SoFI: Security Property-Driven Vulnerability Assessments of ICs Against Fault-Injection Attacks

Huanyu Wang, *Student Member, IEEE*, Henian Li, *Student Member, IEEE*, Fahim Rahman, *Member, IEEE*, Mark M. Tehranipoor, *Fellow, IEEE*, and Farimah Farahmandi, *Member, IEEE*

***Abstract*—Fault-injection attacks have become a major concern for hardware designs, primarily due to their powerful capability in tampering with critical locations in a device to cause violation of its integrity, confidentiality, and availability. Researchers have proposed a number of physical and architectural countermeasures against fault-injection attacks; however, these techniques usually come with large overhead and design efforts making them difficult to use in practice. In addition, the current electronic design automation (EDA) tools are not fully equipped to support vulnerability assessment against fault-injection attacks at the design-time for secure hardware development. To perform a design-time (i.e., pre-silicon) evaluation of such attacks, a designer should be aware of various security vulnerabilities and must perform a tedious manual design review, which is time-consuming and hard to ensure effectiveness. Therefore, it is very important to develop an automatic assessment framework to identify the most security-critical locations in a design to fault-injection attacks and place emphasis on protecting those locations. In this paper, we propose an automated framework for fault-injection vulnerability assessment of designs at gate-level, while considering the design-specific security properties using novel models and metrics. The proposed framework identifies the faults that can violate the security properties of the design. As a result, applying local countermeasures will be more effective and the protection overhead will be reduced significantly. Our experimental results on the security properties of AES, RSA, and SHA implementations show that the security threat from fault-injection attacks can be significantly mitigated by protecting the identified critical locations, which are less than 0.6% of the design.**

***Index Terms*—Hardware security, fault-injection attack, security property, vulnerability assessment, computer-aided design.**

I. Introduction

With the emergence of the Internet of Things (IoTs) regime, promising exciting new applications from smart cities to connected autonomous vehicles, security and privacy have emerged as major design challenges. Within the connected computing and sensing components, or the *things* in an IoT system, notably the cryptographic hardware and field programmable gate arrays (FPGAs) in embedded systems, artificial intelligence (AI) accelerators, digital signal processors (DSPs), and microprocessors are all highly vulnerable to diverse forms of physical and non-physical attacks. These attacks can effectively bypass the security mechanisms built in these devices and put systems at risk. Among them, fault-injection attacks have become a major concern to the computer security community, primarily due to their powerful capability in tampering with vulnerable locations in a device and ability for extracting its secrets. Fault-injection attacks can dangerously break the mathematical strength and robustness of the implemented security mechanisms.

In a fault-injection attack, the faults are intentionally injected in a system to compromise its security by causing the denial of service (DoS), achieving illegal authentication, or facilitating leakage of secrets in the system. Fault-injection attacks can be non-invasive (e.g., clock glitching or voltage glitching), semi-invasive (e.g., local heating or laser), or invasive (e.g., focused ion beam), which can be carried out by a variety of techniques and instruments with different cost and precision [1], [2]. Different forms of fault-injection attacks have been successfully demonstrated by researchers in academia as well as practitioners in the industry on many security-critical applications. This includes AES, DES, and RSA encryption algorithms [1], [3], error correction code (ECC) [4], radio-frequency identification (RFID) [5], virtual machines [6], microcontrollers [7], as well as analog sensors [8]. Almost all platforms, such as smart cards, system-on-chips (SoCs), FPGA-based embedded systems, and IoT devices, are vulnerable to fault-injection attacks, which corroborates the criticality of this attack vector [9], [10].

To help prevent fault-injection attacks, many different countermeasures have been proposed in the past decade. Typically, there are two major categories of countermeasures: (i) intrusion detection and (ii) error detection. The first approach relies on detecting the physical facilitators of fault-injection attacks in the design and making the design physically inaccessible, which requires a tamper-proof packaging for the design as well as sensors to detect any physical tampering attempts. This approach has been applied to the IBM 4764 [11], which is an exclusive cryptographic co-processor. However, such approach would be expensive with a large area/performance overhead and design effort, which makes it inapplicable to common industrial devices and IoTs. The other more cost-effective approach against fault-injection attack is error detection, which enables the design to detect the injected faults at runtime. One example of such approach is to use either hardware or time redundancy. However, these approaches may involve 100% area or performance overhead, making it difficult to deploy in practice.

Additionally, there is limited research in assessing the susceptibility of any design to fault-injection attacks at an early hardware design stage, e.g., at gate-level. So, we lack fully equipped automated tools to assess which location of

H. Wang, H. Li, F. Rahman, M. Tehranipoor, and F. Farahmandi are with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail:{huanyuwang, henian.li}@ufl.edu, {fahimrahman, tehranipoor, farimah}@ece.ufl.edu).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3063998, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

the design might be likely injected with faults to result in security violation or leakage of secret information. There is no available well-defined method for the designers to identify a part of the design as the priority to be protected against fault-injection attacks. As a result, the designers often opt for a solution that aims at protecting the whole design, which might not be necessary and practically feasible, and will require significant resources on the chip. Therefore, it is paramount to develop a technique for a detailed and accurate assessment of the design vulnerability to fault-injection attacks at the pre-silicon stage. With such assessment, more efficient local countermeasures can be developed to reduce the overall protection overhead on area, power, and performance while ensuring security.

In addition, from an attacker's point of view, a random fault would probably result in a random output, which may not necessarily help with the attack. Only the faults that can violate security properties, e.g., bypass the internal rounds of an AES, would facilitate a successful attack. If there are faults injected in the design, but there is no security property violated, the threat from the injected faults would be very limited. Therefore, the fault-injection vulnerability assessment should be guided by a set of security properties in the design. By guaranteeing that no security property will be violated by injecting faults, the resiliency and security of the design would be significantly increased.

Further, fault-injection attack assessment should be performed before the chips are fabricated or the design is implemented in an FPGA. Otherwise, it would significantly impact the overall cost of making it secure. Similarly, a vulnerability to fault-injection attack identified at a later stage in the system life cycle could have a substantial cost impact. Hence, the fault-injection attack assessment must be carried out at the early stage of the design process, e.g., gate-level. At the early stages, a designer has much more flexibility in mitigating the threat with a lower cost. However, there is no automated framework to perform a comprehensive fault-injection attack assessment. The current electronic design automation (EDA) tools are not equipped to support fault-injection attack vulnerability assessment. Therefore, in order to perform a design-time evaluation of such attacks, a designer must not only know about the security requirements of the design but also be able to perform a tedious manual design review, which is time-consuming and hard to guarantee the accuracy of results.

In this paper, for the first time to our knowledge, we present a security property-driven vulnerability assessment framework at gate-level against fault-injection attacks in an attempt to bridge the gap between the need for automated security assessment tools and the capability of existing computer-aided design (CAD) tools commonly used in practice. We make the following distinctive contributions in this paper:

- We develop an automated framework, called security property-driven vulnerability assessments of ICs against fault-injection attacks (SoFI), for designs at gate-level, which can be applied to both FPGA and ASIC.
- The SoFI assessment framework is driven by pre-defined security properties that need to be preserved in order to maintain the integrity, confidentiality, and availability of the design.
- SoFI can identify the most vulnerable locations to fault-injection attacks in the design so that by protecting these locations no security properties would be violated.
- A metric is developed to characterize fault models and generate the corresponding global or local fault list in SoFI.
- The fault feasibility metric is integrated into SoFI to evaluate the feasibility of the fault using a setup-time violation based fault-injection technique.
- The SoFI framework can provide design suggestions against local fault-injection techniques for later stages in the IC design flow (e.g., physical design).
- The SoFI framework is evaluated on different benchmarks with different security properties.

The rest of the paper is organized as follows: in Section II, we provide background on fault-injection techniques and common countermeasures. In Section III, we present the SoFI framework in details. The experimental results are provided in Section IV. Finally, we conclude the paper in Section V.

II. Background

A. Fault-injection Techniques

There are a number of fault-injection techniques that have been developed in order to maliciously alter the correct functionality of a computing device. In the case of non-invasive fault-injection attacks, which are inexpensive and the more frequent ones, one can perform clock or voltage glitching or apply electromagnetic (EM) [12]–[14]. For semi-invasive attacks, one can apply optical fault-injection techniques [15]. Finally, in the case of invasive attacks, active fault injections can be done by physical probing [16]–[18]. The most common fault-injection techniques are briefly discussed in the following:

1) Clock Glitching: One very low-cost and non-invasive technique to inject faults is to tamper with the clock signal to cause either setup or hold time violations [19]. For example, the length of a clock cycle can be shortened by driving a premature toggling of the clock signal, as shown in Fig. 1(a). Fig. 1(b) shows a typical sequential logic path. In normal operation, the clock cycle (T_{CLK}) should be longer than the maximum path delay (τ) of the combinational logic to make

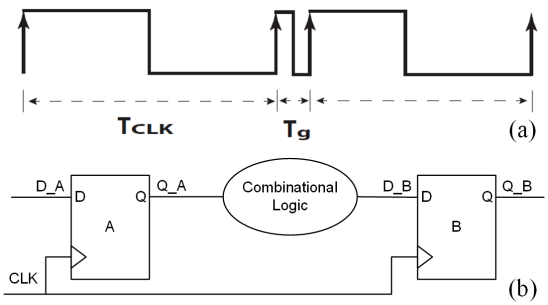

Fig. 1: (a) Glitch in the clock signal, (b) A typical sequential logic path.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3063998, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

sure that the correct/stable value is captured by the sequential element. However, when a clock glitch occurs, T_g is less than τ, so register B may capture the wrong data, in which a fault is injected and will propagate in the circuit. Such a glitch in a processor can result in skipping an instruction or storing incorrect data in the memory modules [20]. In addition, a clock glitch may cause the wrong data to be latched in flip-flops or registers. For example, it has been shown that if one bit of the key is not latched correctly by the key register in a crypto engine, this key bit can be deduced by comparing the faulty and the correct output of the crypto engine. Such faults are transient so that they can be injected without any footprint of tampering [12].

2) Voltage Glitching: Another low-cost technique is tampering with the power supply of a device. For example, one can run the chip with a depleted power supply so that some high threshold voltage transistors would never be open, thus transient faults are injected in the device. Another method is to leverage power spikes so that the setup time requirement of flip-flops may be violated similar to clock glitching, which can cause a processor to skip an instruction or a crypto engine to skip a round of encryption/decryption. This fault-injection technique is commonly utilized to tamper with a program counter, or a loop bound [21]. Usually, voltage and clock glitching are used together to increase the possibility of setup time violation of flip-flops.

3) Electromagnetic (EM): An external electromagnetic field can also be exploited to inject faults. It can cause malfunctioning of a chip or flip memory cell(s). Eddy currents on the chip surface can be induced by the EM field, which can cause a single-bit fault [22]. For example, a gas-lighter can be used to inject EM faults at a very low cost [23]. The single-bit fault injected by EM can be used to facilitate the propagation of secret data, e.g., keys, to observable nodes [22].

4) Light and Laser: A strong and precisely focused light beam or laser can be exploited to induce alterations in one or more logic gates. As shown in Fig. 2, a laser can create electron-hole pairs at the drain of a NMOS and thus create a current pulse. The generated current pulse will charge the load capacitance and create a voltage pulse as a transient fault to be further propagated in the circuit. For example, by targeting one transistor in static random-access memory (SRAM) cell, the cell can be flipped up or down at will [25]. Hence, it is possible for attackers to flip one of the key bits loaded in the cache and deduce the value of that key bit by comparing the output of a crypto process. However, the spot size of the light is physically limited by the wavelength of the photons. It is no longer possible to hit a single SRAM cell using the current optical technologies since the width of the gate dielectric in the advanced technology nodes is now an order of magnitude smaller than the shortest wavelength of visible light. However, it does not necessarily imply the inability to inject a single-bit fault. Agoyan et al., [26] demonstrated how to inject a single-bit fault in a reproducible way, despite the fact that the optical precision of the equipment was not able to target the smallest features of the chip [24].

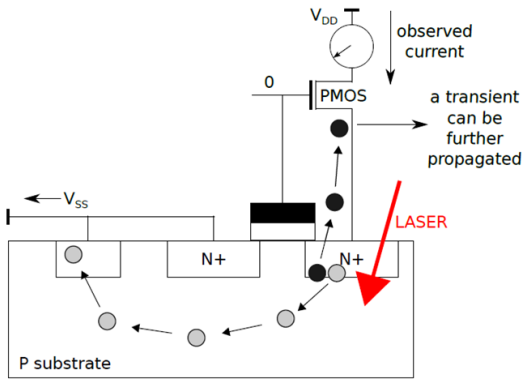

Fig. 2: Impact of laser on a NMOS transistor [24]. Electron-hole pairs are generated at the drain of the NMOS by the laser, which will create a current through the transistors and inject a transient fault to propagate in the circuit.

5) Focused Ion Beam (FIB): The most accurate fault-injection technique uses a focused ion beam, which is a powerful instrument commonly used in the development, manufacturing, and reworking (editing) of semiconductor devices and integrated circuits (ICs) [17], [18]. FIBs use ions at the low beam and high beam currents for imaging surface topology and site-specific milling/deposition, respectively. These capabilities allow designers to cut or add traces to the substrate within a chip, thereby enabling them to redirect signals, modify trace paths, and add/remove circuits. An attacker can use FIB to build a conducting path from chip surface to the internal net so that signals carried on the net can be extracted through this path (probing attack) and transient faults can be injected through this path as well (fault-injection attack).

B. Fault-injection Countermeasures

The countermeasures evolve over time with the sophistication of fault-injection attacks. Since any countermeasure comes at a cost, in practice, they are selected with a good balance between overhead and security. In fact, many countermeasures are developed to make an attack sufficiently expensive for the attacker but not impossible [15]. There are two major categories to protect a design against fault-injection attacks: *intrusion detection* and *error detection*.

1) Intrusion Detection: Countermeasures in this category are developed to detect any attempted tampering with the device and make the device physically inaccessible. They are developed to prevent a specific fault-injection technique. One notable example is using shields (passive or active), in which wire mesh is used to cover a part of or the entire chip to detect an optical fault or probing attacks [17], [18]. In addition, analog sensors can be applied in the chip to detect different fault-injection attacks. For example, light sensors and frequency detectors are used to detect optical fault-injection and clock glitching, respectively [27]. The main drawback of the intrusion detection-based countermeasures is their high cost with large overhead and design efforts.

2) Error Detection: This approach modifies the design to allow the detection of injected faults at the algorithm level. One common method is concurrent error detection (CED), which can check the correctness of the algorithm

by introducing redundancy. Typically, there are three types of redundancy in terms of resources: hardware, time, and information [28]. As an example, hardware redundancy indicates adding extra hardware into the device to either detect or correct the impacts of the faults injected. The most common example is the triple modular redundant (TMR) structure [29] which has three identical modules whose outputs are voted for correct functionality. Time redundancy can also be utilized to detect faults by re-running the same process on the same hardware. However, these approaches introduce at least 3X/2X area/performance overhead, respectively, which is too high to be practical for large-scale complex designs [30]. As another example, information redundancy is based on error detection codes (EDCs) or error correction codes (ECCs), which may require a smaller overhead. The main drawback of the EDC/ECC based countermeasures, however, is possible lower fault coverage because not every combination of errors can be detected, e.g., parity-based EDCs are capable of detecting any fault that consists of an odd number of bit errors. However, an even number of bit errors occurring in a single byte will not be detected [31].

C. Related Work

Several researches have been reported to evaluate the susceptibility of a design to fault-injection attacks [32]–[36]. However, these frameworks are at the algorithm or software level without considering the feasibility of the faults and the different hardware implementations. For those gate-level fault evaluation tools, VerFI [37] is proposed to examine the desired fault detection/correction capabilities and reveal undetected faults, however, the vulnerability of the original design is not assessed. [38] is actually a fault simulator without automated security assessment. SOLOMON [39] uses formal methods to map vulnerable regions in the cipher algorithm to specific locations in the hardware, however, it can only be applied to crypto cores against DFA. Further, almost all these approaches can be only applied to one type of designs, such as block cipher with a focus on DFA-based attack, which are difficult to be applied to other designs. In fact, there can be many fault attacks on the control logic, e.g., FSMs, to result in confidentiality, integrity, or availability issues.

D. Threat Model

In this paper, we restrict our focus to the vulnerability assessment of common fault-injection attacks (e.g., clock glitch or laser) at gate-level. An adversary's objective is to tamper with security-critical locations in the design using one of the fault-injection techniques to violate targeted security properties of the design, such as confidentiality or integrity. To offer a comprehensive solution, we assume a *strong attack model* where the adversary has full gate-level information of the design from various sources, such as untrusted foundry, performing reverse engineering, stealing the IP, or cooperating with a rogue employee in the design house. We also assume that the attacker is able to achieve the highest theoretical resolution that each fault-injection technique can perform, e.g., clock glitching can inject a fault in a specific register by extending the data path delay of that register. Therefore, the actual vulnerable locations from a practical fault-injection attack (e.g., lower resolution) standpoint would always be a subset of the identified critical locations. Hence, the actual resilience of the design against fault-injection attacks may be stronger than the assessment provided by the SoFI framework.

III. SoFI Framework

Although fault-injection has received significant attention over the past decade as a strong threat on security critical designs and applications, unfortunately there has been no comprehensive study to formally model the faults as well as their effects induced by different fault-injection techniques. In addition, during the propagation of the faults injected in the system, there are vulnerable locations that attackers expect to target to either cause confidentiality or integrity violations. For example, the attacker can inject faults during the initial round in the finite state machine (FSM) of the AES controller to skip the intermediate rounds and go to the final round directly so that the plaintext or key could be leaked. The faults can be injected directly at the FSM state registers or their fan-in cone cells. In the case of a microprocessor, an attacker may want to disable a password checking function, obtain access to the protected area of the memory, obtain access to the shared bus when sensitive information is being transferred, etc. Hence, it is critically important that designers are able to define a set of *security properties* (SP) that they should be enforced to preserve the confidentiality, integrity, and availability of the design. However, fault-injection would enable the violation of those properties. Fault-injection is carried out for any or all of the three features of a secure device, i.e., confidentiality violation, integrity violation, or denial of service (DoS).

If a fault injected in a design does not violate any of these secure design features, it will not be an effective fault-injection attack. Hence, to verify the effectiveness of fault-injection attack, in this work, we focus on security properties. SoFI targets faults on security properties in the design, making the analysis localized by focusing only on the part of the circuit for which the security property is being checked.

The overall flow for the SoFI framework is shown in Fig. 3. Generally speaking, the critical locations to fault-injection attacks are identified by checking whether any security properties can be violated if the faults are injected at these locations. The more critical locations identified from the design, the more vulnerable the design is to fault-injection attacks. The SoFI framework takes fault-injection technique's specification, executable security properties, and the gate-level design as the inputs (Section III-A illustrates the requirement of executable security properties for fault-injection assessment). First, to map a specific fault-injection technique (e.g., clock glitch or laser) in the assessment, the fault models are characterized from the specification of the targeted fault-injection techniques and a fault list is generated based on the fault model and the executable security properties (discussed in Sections III-B and III-C in details). Then, the fault simulation is performed and the critical locations are identified (illustrated in Sections III-D and III-E). Finally, the fault feasibility analysis is conducted

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3063998, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

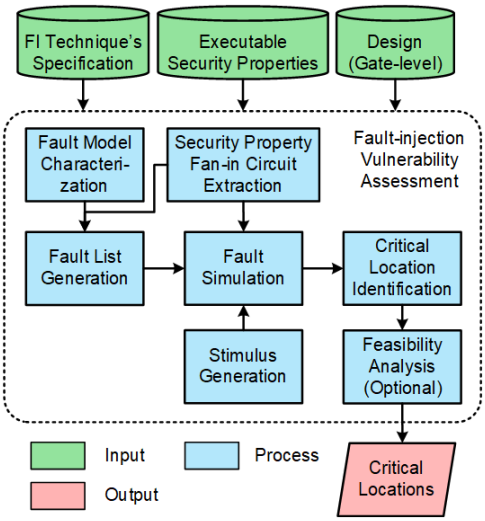

Fig. 3: SoFI: our security property driven fault-injection vulnerability assessment framework.

to check whether the faults can be practically implemented by the specific fault-injection technique (as illustrated in Section III-F).

A. Executable Security Properties

As defined in [40], a security asset in a chip is a value that is worth protecting against different adversaries. It can be either tangible or intangible, such as passwords or one's fingerprint that defines the accessibility to a mobile phone. For integrated circuits, the asset could be encryption keys, obfuscation keys, device configurations, manufacture's firmware, communication credentials, configuration bits, etc. These assets should be properly protected against various attacks to avoid any secret leakage, illegal authentication, loss of profit, or reputation decline. Therefore, we need corresponding security properties (SP) specifying the secure utilization of the design as a guidance to protect these assets [40], [41] and improve the scalability of the proposed SoFI framework. Further, by tampering security-critical locations using fault-injection techniques, the attacker can violate the security properties and achieve privileges to leak/tamper asset information. Hence, the capability to violate one of the security properties can be utilized as the criteria to identify the critical locations. If the injected faults cannot violate any of the security properties, the underlying threat is much less of a concern compared to the faults that can violate properties. This can help with prioritizing the critical faults and locations, and to develop effective and resource-constrained local countermeasures. Note that security properties are important inputs to SoFI, however, how to identify them is out of the scope of this paper.

As an input to SoFI, the appropriate selection of security properties dictates the quality of the assessment because not all security properties are suitable for fault-injection vulnerability assessment. For example, the security property described in [40], "exposed area [18] to probing attacks should be lower than a threshold value", is a good security property to mitigate the threat from probing attacks at the layout level. However, it is not suitable for our fault-injection vulnerability assessment since this security property cannot be violated by fault-injection attacks; further, this property is at layout level instead of the gate level. Therefore, one requirement for identifying the security property subset in this work is that *the security property should be related to or can be violated by one of the fault-injection attacks*.

In addition, most of the security properties available in the literature [40], [41] are described at a high level (often using natural language) without detailed metrics. It may not be clear how to check if the security property is violated in the target level of abstraction of the design. Therefore, the second requirement for the security property in this work is that *the security property should be converted to one or more executable formal presentations with explicit verification metrics*. For example, the security property, "AES internal rounds cannot be skipped and directly jumped to the final round", can be converted to an executable one as described below.

SP1: The *done* signal that indicates the completion of ten AES rounds cannot be raised in the 1^{st} round.

In this case, the time (1^{st} AES round) and the location (*done* signal) to check the security property violation are clear, which is more executable than the original one. Note that the similar properties can be extracted for any intermediate round.

If SP1 is violated, the 1^{st} round AES results would be leaked at the primary output in which the encryption strength provided by the AES algorithm would be significantly reduced [41]. Fig. 4(a) shows the extracted fan-in circuit of the *done* signal which is a 4-bit counter. Fig. 4(b) shows the waveform of SP1 in AES. *done* is the correct waveform in the normal operation while *done'* is the faulty waveform with the security property violated. When the *ld* signal is raised at clock cycle 24, the keys and plaintext are loaded in the design and the AES encryption operation starts. It takes 2 clock cycles (25 and 26) to initialize the key expansion and the 1^{st} AES round starts from the next cycle (cycle 27). In the normal operation, it takes 10 rounds to encrypt the plaintext, so the *done* signal will be raised 9 clock cycles after the 1^{st} round (cycle 36) as shown

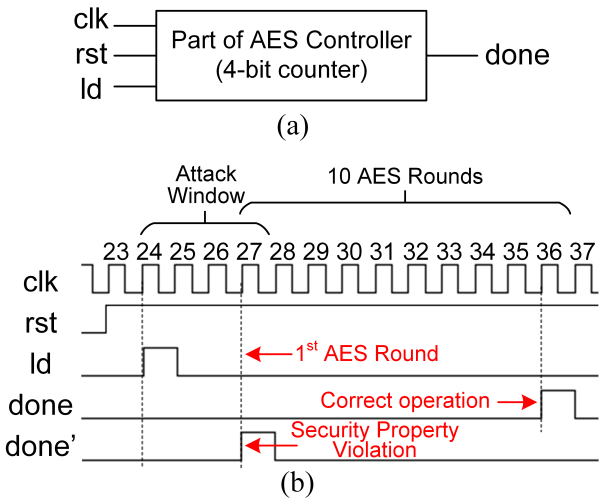

Fig. 4: (a) Fan-in circuit, (b) waveform of SP1.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3063998, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

in Fig. 4(b). However, when faults are injected in the security property fan-in circuit as shown in Fig. 4(a) and the *done* signal is raised 3 clock cycles after the AES keys are loaded (cycle 27), which means the output can be read, however, it is not fully encrypted. Therefore, the security property is violated and the strength of the encryption algorithm is compromised. The goal of the fault simulation is to identify those faults that can enable this security property violation.

Based on one security property, more variant security properties can be created to improve the completeness of evaluation. For example, SP1 can be extended: the *done* signal that indicates the completion of ten AES rounds cannot be raised in $1-7$ rounds. Similar to the spatial and temporal locality concept in cache, if violating a security property at time *t* and location *l* would result in the leakage of asset information, violating a similar security property at time *t+1* and location *l+1* would be likely to leak asset information as well. Therefore, by adding more variant security properties, the completeness of the evaluation can be improved.

B. Fault Model

There are many techniques to inject faults in a system, such as clock/voltage glitching, EM, laser beam, or FIB as discussed in Section II-A. The mechanism of fault generation from these techniques are fundamentally different. Hence, the format and impact of the faults injected by these techniques also differ significantly. For example, the faults injected by clock glitching may be *global* and *random*, while the faults injected by laser may be *local* and *deterministic*. Therefore, a comprehensive modeling of the existing fault-injection techniques is necessary to enable fast, reliable, and accurate assessment of the fault-injection vulnerability. Without such models, it is difficult to evaluate how these faults injected by different techniques would impact the circuit and security properties.

A fault model is a set of attributes characterized from the physical impact of the faults injected by a specific technique. It converts a physical event of fault-injection into a logical model. Using the logical model, we can simulate the fault injection and propagation in the digital circuit and analyze the impact of the faults for different fault injection techniques. Different fault-injection techniques differ greatly in their ability to control the location and time (spatial and temporal) of the injected faults, in the number of bits affected, etc. The list below shows fundamental attributes for our proposed fault model characterization.

- **Fault category:** Faults can be classified into two major categories: (i) *global faults* where they are injected globally across the whole design, such as clock faults or voltage faults, and (ii) *local faults* where they are injected locally in a small portion of the design, such as laser or FIB.
- **Fault-injection location:** *Complete control* spatially means a single specific cell in a design can be affected, e.g., using a high-resolution laser. *Some control* means a specific group of cells or a region can be targeted, but the sub-location within the specific group or region is unknown, e.g., faults can only be injected into sequential cells using the clock-based fault-injection technique. *No control* represents the faults that would be injected randomly among all cells in the design.
- **Fault-injection time:** *Complete control* temporally indicates that the faults can be fully synchronized with the design or operation and can be injected at a specific point of time. *Some control* means a set of operations or clock cycles can be targeted, but not a specific one. *No control* represents faults that can only be injected at a random time.
- **Fault type:** Faults can have different effects on the chip: *stuck-at fault, bit-flip fault, set/reset fault, destructive fault (permanent damage), etc*.
- **Fault duration:** Faults can have different duration: *transient* and *permanent*.

TABLE I: Our proposed fault model characterization.

Technique	Fault Category	Spatial Control			Temporal Control			Fault Type	Fault Duration
		No	Some	Compl	No	Some	Compl		
Clock Glitching	Global	✓	✓			✓	✓	Bit-flip	Transient
Voltage Glitching	Global	✓	✓		✓	✓		Bit-flip	Transient
EM	Global	✓	✓		✓	✓		Bit-flip	Transient
Laser	Local		✓	✓		✓	✓	Bit-flip	Transient
FIB	Local			✓		✓	✓	Bit-flip Stuck-at	Transient Permanent

In addition, the specification (i.e., high or low resolution) of the fault-injection technique and the targeted technology node of the design could also impact the fault model. For example, a high-resolution laser can inject a single bit fault at any cell in a design with large feature size (complete control on fault location). However, a low-resolution laser may impact tens/hundreds of cells at a time and inject a fault with multiple-bit flips in a design with small feature size (some control on fault location).

Table I shows the characterized fault model for different fault-injection techniques. Clock glitching, voltage glitching, and EM are classified as global fault-injection techniques, while laser and FIB are classified as local fault-injection techniques. Generally, local fault-injection techniques are more controllable in fault location and time. For fault type and duration, all fault-injection techniques evaluated in the table can be modeled as bit-flip and transient, respectively.

C. Fault List Generation

A detailed fault list is required for the fault simulation. The fault list is generated based on the security property and fault model. Each fault contains four attributes: fault injection time, fault location, fault type, and fault duration. For most fault-injection techniques, the fault type can be modeled as *bit-flip* and the fault duration can be modeled as *transient* for one clock cycle as shown in Table I. The following content of the section illustrates how the fault injection time and location are determined.

For the fault-injection time, we assume the strongest attacker who has *complete control* on the clock cycle at which the fault would be injected (note that in practical scenarios, this assumption may not always hold true; therefore, the attack success is actually lower), which means the attacker is able to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3063998, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

inject fault at any specific clock cycle. The exact fault injection time is at any clock cycle within the attack window. The attack window usually starts with raising a control signal, like *start, load, etc.*, that launches a new round of operation in the design. The attack window would close when the security property is checked. Taking the SP1 mentioned at the end of Section III-A as an example, the attack window starts when the *ld* signal is raised (clock cycle: 24) to start the AES and the attack window is closed in the first AES round (clock cycle: 27) at which the security property is checked as shown in Fig. 4(b). Therefore, if only one-time attack is considered in which the faults are injected only once, the fault-injection time falls in any one of the clock cycles from 24 to 27 for SP1. So, the time option for fault-injection (T_f) is 4, meaning there are opportunities in 4 clock cycles to inject a fault.

For the fault location, first, the fault are generally injected in the fan-in cells of the signal where the security property is checked. Second, it depends on the fault category: global or local. For the fault-injection techniques that result in global faults, such as clock glitching or voltage glitching, in order to reduce the simulation workload, the potential fault-injection locations can be modeled only at sequential cells because only the latched faults at sequential cells are impacted, and their contents are propagated in the design. This is similar to single event upset (SEU) faults, but multiple events are also considered in this work. For the fault-injection techniques that result in local faults, such as FIB or laser, the potential fault-injection locations can be any cells (sequential and combinational cells) in the design, which is similar to the union of single event upset faults and single event transient faults (SEU+SET), but multiple fault locations are also considered in this work.

Practically, the actual concurrent fault locations in an attack can be any combination of cells in the potential injection locations. However, if the design size is large, it is not necessary to simulate all possible combinations of fault locations because the possibility of implementing a specific fault combination decreases exponentially with the increase in concurrent fault locations. For example, the possibility of injecting a fault at only one specific cell among 1000 cells might be 1×10^{-3} $(1/\binom{1000}{1})$. However, the possibility to inject concurrent faults at 20 specific cells among 1000 cells is 3×10^{-42} $(1/\binom{1000}{20})$ which is exponentially lower than the possibility to guess the key value of a 128-AES module at once $(1/2^{128} = 3 \times 10^{-39})$. Further, for some fault-injection techniques, like laser, the number of concurrent fault locations is also limited by the number of laser beams (typically, it is only one). Therefore, a small threshold can be set for the number of concurrent fault locations considered in the fault simulation. This threshold can vary depending on the fault-injection technique's specification, simulation capability, and the benchmark size. In practice, this threshold can be set to 1 or 2.

For example in SP1, first, the fan-in circuit of this security property, i.e., the fan-in circuit of the *done* signal, is extracted as shown in Fig. 4(a). The RTL of the AES module is from OpenCores [42] and the gate level netlist is synthesized using Synopsys Design Compiler with SAED32nm library. Table II shows the general information of the extracted fan-in circuit. It is a small logic, part of AES control logic. There are only

TABLE II: Fan-in circuit information of SP1.

Inputs	Outputs	Nets	Sequential Cells (N_S)	Combinational Cells (N_C)	Total Cells (N_T)
3	1	36	5	26	31

three primary inputs associated with SP1: *clk* (clock), *ld* (key load), and *rst* (reset). The only output is the *done* signal which is the checkpoint of the security property. By checking when the *done* signal is raised, one would know whether the security property is violated.

For a global fault-injection technique assessment, the potential fault-injection locations are modeled only at the output of sequential cells to minimize the fault simulation workload. As shown in Table II, the number of sequential cells (N_S) in this fan-in circuit is 5. Since it's a small number of sequential cells, all possible combinations of these 5 locations are considered as fault-injection locations in the fault list. So, the concurrent fault threshold (CF_{th}) is set to 5. If the potential fault-injection locations are large, only 1 or 2 concurrent fault locations are considered in this work as discussed earlier. Hence, the total number of faults for global fault-injection technique assessment (TF_g) in the fault list is:

$$TF_g = T_f \times \sum_{i=1}^{CF_{th}} \binom{N_S}{i} = 4 \times \left[\binom{5}{1} + \binom{5}{2} + \binom{5}{3} + \binom{5}{4} + \binom{5}{5}\right] = 124 \quad (1)$$

where T_f is showing the number of clock cycle options (24-27) for fault-injection. For the local fault-injection technique assessment, the potential fault-injection locations are the output of all cells in the circuit. As shown in Table II, the number of total cells (N_T) is 31 in total. As illustrated before, it is not necessary to consider all possible combinations of the potential fault-injection locations. Here, up to 4 concurrent fault locations ($CF_{th} = 4$) are considered in this fault list and the total number of faults for local fault-injection technique assessment (TF_l) is:

$$TF_l = T_f \times \sum_{i=1}^{CF_{th}} \binom{N_T}{i} = 4 \times \left[\binom{31}{1} + \binom{31}{2} + \binom{31}{3} + \binom{31}{4}\right] = 145,824 \quad (2)$$

D. Fault Simulation

After generating the fault list, faults need to be simulated to check their effects. In our framework, the fault simulation is performed using Z01X from Synopsys [43], which is a fast and comprehensive functional fault simulator. By injecting faults in the design, the attacker can achieve privileges to leak/tamper asset information. Hence, the critical locations of the design to fault-injection attacks should be identified to avoid security property violation. Protection of these critical locations with high priority ensures that the assets cannot be leaked/tampered and the security properties cannot be violated by fault-injection attacks.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3063998, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

In a fault-injection attack, the faults might be injected directly at locations where a property is checked (e.g., *done* signal in SP1) or in their fan-in circuit (e.g., Fig. 4(a)), if it is easier. Therefore, during fault simulation, we inject faults at both locations based on the fault model characterized from the targeted fault-injection technique and the corresponding fault list, considering the security property as illustrated in Sections III-B and III-C. If the injected faults violate the security property, the fault simulation would capture this violation and mark the corresponding faults. The critical locations can be extracted from those faults. These critical locations should be properly protected using countermeasures against fault-injection attacks.

Different security properties may involve different portions of the design, which means the property can only be violated if the faults are injected or reach to the specific portion of the design. In most cases, this portion would be the fan-in circuit of the location referred in the security property. For example, in the case of SP1, the security property can be violated only if the faults are injected in the fan-in circuit of the *done* signal. Therefore, to increase the efficiency of fault simulation, the fan-in circuit of the security property would be extracted and the fault simulation would be performed on the extracted circuit only.

Besides the fault and the design implementation, violation of a security property also depends on the stimulus vector. With the same fault in a design, some stimulus can successfully violate the security property, but some others cannot because the toggling activities of internal signals vary from different stimulus. Therefore, by feeding different stimulus to the simulation with the same fault, the rate of the security property violation for the targeted fault can be calculated. Faults with zero security property violation rate can be defined as *non-effective faults*. Faults with non-zero security property violation rate can be defined as *effective faults*. Table III summarizes the terms used in this paper and their definitions. More terms will be illustrated in the following sections.

If the extracted fan-in circuit of the security property is a part of the control logic (e.g., FSM), specific input patterns that are commonly used for the functionality of the control logic can be utilized as the stimulus in the fault simulation. For example, in SP1, since one input is the clock and the other two inputs are control signals, these signals are switching with a fixed pattern instead of random transitions. Therefore, only one specific pattern is applied as the input stimulus to the extracted fan-in circuit of SP1 as shown in Fig. 4(b). If the extracted circuit is part of an arithmetical logic, e.g., arithmetic-logic unit (ALU), random input vectors could be used as the stimulus.

Once the security property, fault list, stimulus, and the security property fan-in circuit are available, the fault simulation can be performed.

E. Critical Location Identification

When the fault simulation is done, we would know whether a fault in the fault list is effective at violating the target security property. One fault can consist of one or more fault locations.

TABLE III: Terminologies used in the paper and their definitions.

Terms	Definitions
Non-effective faults	The faults with zero security property violation rate
Effective faults	The faults with non-zero security property violation rate
Critical faults	A subset of effect faults in which all fault locations are necessary to the security property violation
Feasible faults	A subset of critical faults that can be implemented by a setup-time based fault-injection technique
Critical locations	A set of locations that has overlap with every critical or feasible fault and the set size is minimized

As an example shown in Fig. 5, its fault list is shown in Table IV. As we can see, faults #1-3 only have one fault location, while faults #4-6 have two fault locations and fault #7 has three fault locations. If an effective fault consists of two or more fault locations, not every fault location is necessarily contributing to the security property violation. In other words, injecting faults at a subset of the fault locations of an effective fault may still violate the security property. Therefore, faults with all fault locations contributing to the security property violation are defined as *critical faults*. For the example shown in Fig. 5, we consider that the following security property: *the output of cell C should not be 0*. Also, assume that faults can be injected at any combination of the output of cells A, B, and C. Table IV shows the fault list and the identified critical faults in this case. Three possible fault locations (cell output: A, B, and C) result in 7 different faults considering all combinations of the three fault locations ($\sum_{i=1}^{3}\binom{3}{i}$). Except for faults #1 and #2, all other faults can effectively violate the security property. However, one can see that some fault locations are not critical to the property violation. For example, a single location fault at cell C (fault #3) alone can violate the security property. It is the critical contributor to the violation, so this fault is identified as a critical fault. Any other fault that contains the fault location at cell C (e.g., fault #5-#7) can violate the security property because of the existence of the fault at cell C, instead of the faults at other fault locations. They are effective faults but not critical faults. Another critical fault identified in this case is the fault at location A+B (fault #4: concurrent fault at A and B). Although there are 5 effective faults in this example, only 2 can be identified as critical faults.

Taking into consideration that every location in a critical fault is vital to the violation of the security property, if the fault cannot be injected at one of the locations in a critical fault (e.g., location A of fault #4 in Table IV), the corresponding

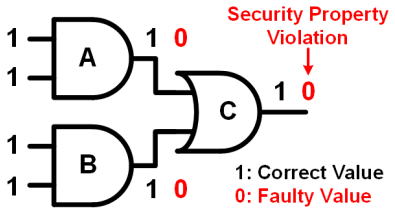

Fig. 5: Critical fault identification.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3063998, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

security property will never be compromised by the attacker. Therefore, for all critical faults, if at least one fault location is properly protected, all critical faults cannot be implemented. Hence, the *critical location* is defined as a set of locations that covers all critical faults and the set size is minimized.

Table V shows one example of critical locations identified from critical faults. In this example, there are 4 critical faults with different fault locations that every fault location is contributing to the security property violation. Without the critical location identification, all 7 fault locations (H-N) should be protected against fault-injection attacks. However, location H can be identified as *critical location* with the set size being only one. Therefore, by only protecting this one critical location, all the 4 critical faults would never be enabled. In addition, another location set {I,J,K,M} covers all critical faults as well. However, the set size is 4 which is not the minimal size. Hence, this location set is not the critical location.

F. Fault Feasibility Analysis

In the fault simulation, each fault can be injected precisely at the desired location and time. However, in practice, even with the fault model characterization, not every fault in the fault list can be executed corresponding to its model on physical devices by the specific fault-injection technique since different techniques may have different requirements and limitations on where and when the fault can be injected. Therefore, we perform the fault feasibility analysis for specific fault-injection techniques, so that the critical faults and critical locations identified from Section III-E make sense in a practical attack.

This step is optional in the SoFI framework because of two reasons. First, the fault-injection techniques are always evolving where some infeasible faults/attacks today may become possible in the near future. Hence, it is desirable to always protect critical locations against all critical faults as long as the protection overhead is acceptable. Additionally, our current assessment is performed at the gate-level design phase. It might be difficult to evaluate the feasibility of some fault-injection techniques at the gate-level before the following design phases (e.g., physical design). For example, the feasibility of a fault using laser-based fault-injection technique is mainly dependent on the layout information of the design, which is unknown at the gate level design stage. Hence, for some fault-injection techniques, such as laser, the fault feasibility analysis cannot be performed for the specific critical faults. However, some design suggestions can be made for later design phases (e.g., physical design) to make the critical faults infeasible.

1) Feasibility Analysis of Setup-time Based Fault-Injection Techniques: Many fault-injection techniques, such as clock glitching or voltage depletion, are leveraging the setup-time violation of flip-flops to inject faults. Clock glitching reduces the clock period and the voltage depletion extends the path delay so that the setup time constraint of the flip-flops can be violated in both scenarios. However, the setup time violation cannot guarantee a bit-flip fault to be injected into the design. Essentially, when the setup time is violated, the flip-flop will latch the value of the previous clock cycle. If the value of the current clock cycle in normal operation is consistent with the previous cycle, the flip-flop will still latch the correct value even if the setup time is violated. Therefore, the bit-flip fault can only be injected in flip-flops when a state transition is expected in normal operation.

Considering the circuit in Fig. 1(b) and assuming the combinational logic is only delay buffers, the correct value of flip-flop A and B in normal operation is shown in columns 2 and 3 of Table VI. There is only one transition for flip-flop A at cycle T and it is propagated to flip-flop B in the next cycle (T+1). Column 4 of this table shows the latched value in flip-flop B if its setup time is violated in the corresponding clock cycle. We can see that only the value at cycle T+1 is different from the correct value in column 3, so that fault can only be injected at cycle T+1. In other cycles, even though the setup time of flip-flop B is violated, it can still latch the correct value because there is no transition at that cycle. Therefore, for flip-flop B in this example, fault at cycle T+1 is a feasible fault and faults at other cycles are infeasible because they cannot be practically implemented by the setup-time based fault-injection technique. SoFI can automatically find such feasible faults by going through the gate-level simulation traces.

After the fault-injection vulnerability assessment is done, local countermeasures can be developed to protect the identified critical locations. This will ensure all security properties evaluated in the assessment would never be violated, and also the protection overhead would be significantly reduced because critical locations should be a very small portion of the design. In addition, because our SoFI is performed at early stage of the IC design flow, the design might be altered to address these vulnerabilities by using different FSM encoding, applying different timing constraints, etc.

2) Design Suggestions against Local Fault-injection Techniques: For those local fault-injection techniques, such as

TABLE IV: Fault list and critical faults.

Fault Index (#)	Location (Cell output)	Effective Fault	Critical Fault
1	A	No	No
2	B	No	No
3	C	Yes	Yes
4	A+B	Yes	Yes
5	A+C	Yes	No
6	B+C	Yes	No
7	A+B+C	Yes	No

TABLE V: Example critical location identification.

Critical Fault Index (#)	Location (Cell output)	Critical Location
8	H+I	H
9	H+J	
10	H+K+L	
11	H+M+N	

TABLE VI: The latched value of flip-flops in Fig. 1(b).

Clock Cycle	Q_A	Q_B	Q_B'
0∼T-1	0	0	0
T	1	0	0
T+1	1	1	0
T+2∼End	1	1	1

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3063998, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE VII: International Technology Roadmap for Semiconductors (ITRS) 2013 [44].

Node	Pitch	Year
16nm	80nm	2013
10nm	64nm	2015
7nm	50nm	2017
5nm	40nm	2019
3.5nm	32nm	2021
2.5nm	26nm	2023
1.8nm	20nm	2025

laser or FIB, the faults feasibility depends on the physical implementation of the design, such as place and route. One limitation of these local fault-injection techniques is the number of laser beams/focused ion beams. Typically, there is only one laser beam available for fault-injection. Hence, if a critical fault requires two or more concurrent fault locations, e.g., cell A and cell B, by placing cell A and cell B with a far enough distance that larger than the maximum spot size of the laser, this critical fault would never occur by the laser. Such design suggestions can be made in our assessment framework, so that all critical faults with two or more locations would never be implemented using the laser with only one beam.

Another limitation of these local fault-injection techniques is the need for the best resolution (minimum spot size) for advanced technology nodes. Taking laser as an example, the resolution R is a function of wavelength λ [45]:

$$R = \lambda/(2\ \text{NA}),\ \text{NA: Numerical Aperture (in air} < 1) \quad (3)$$

Typically, a near infra red light ($\lambda \approx 1\mu m$) is used for laser fault-injection from the backside. Hence, R is at best around 500 nm. Table VII shows the technology trend targeted in 2013 [44]. From the table, we can see that for the latest technology nodes with pitch size <100 nm, tens or even hundreds of cells would be involved in one laser spot, which means if the attacker is targeting only one cell, it would be very challenging to inject fault only at that cell as all the cells around the target cell within the laser spot would potentially experience injected faults. Hence, if a critical fault at location A can violate a security property while a fault at location A+B cannot violate the security property, by placing cell A and cell B closely enough to make the faults injected at these two cells at the same time, the threat from this critical fault can be mitigated. Further, fan-out cells of cell A can be placed closely around cell A so the propagation of the fault at cell A may be blocked by the fault injected at its fan-out cells. By creating such placement constraints, the critical faults with only one fault location can be mitigated significantly.

IV. Experimental Results

In this section, the SoFI framework is evaluated on different benchmarks. Our experiments are designed to show how efficient the assessment flow is and how many critical locations to fault-injection attacks can be identified in the design. For each benchmark, its gate-level netlist as well as the corresponding security properties are considered for the evaluation of SoFI.

All benchmarks used in this work are from OpenCores [46]. They are described in register-transfer level (RTL) code and synthesized using Synopsys *Design Compiler* with SAED 32nm technology library. The fault simulation is performed using Synopsys *Z01X* on Red Hat Linux server 7.8 with Intel Xeon CPU E5-2640 @ 2.6GHz.

TABLE VIII: Fault simulation results for SP1.

Fault Category	Total Faults	Effective Faults	Critical Faults	Feasible Faults	Critical Locations	% in AES
Global	124	20	3	1	1	0.010%
Local	145,824	9,563	83	NA	6	0.058%

A. Information Leakage Results

As an example across the SoFI framework section, the SP1 definition was given in Section III-A as a security property to preserve information leakage. To check the resiliency of this property against fault injection attacks, the AES benchmark is synthesized with 10290 cells in total; the fan-in circuit of SP1 is extracted whose information is shown in Table II; the fault list is generated with 124 global faults and 145,824 local faults as illustrated in Section III-C; one pattern is applied to the fault simulation as illustrated in Section III-D and Fig. 4 (b).

Table VIII shows the fault simulation results for SP1. For the global fault-injection technique assessment, 20 effective faults that can violate SP1 are achieved directly from the fault simulation. Among these effective faults, 3 critical faults that are actually contributing to the security property violation are identified as shown in Table IX. Only one feasible fault that can be implemented by setup-time-violation-based fault-injection technique is identified from the feasibility analysis. Therefore, only one critical location is identified from the feasible fault, which is 0.01% (1/10290) of the AES module, so that by protecting this critical location, the feasible fault would never be able to cause violation. For the local fault-injection technique assessment, the feasibility analysis is not performed because the layout information of the design is not available at the gate-level assessment. 6 critical locations are identified, which is 0.058% (6/10290) of the AES module. Therefore, by protecting these 6 critical locations using redundancy for example, all critical faults would not be enabled using local fault-injection techniques.

To understand how the identified critical faults can violate SP1, the 3 critical faults for global fault simulation are extracted. Table IX shows the time (i.e., clock cycle #) and location of the 3 critical faults. The 5 sequential cells in the circuit are done_reg and dcnt_reg[3:0] whose outputs are *done* and *dcnt[3:0]*, respectively. dcnt_reg[3:0] is the output of a 4-bit counter that counts the AES round. The waveform of output signals of the 5 sequential cells is shown in Fig. 6. When the *ld* signal is raised, the counter is set to 11 (b'1011) in the next cycle. Then, it is counting down from 11 to 0. When the

TABLE IX: Critical faults for global fault simulation.

Critical Fault Index (#)	Time (clock cycle #)	Location	Feasible Faults
1	27	done	No
2	25	dcnt[3]+dcnt[0]	Yes
3	26	dcnt[3]+dcnt[1]+dcnt[0]	No

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3063998, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

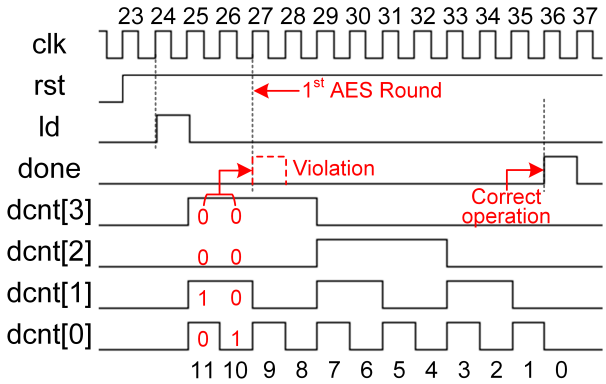

Fig. 6: Waveform of 5 sequential cells for SP1.

counter is 1 (b'0001) the *done* signal will be raised in the next clock cycle.

The security property violation requires that the *done* signal is raised at clock cycle 27. Therefore, it is easy to understand if the fault can be injected directly in *done* at cycle 27, the security property will be violated, which is the case of fault #1 in Table IX. In addition, if the counter is set to 2 (b'0010) at cycle 25, the *done* signal will be raised two cycles later (cycle 27). Similarly, if the counter is set to 1 (b'0001) at cycle 26, the *done* signal will be raised one cycle later (cycle 27). In both cases, the security property is violated as well and these two cases are implemented by faults #2 and #3 in Table IX, respectively. Therefore, the fault simulation results are correct and consistent with the circuit logic analysis. Further, only fault #2 is feasible by setup-time-violation-based fault-injection techniques because there are transitions for dcnt[3] and dcnt[0] at cycle 25 as shown in Fig. 6.

B. Security Properties in FSMs

Many fault-injection attacks focus on analyzing the fault effects on data paths. However, finite state machines in the control path are also vulnerable to fault-injection attacks [47]. The security of an SoC can be compromised if the FSMs controlling the SoC are tampered by fault-injection attacks. Hence, the security properties to protect FSMs should be considered and the FSM's vulnerability to fault-injection attacks should be assessed using SoFI. In this subsection, 4 security properties in the FSM of AES, RSA, and SHA controllers are evaluated, respectively.

1) FSM in AES Controller: The FSM of AES controller is composed of five states: 1) Wait Key; 2) Wait Data; 3) Initial Round; 4) Do Round; and 5) Final Round. During Wait Key and Wait Data states, the secret key and plaintext are loaded into the AES data path, respectively, while during Initial Round and Do Round states, ten rounds of AES occur. After ten rounds, the Final Round state is reached and the result is latched to the output registers. One possible attack can be implemented against this FSM as follows: if an attacker can inject a fault and gain access to the Final Round directly from Initial Round without going through the Do Round state, then premature results will be stored, which significantly weakens the encryption strength of the AES algorithm and potentially leaks the secret key. Therefore, for this FSM, one security property can be defined as:

SP2.1: In the FSM of AES controller, Initial Round state cannot directly jump to Final Round state without going through Do Round state.

2) FSM in RSA Controller: RSA is a widely used asymmetric encryption algorithm. The FSM of the RSA controller module consists of seven states: 1) Idle; 2) Init; 3) Load1; 4) Load2; 5) Multiply; 6) Square; and 7) Result. Here, the attacker's objective would be to bypass the intermediate rounds of Square and Multiply states and access the "Result" state to obtain the premature result of RSA encryption. Therefore, for this FSM, one security property can be defined as:

SP2.2: In the FSM of RSA controller, Square and Multiply states cannot be bypassed to Result state.

3) FSM in SHA Controller: The SHA FSM is composed of 7 states: 1) Reset; 2) Data Input; 3) Padding; 4) Block Process; 5) Block Next; 6) Valid; and 7) Error. Each of these states controls specific operations in the SHA-256 digest engine. If an attacker can successfully inject a fault in the FSM to get access to specific states without going through the valid state transitions, it can compromise the security of the SHA-256 digest engine and make it vulnerable to hash collision and preimage attack [48]. Therefore, two security properties can be defined as:

SP2.3.1: In the FSM of SHA controller, each time when a block is loaded, the Data Input state should not be bypassed.

SP2.3.2: In the FSM of SHA controller, when the last block is loaded, the Block Process and/or Block Next state should not be bypassed.

Table X shows the information of the extracted fan-in circuit benchmark for different security properties. The AES benchmark used for SP2.1 differs from the one used for SP1. The only difference between AES FSM 1 and AES FSM 2 is from the different FSM encodings; same for RSA FSM 1 and RSA FSM 2.

TABLE X: Security property fan-in circuit information.

Benchmark	Security Property	Input	Output	N_S	N_C	N_T
AES FSM 1	2.1	3	3	7	24	31
AES FSM 2	2.1	3	3	7	20	27
RSA FSM 1	2.2	2	1	7	23	30
RSA FSM 2	2.2	2	1	7	21	28
SHA FSM	2.3.1/2.3.2	20	86	3	64	67
AES KS	3.1/3.2	130	128	140	2167	2307

Table XI shows the results of the fault simulation. As we can see, in the global fault simulation of AES FSM 1, there are no feasible faults, which indicates the FSM is resistant against setup-time violation-based fault-injection techniques. However, for AES FSM 2, there is 1 feasible fault. Similar results are achieved for the global fault simulation of RSA FSM 1 and RSA FSM 2 in which there is 0 and 1 feasible fault, respectively. Hence, inappropriate FSM encoding scheme can bring additional vulnerability to the design against fault-injection attacks. For the local fault simulation of both AES FSM 1 and AES FSM 2, 7 critical locations are identified, which indicates, by protecting these 7 critical locations in the AES, the security property would never be violated and the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3063998, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

threat from fault-injection attacks is significantly mitigated. For the local fault simulation of RSA FSM 1 and RSA FSM 2, there are 8 and 7 critical locations identified, respectively. For SP2.3.1 or SP2.3.2 in SHA FSM, there are 1 and 25 critical locations identified for global and local fault simulation, respectively. Therefore, for these four SPs in FSMs, less than 0.6% of the cells in AES/RSA/SHA are critical to fault-injection attacks. By protecting these critical locations, the threat from fault-injection attacks can be significantly mitigated and the protection efficiency could be much higher than conventional approaches in which the whole FSM or the whole AES/RSA/SHA is protected.

C. Security Properties against DFA

Differential fault analysis (DFA) is one of the most well known fault-injection attacks to compromise the secret key of cryptography devices [1]. By injecting faults at a specific location and time during the encryption and comparing the faulty and correct ciphertext, clues of the secret key can be deduced. Hence, the key space can be significantly reduced to make brute force attack practical. Different DFA attacks may require different faults in terms of size, location, and time. If the required faults for DFA can never be satisfied, the attack would not succeed.

In this subsection, the threat from three highly cited DFA attacks [49]–[51] on the key schedule (KS) of AES are evaluated. The corresponding security properties are defined to protect the design against these DFA attacks.

SP3.1: At the 9^{th} round of AES, any 1-3 bytes of the first word in the round key cannot be faulty and the faulty bytes cannot propagate to the following words in the same round.

SP3.2: At the 9^{th} round of AES, 4 bytes of any word in the round key cannot be faulty and the faulty bytes cannot propagate to the following words in the same round.

The AES benchmark used in this experiment is the same one used in SP1. The information of the extracted key schedule module in the AES is shown in Table X. 100 random key input vectors are applied as stimulus in the fault simulation. Considering the AES key schedule benchmark is much larger than the AES FSM and more stimulus are applied, only 1 concurrent fault location is considered in the fault simulation. Hence, the number of critical faults would be equal to the number of effective faults and the number of critical locations would be equal to the number of critical faults or feasible faults, if available.

The fault simulation results are shown in Table XI. As we can see, there are many more critical locations for SP3.1 than SP3.2, especially for the local faults assessment. It is because a single transient fault injected at any fan-in cell of the first word registers is very likely to result in 1-3 bytes fault in the first word of the round key, which is violating SP3.1. In addition, the total number of fan-in cells of the first word registers is ∼1,830. Therefore, it is easy to understand why there are 1,783 critical locations identified in the local faults assessment for SP3.1. Compared to SP3.1, the critical locations identified for SP3.2 are much fewer: 0 and 1 for global and local faults assessment, respectively. It is because SP3.2 requires all 4 bytes in a word to be faulty, which is difficult to be implemented by a fault with only one fault location. If a fault with multiple fault locations is injected, there might be more critical locations identified. However, the possibility to inject such fault at specific locations to violate SP3.2 might be much smaller than implementing a single location fault. Therefore, SP3.1 is more vulnerable to fault-injection attack, which indicates the DFA attack targeting a fewer bytes in the round key is more dangerous and requires more resources to defend. Except the local faults assessment for SP3.1, there are less than 0.1% of the cells in AES identified as critical locations to fault-injection attacks.

The last column shows the CPU run time of different fault simulations. As we can see, the CPU time is dependent to many factors, such as the benchmark size, the number of stimulus, the fault category, and the number of total faults. Generally, the larger benchmark, the more stimulus, and the

TABLE XI: Fault simulation results.

Benchmark	Security Property	Stimulus	Fault Category	# of Concurrent Fault Locations	Total Faults	Effective Faults	Critical Faults	Feasible Faults	Critical Locations	% in AES/ RSA/SHA	CPU Run Time (s)
AES FSM 1	2.1	1	Global	1-7	508	19	4	0	0	0.00%	1
			Local	1-4	145,824	7,173	113	NA	7	0.07%	82
AES FSM 2	2.1	1	Global	1-7	508	18	3	1	1	0.01%	1
			Local	1-4	83,412	6,516	63	NA	7	0.07%	44
RSA FSM 1	2.2	1	Global	1-7	381	18	3	0	0	0.00%	1
			Local	1-4	95,790	8,533	13	NA	8	0.01%	141
RSA FSM 2	2.2	1	Global	1-7	381	21	3	1	1	0.002%	1
			Local	1-4	72,471	5,104	14	NA	7	0.01%	137
SHA FSM	2.3.1	1	Global	1-3	42	2	2	0	0	0.00%	1
			Local	1-3	301,098	2,491	160	NA	12	0.28%	401
SHA FSM	2.3.2	1	Global	1-3	42	2	2	1	1	0.02%	1
			Local	1-3	301,098	17,508	85	NA	24	0.55%	411
SHA FSM	2.3.1 or 2.3.2	1	Global	1-3	84	4	4	1	1	0.02%	1
			Local	1-3	602,196	19,999	245	NA	25	0.58%	801
AES KS	3.1	100	Global	1	420	12	12	6	6	0.06%	432
			Local	1	6,921	1,783	1,783	NA	1,783	17.33%	5,855
AES KS	3.2	100	Global	1	420	0	0	0	0	0.00%	414
			Local	1	6,921	1	1	NA	1	0.01%	4,993

more faults require longer run time to execute the fault simulation.

V. Conclusions and Future Works

In this paper, for the first time to our knowledge, we bridge the gap between the need for automated security assessment tools against fault injection attacks and the capability of existing computer-aided design (CAD) tools commonly utilized by chip designers. We develop an automated framework for fault-injection vulnerability assessment at gate-level while targeting security properties using novel models and metrics. The fault models are characterized from specific fault-injection techniques and the fault simulation is performed with security properties taken into consideration so that the critical locations to fault-injection attacks are identified. Our experimental results from AES, RSA, and SHA show that for most security properties considered in the paper, by protecting less than 0.6% critical locations in the design, the threat from fault-injection attacks can be significantly mitigated.

In the future, we plan to expand the SoFI framework to the RTL level as well as the physical level and apply SoFI to larger SoC benchmarks with more security properties. In addition, local countermeasures, such as hardware, time, or information redundancy-based techniques, will be developed to protect the identified critical locations more efficiently with lower overhead. Further, the identification and mapping of security properties to gate-level netlist will be automated as much as possible with less manual work.

References

[1] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures," *Proceedings of the IEEE*, vol. 100, no. 11, pp. 3056–3076, 2012.

[2] S. Bhunia and M. Tehranipoor, *Hardware Security - A Hands on Learning Approach*, S. Bhunia and M. Tehranipoor, Eds. Morgan Kaufmann, 2019.

[3] M. Tehranipoor and C. Wang, *Introduction to Hardware Security and Trust*. Springer Publishing Company, Incorporated, 2011.

[4] M. Karpovsky and A. Taubin, "New class of nonlinear systematic error detecting codes," *IEEE Transactions on Information Theory*, vol. 50, no. 8, pp. 1818–1819, 2004.

[5] M. Hutter, J. Schmidt, and T. Plos, "Contact-based fault injections and power analysis on rfid tags," in *2009 European Conference on Circuit Theory and Design*, pp. 409–412, 2009.

[6] S. Govindavajhala and A. W. Appel, "Using memory errors to attack a virtual machine," in *2003 Symposium on Security and Privacy, 2003.*, pp. 154–165, 2003.

[7] N. Moro, A. Dehbaoui, K. Heydemann, B. Robisson, and E. Encrenaz, "Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller," in *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 77–88, 2013.

[8] D. F. Kune, J. Backes, S. S. Clark, D. Kramer, M. Reynolds, K. Fu, Y. Kim, and W. Xu, "Ghost talk: Mitigating emi signal injection attacks against analog sensors," in *2013 IEEE Symposium on Security and Privacy*, pp. 145–159, 2013.

[9] A. Avizienis, J. . Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[10] M. H. Tehranipour, N. Ahmed, and M. Nourani, "Testing soc interconnects for signal integrity using boundary scan," in *Proceedings. 21st VLSI Test Symposium, 2003.*, pp. 158–163, 2003.

[11] IBM, Ibm 4764 pci-x Cryptographic Coprocessor Specifications, http://www.ibm.com/security/cryptocards/pdfs/bs330.pdf.

[12] B. Ning and Q. Liu, "Modeling and efficiency analysis of clock glitch fault injection attack," in *2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pp. 13–18, 2018.

[13] N. Timmers and C. Mune, "Escalating privileges in linux using voltage fault injection," in *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pp. 1–8, 2017.

[14] M. Yilmaz, K. Chakrabarty, and M. Tehranipoor, "Interconnect-aware and layout-oriented test-pattern selection for small-delay defects," in *2008 IEEE International Test Conference*, pp. 1–10, 2008.

[15] J. G. J. van Woudenberg, M. F. Witteman, and F. Menarini, "Practical optical fault injection on secure microcontrollers," in *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 91–99, 2011.

[16] H. Wang, D. Forte, M. M. Tehranipoor, and Q. Shi, "Probing attacks on integrated circuits: Challenges and research opportunities," *IEEE Design Test*, vol. 34, no. 5, pp. 63–71, 2017.

[17] H. Wang, Q. Shi, D. Forte, and M. M. Tehranipoor, "Probing assessment framework and evaluation of antiprobing solutions," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 6, pp. 1239–1252, 2019.

[18] H. Wang, Q. Shi, A. Nahiyan, D. Forte, and M. M. Tehranipoor, "A physical design flow against front-side probing attacks by internal shielding," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2019.

[19] M. Yilmaz, K. Chakrabarty, and M. Tehranipoor, "Test-pattern selection for screening small-delay defects in very-deep submicrometer integrated circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 5, pp. 760–773, 2010.

[20] F. Amiel, C. Clavier, and M. Tunstall, "Fault analysis of dpa-resistant algorithms," in *Proceedings of the Third International Conference on Fault Diagnosis and Tolerance in Cryptography*, ser. FDTC'06, p. 223–236. Berlin, Heidelberg: Springer-Verlag, 2006.

[21] O. Kömmerling and M. G. Kuhn, "Design principles for tamper-resistant smartcard processors," in *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*, ser. WOST'99, p. 2. USA: USENIX Association, 1999.

[22] J.-J. Quisquater and D. Samyde, "Eddy current for magnetic analysis with active sensor," 01 2002.

[23] J.-M. Schmidt and M. Hutter, "Optical and em fault-attacks on crt-based rsa: Concrete results," in *Austrochip 2007, 15th Austrian Workhop on Microelectronics, 11 October 2007, Graz, Austria, Proceedings*, pp. 61–67. Verlag der Technischen Universität Graz, 2007.

[24] Jakub Breier, "Fault Injection Attacks and Countermeasures", Brno Security Meetings, FEKT VUT, Brno, Czech Republic, 28 March 2018.

[25] S. Skorobogatov, "Semi-invasive attacks-a new approach to hardware security analysis," 01 2005.

[26] M. Agoyan, J.-M. Dutertre, A.-P. Mirbaha, D. Naccache, A.-L. Ribotta, and A. Tria, "How to flip a bit?" in *Proceedings of the 2010 IEEE 16th International On-Line Testing Symposium*, ser. IOLTS '10, p. 235–239. USA: IEEE Computer Society, 2010.

[27] C. Helfmeier, C. Boit, and U. Kerst, "On charge sensors for fib attack detection," in *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*, pp. 128–133, 2012.

[28] I. Koren and C. M. Krishna, *Fault-tolerant systems*. Elsevier, 2010.

[29] R. E. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM Journal of Research and Development*, vol. 6, no. 2, pp. 200–209, 1962.

[30] A. Dominguez-Oviedo and M. A. Hasan, "Error detection and fault tolerance in ecsm using input randomization," *IEEE Transactions on Dependable and Secure Computing*, vol. 6, no. 3, pp. 175–187, 2009.

[31] G. Bertoni, L. Breveglieri, I. Koren, and P. Maistri, "An efficient hardware-based fault diagnosis scheme for aes: performances and cost," in *19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2004. DFT 2004. Proceedings.*, pp. 130–138, 2004.

[32] F. Zhang, S. Guo, X. Zhao, T. Wang, J. Yang, F. Standaert, and D. Gu, "A framework for the analysis and evaluation of algebraic fault attacks on lightweight block ciphers," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 5, pp. 1039–1054, 2016.

[33] P. Khanna, C. Rebeiro, and A. Hazra, "Xfc: A framework for exploitable fault characterization in block ciphers," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2017.

[34] J. Breier, X. Hou, and Y. Liu, "Fault attacks made easy: Differential fault analysis automation on assembly code," Cryptology ePrint Archive, Report 2017/829, 2017, https://eprint.iacr.org/2017/829.

[35] Saha, S., Mukhopadhyay, D., & Dasgupta, P. (2018). ExpFault: An Automated Framework for Exploitable Fault Characterization in Block Ciphers. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2018(2), 242-276.

[36] I. Roy, C. Rebeiro, A. Hazra, and S. Bhunia, "Safari: Automatic synthesis of fault-attack resistant block cipher implementations," *IEEE*

Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 4, pp. 752–765, 2020.
[37] V. Arribas, F. Wegener, A. Moradi, and S. Nikova, "Cryptographic fault diagnosis using verfi," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 229–240, 2020.
[38] M. Gay, T. Paxian, D. Upadhyaya, B. Becker, and I. Polian, "Hardware-oriented algebraic fault attack framework with multiple fault injection support," in *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pp. 25–32, 2019.
[39] M. Srivastava, P. SLPSK, I. Roy, C. Rebeiro, A. Hazra, and S. Bhunia, "Solomon: An automated framework for detecting fault attack vulnerabilities in hardware," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 310–313, 2020.
[40] K. Xiao, A. Nahiyan, and M. Tehranipoor, "Security rule checking in ic design," *Computer*, vol. 49, no. 8, pp. 54–61, 2016.
[41] N. Farzana, F. Rahman, M. Tehranipoor, and F. Farahmandi, "Soc security verification using property checking," in *2019 IEEE International Test Conference (ITC)*, pp. 1–10, 2019.
[42] AES IP Core, OpenCores, https://opencores.org/projects/aes_core.
[43] Z01X, Synopsys, https://www.synopsys.com/verification/simulation/z01x-functional-safety.html.
[44] International Technology Roadmap for Semiconductors, 2013 Edition, https://www.semiconductors.org/wp-content/uploads/2018/08/2013Overview-1.pdf.
[45] C. Boit, S. Tajik, P. Scholz, E. Amini, A. Beyreuther, H. Lohrke, and J. P. Seifert, "From ic debug to hardware security risk: The power of backside access and optical interaction," in *2016 IEEE 23rd International Symposium on the Physical and Failure Analysis of Integrated Circuits (IPFA)*, pp. 365–369, 2016.
[46] OpenCores, https://opencores.org/projects/.
[47] A. Nahiyan, F. Farahmandi, P. Mishra, D. Forte, and M. Tehranipoor, "Security-aware fsm design flow for identifying and mitigating vulnerabilities to fault attacks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 6, pp. 1003–1016, 2019.
[48] D. PATEL, *INFORMATION SECURITY: Theory and Practice*. PHI Learning, 2008. [Online]. Available: https://books.google.com/books?id=FFPzGN8Uk9cC
[49] C. H. Kim and J.-J. Quisquater, "New differential fault analysis on aes key schedule: Two faults are enough," in *Smart Card Research and Advanced Applications*, G. Grimaud and F.-X. Standaert, Eds., pp. 48–60. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.
[50] J. Takahashi, T. Fukunaga, and K. Yamakoshi, "Dfa mechanism on the aes key schedule," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007)*, pp. 62–74, 2007.
[51] D. Peacham, B. Thomas: A DFA attack against the AES key schedule, SiVenture White Paper 001, 2006.

Huanyu Wang (S'16) received the B.S. degree from Huazhong University of Science and Technology (HUST), Wuhan, China, in 2014, and the M.S. degree in Electrical Engineering from Northwestern University, Evanston, IL USA, in 2016.

He is currently working toward the Ph.D. degree in the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, USA. His current research interests include hardware security and trust, ASIC design, VLSI CAD, and clam chowder recipe.

Henian Li (S'20) received the B.S. degree in integrated circuits and integration system from Hefei University of Technology, Hefei, China, and joined University of Florida, Gainesville, FL, in 2019, where he is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, under the supervision of Prof. M. Tehranipoor. His current research interests include hardware security, fault-injection countermeasures and secure scan.

Fahim Rahman (S'13-M'19) is currently a Research Assistant Professor with the Electrical and Computer Engineering Department, University of Florida, Gainesville, FL, USA. He received his Ph.D. in electrical and computer engineering from the University of Florida, Gainesville, USA in 2018. He received his BS in electrical and electronic engineering from Bangladesh University of Engineering and Technology, Bangladesh and MS in electrical and computer engineering from the University of Connecticut, USA in 2015, respectively. His current research is in the domain of hardware and cybersecurity and trust including investigation of hardware security primitives, CAD for security and automatic assessment, FPGA security, electronic supply-chain security, and hardware-assisted cybersecurity. He published 3 book chapters and 15+ peer-reviewed papers. His research has been sponsored by SRC, AFOSR, AFRL, DARPA, Cisco, TI, and NIST. He is a member of IEEE and ACM.

Mark M. Tehranipoor (S'02-M'04-SM'07-F'18) is currently the Intel Charles E. Young Preeminence Endowed Chair Professor in Cybersecurity at the University of Florida. His current research projects include: hardware security and trust, supply chain security, IoT security, VLSI design, test and reliability. Dr. Tehranipoor has published over 500 journal articles and refereed conference papers and has delivered about 200 invited talks and keynote addresses. He has published 13 books. He is a recipient of a dozen best paper awards and nominations, as well as the 2008 *IEEE Computer Society (CS) Meritorious Service Award*, the 2012 *IEEE CS Outstanding Contribution*, the 2009 *NSF CAREER Award*, and the 2014 AFOSR MURI award. He received the 2020 University of Florida Innovation of the year award. He serves on the program committee of more than a dozen leading conferences and workshops. He has also served as *Program Chair* of a number of IEEE and ACM sponsored conferences and workshops (HOST, ITC, DFT, D3T, DBT, NATW, and more). He co-founded the IEEE International Symposium on Hardware-Oriented Security and Trust (HOST) and served as HOST-2008 and HOST-2009 *General Chair*. He is currently serving as a founding EIC for Journal on Hardware and Systems Security (HaSS) and *Associate Editor* for JETTA, JOLPE, IEEE TVLSI and ACM TODAES. Prior to joining UF, Dr. Tehranipoor served as the founding director for CHASE and CSI centers at the University of Connecticut. He is currently serving as a founding director for Florida Institute for Cybersecurity Research (FICS). Dr. Tehranipoor is a Fellow of the IEEE, a Golden Core Member of IEEE CS, and Member of ACM and ACM SIGDA.

Farimah Farahmandi (S'13-M'18) is an assistant professor in the Department of Electrical and Computer Engineering (ECE) at the University of Florida (UF). She received her Ph.D. from the Department of Computer and Information Science and Engineering (CISE) at the University of Florida, 2018. She received her B.Sc. and M.Sc from the Department of Computer Engineering at the University of Tehran, Tehran, Iran in 2010 and 2013, respectively. Her research interests include design automation of System-on-Chips and energy-efficient systems, formal verification, hardware security validation, and post-silicon validation and debug. Her research has been sponsored by SRC, DARPA, AFRL, DoD, Analog Devices, Ansys, and Cisco. Dr. Farahmandi currently is the associate director of Edaptive Computing Inc, Transition Center (ECI-TC) at the University of Florida.