP is the second full adder (FA2). SoP outputs are \( S1 \) and \( C1 \). \( P_{obs}(S1) \) is 1 because it is a primary output of the RCA. \( P_{obs}(C1) \) is 0.5.

### C. How do we compute TVVF for an attack on a circuit?

TVVF \((C,A)\) studies the probability of bit-flip faults in the outputs of a circuit \( C \) for a given attack model \( A \). \( P^n \) is...
three factors. In other words, there are values required by the fault model. 

Algorithm 2 considers the contribution of each possible n-combination to compute \( P^n \).

In our RCA example, we assume that the fault model requires random bit-flips at the output of FA1. To compute TVVF, we need to find \( P^1 \) and \( P^2 \) by using Algorithm 2. The results for \( P^1 \), \( P^2 \), \( P^3 \), and \( P^{10} \) are shown in the last two columns of Table II. In the following paragraph, we show the computation of \( P^1 \), \( P^2 \) and \( P^3 \) using Algorithm 2.

For computing \( P^1 \), we have two combinations. We can either inject fault in \( O_1 \) and observe it in the output, or inject fault in \( O_2 \) and observe it (Equation 1). To get \( P^1 \) we use Equation 2.

![Fig. 3: Observability Analysis for a Full Adder Circuit](image-url)

The probability of injecting n-bit-flips in the output of SoV and observing its effect in the output of SoP. Based on the required fault type for an attack, TVVF\((C, A)\) can be an appropriate \( P^n \) value or a combination of multiple \( P^n \) values. For example, we should select \( P^1 \) as the TVVF\((C, A)\) if our fault attack requires 1-bit fault. Similarly, we should select \( P^1 + P^2 + ... + P^8 \) if our fault attack requires a random byte fault. To compute TVVF, we need to compute all of the \( P^n \) values required by the fault model.

To observe an n-bit-flip in the output of SoV, we need \( n \) erroneous output bits. In other words, there are \( (O_{max}) \) possible n-combinations of output bits that might yield an n-bit-flip. Here, \( O_{max} \) is the total number of SoV output bits. The contribution of each n-combination to the \( P^n \) depends on three factors.

- **The probability of injecting a clock period \( G_i \):** Since the attacker can inject at most \( G_{max} \) clock periods with the equal step size, this probability value is \( \frac{1}{G_{max}} \).

- **The probability of obtaining a bit-flip in each output \( O_j \) of SoV for a clock period \( G_i \):** Section III.A explains the method to compute \( P_{G_i, O_j} \).

- **The probability of observing an exploitable fault in the output of the circuit:** Section III.B explains the method to compute \( P_{obs}(O_j) \).

Algorithm 2 considers the contribution of each possible n-combination to compute \( P^n \).

For computing \( P^1 \), we have two combinations. We can either inject fault in \( O_1 \) and observe it in the output, or inject fault in \( O_2 \) and observe it (Equation 1). To get \( P^1 \) we use Equation 2.

![Table III: Observability Computation Rules for Different Gates](table-url)

<table>
<thead>
<tr>
<th>Gate Type</th>
<th>( P_{10}(out) )</th>
<th>( P_{11}(out) )</th>
<th>( P_{obs}(in_j) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>1 - ( P_{11}(out) )</td>
<td>( \prod_i P_{11}(in_i) )</td>
<td>( P_{obs}(out) \times \prod_{i \neq j} P_{11}(in_j) )</td>
</tr>
<tr>
<td>OR</td>
<td>( \prod_i P_{00}(in_i) )</td>
<td>1 - ( P_{00}(out) )</td>
<td>( P_{obs}(out) \times \prod_{i \neq j} P_{00}(in_j) )</td>
</tr>
<tr>
<td>XOR</td>
<td>1 - ( P_{11}(out) )</td>
<td>( P_{11}(in_1)P_{00}(in_2) + P_{11}(in_2)P_{00}(in_1) )</td>
<td>( P_{obs}(out) \times \max_{i \neq j} (P_{00}(in_j), P_{11}(in_j)) )</td>
</tr>
<tr>
<td>Branch/Stem</td>
<td>( P_{10}(stem) )</td>
<td>( P_{11}(stem) )</td>
<td>( \max(\text{Branch observabilities}) )</td>
</tr>
</tbody>
</table>

The final step of computing TVVF is combining appropriate \( P^n \) values. Since we require 2-bit random fault injection for our RCA example, we combine \( P^1 \) and \( P^2 \) to compute the TVVF as shown in Equation 3.

\[
TVVF(RCA, \text{2-bit-flip}) = P^1 + P^2 = 0.374
\] (3)

To compare multiple attacks on one implementation, we calculate the TVVF\((C, A)\) for each attack. Then the max(TVVF\((C, A)\)) value is selected as the TVVF of the hardware structure. The reason is that the TVVF must show the weakness of circuit against fault attacks. Therefore, we choose the highest probability (in other words the easiest attack on the implementation) as the TVVF of the circuit.

IV. RESULTS

We performed two types of experiments. First (Section IV.A), we verified our base assumption that the probability of injecting an exploitable fault is proportional to the number of violated paths. Second (Section IV.B), we computed TVVF for two different attacks on two different AES hardware implementations.
A. Experimental Verification of Fault Generation Probability

In this subsection, we demonstrate that the probability of injecting an exploitable fault is proportional to the number of violated paths. We analyzed two AES SBox architectures: PPRM1 [11] and Boyar-Peralta [12]. We first generated Sbox netlists for an Altera Cyclone IV FPGA. Then, we applied gate-level timing simulation and STA.

First, we performed STA to obtain path delays. We applied Algorithm 1 and 2 to get the theoretical bit-flip injection probabilities. In the gate-level simulation, we gradually decreased the clock period from 6ns to 3.2ns with 20ps precision. For each step, we applied all possible 2552 transitions to Sbox inputs and counted the number of bit-flips at each output bit. Then we computed the experimental bit-flip rate for each output bit, which is the ratio of the number of bit-flips to the total number of input transitions (i.e., 2552). For each step, we also determined the number of violated paths within the fan-in cone of each output bit using the STA results.

Fig. 4(a) shows the experimental bit-flip rate and the number of violated paths for output bit 1 of the PPRM1 Sbox. The experimental bit-flip rate increases with the number of violated paths for a range of clock periods (5.621, 4.441 = 600 steps). For smaller clock periods, the experimental bit-flip rate saturates around 0.5. This shows that after a certain clock period, the behavior of the circuit becomes random. After this point, the chance of getting a bit-flip in an output bit is 0.5. The other output bits of PPRM1 and Boyar-Peralta Sbox architectures also show similar behavior. Fig. 4(b) shows the average number of bit-flips at the output of the PPRM1 Sbox and the number of violated paths. Again, the average number of faulty bits follows the number of violated paths for a certain range and then saturates at around 4. Hence, we can conclude that our assumption is valid.

As shown in Fig. 4(c), for bit 1 of the PPRM1 Sbox, there is a significant correlation (with a correlation coefficient of 0.84) between the experimental bit-flip rate and the theoretical bit-flip probability. However, the theoretical probability is optimistic compared to the experimental bit-flip rate. The reason behind this behavior is that we use simple assumptions in our computations. First, the probability computation in Algorithm 1 is a rough estimation for the actual fault behavior of the circuit. Second, we do not include a false-path analysis phase in STA, which makes our computations conservative. To address this problem, a false-path-aware STA method [13] can be used. Third, our computation assumes that every violated path contributes positively to the bit-flip probability. In practice, multiple faults that are caused by multiple violated paths may mask each other, and thus, they may negatively affect the bit-flip probability. This problem can be addressed by incorporating a functional simulation phase into the computation.

B. An Example Case Study

We applied our methodology for two attacks on two AES implementations as a case study. We computed TVVF for each possible attack/implementation pair. We first compared the feasibility of each attack on each implementation. Then, we compared the vulnerabilities of the AES implementations to these attacks.

The aim of Piret attack [14] is to inject one random-byte fault into the output of SubBytes function of AES algorithm in round 9th. For Piret attack, the SoV is the Sbox of round 9th and the SoP is from the input of ShiftRows of round 9th to the output of round 10th. The Tunstall attack [2] relies on injecting one random-byte fault to the output of SubBytes of round 9th. For Tunstall attack, the SoV is the Sbox of round 8th and the SoP is from the input ShiftRows of round 8th to output of round 10th. For simplicity, we assume that the fault location is first state byte of the AES for each attack. Thus, SoV is the Sbox that corresponds to this byte.

Our AES netlists are generated for an FPGA. The only difference between these two AES implementations is their Sbox architectures. The first one, called P-AES, uses the PPRM1 [11] Sbox. The second one uses Boyar-Peralta [12] Sbox, which we call B-AES.

We first performed an STA analysis on both Sbox architectures. Then we included the observation probability in our computations. Table IV shows the ratio of observability values for each attack. As it is shown, the observability of Piret attack is greater than of Tunstall attack. The reason is that Piret attack injects faults into round 9th and Tunstall attack injects faults to round 8th. The faulty values need to go through a shorter path in Piret compared to Tunstall attack. Then, we computed
the TVVF for each attack using Algorithm 1 and 2. Table V shows the results of this experiment.

TVVF can be used to compare the feasibility of two attacks on a given hardware structure. As seen in Table V, Tunstall attack is 53.4 and 45.5 times less feasible than the Piret attack on P-AES and B-AES, respectively. We can also use TVVF to compare the vulnerability of two implementations to a given attack. Table V shows that P-AES is 1.27 (1.08) times more vulnerable to Piret (Tunstall) attack than the B-AES. Due to the asymmetric structure of PPRM1 and delay-balanced structure of Boyar-Peralta, PPRM1 Sbox is more vulnerable to fault attacks than Boyar-Peralta [15].

### V. Related Work

Many methods assess the reliability of a design to soft errors such as the Architectural Vulnerability Factor (AVF) [16]. These methods are widely used by fault tolerant system designers to make proper reliability/cost tradeoffs. However, they are not applicable to vulnerability analysis to fault attacks, because their assumption is random fault location and timing.

Feiten et al [17] propose a methodology to compute the detection probability of a modeled fault. They model the faults with boolean satisfiability formulas and compute their detection probabilities using an automatic test pattern generation method. Their method only considers propagation probability of an exploitable fault, which corresponds to the SoP part of our method. They ignore the implementation aspects of the design and likelihood of injecting an exploitable fault, which is the SoV part of our method.

### VI. Conclusions

This paper is the first to propose an evaluation metric for assessing the vulnerability of a secure hardware design to fault attacks. This metric considers both generating an exploitable fault and observing the effect of the fault. We divide a cryptosystem into two parts. SoV covers the timing and location of the fault. SoP covers the fault propagation to the output. We perform STA on the SoV to find the likelihood of the fault generation. Then we perform the observability analysis to quantify the propagation of the fault to the output. We also show a case study of this factor on AES algorithm, which compares two fault attacks and two AES implementations.

### VII. Acknowledgments

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

### References


