Security-aware FSM Design Flow for Identifying and Mitigating Vulnerabilities to Fault Attacks

Adib Nahiyan*, *Student Member, IEEE,* Farimah Farahmandi*, *Student Member, IEEE,*
Prabhat Mishra, *Member, IEEE,* Domenic Forte, *Member, IEEE,* and Mark Tehranipoor, *Fellow Member, IEEE*

Abstract—**The security of a system-on-chip (SoC) can be compromised by exploiting the vulnerabilities of the finite state machines (FSMs) in the SoC controller modules through fault injection attacks. These vulnerabilities may be unintentionally introduced by traditional FSM design practices or by CAD tools during synthesis. In this paper, we first analyze how the vulnerabilities in an FSM can be exploited by fault injection attacks. Then, we propose a security-aware FSM design flow for ASIC designs to mitigate them and prevent fault attacks on FSM. Our proposed FSM design flow starts with a security-aware encoding scheme which makes the FSM resilient against fault attacks. However, the vulnerabilities introduced by the CAD tools cannot be addressed by encoding schemes alone. To analyze for such vulnerabilities, we develop a novel technique named Analyzing Vulnerabilities in FSM (AVFSM). If any vulnerability exists, we propose a secure FSM architecture to address the issue. In this paper, we mainly focus on setup-time violation based fault attacks which pose a serious threat on FSMs; though our proposed flow works for advanced laser-based fault attacks as well. We compare our proposed secure FSM design flow with traditional FSM design practices in terms of cost, performance, and security. We show that our FSM design flow ensures security while having a negligible impact on cost and performance.**

Index Terms—**FSM Integrity Analysis, Fault Injection Attacks, Security Design Rules, Secure FSM Architecture**

I. Introduction

Different hardware-based attacks, e.g., side channel attacks using power and timing analysis, exploitation of test and debug structures, and fault injection attacks have been demonstrated to compromise the security of a system-on-chip (SoC). These attacks can effectively bypass the security mechanisms built in the software level and put systems at risk. Among these attacks, fault injection poses a particularly serious threat. During fault attacks, an attacker injects faults to produce erroneous results and then analyzes these results to extract secret information from a SoC [1]. Over the past decade, fault injection attacks have grown from a crypto-engineering curiosity to a systemic adversarial technique [2]. However, most of the research on fault attacks are concentrated on analyzing the fault effects and developing countermeasures for fault injection on datapaths. Finite state machines (FSMs) in the control path are also susceptible to fault injection attacks, and the security of the overall SoC can be compromised if the FSMs controlling the SoC are successfully attacked. For example, it has been shown that the secret key of RSA encryption algorithm can be detected when FSM implementation of the Montgomery ladder algorithm is attacked using fault injection [3]. Therefore, it is also extremely important to understand how fault injection attack works in an FSM and develop proper countermeasures to protect against fault attacks.

Fault-tolerant FSM has been extensively studied for space-based applications [4], [1]. The fault model used for such applications is typically based on single fault event caused by radiation. Since these faults are random, they do not fit into threat model of fault injection attack caused by an intelligent attacker who can precisely inject faults to exploit FSM vulnerabilities. There are few works that try to identify and address vulnerabilities of FSM to fault injection attack. Authors in [3], [5], [6], [7] have proposed linear error detection techniques (e.g., triple modular redundancy, parity prediction, etc.) to protect the FSM from fault injection attacks. However, the proposed techniques suffer from large area overhead ($\backsim$ 200%) and assume specific error models that will not work for other adversarial models [8]. As alternatives to linear codes; nonlinear [9], multilinear [10] and multirobust [8] codes have been proposed to improve the protection of cryptographic devices against fault injection attacks. However, none of these methods address the vulnerabilities introduced by synthesis process, which makes them inadequate in protecting the FSM from fault injection attacks.

It has been shown that synthesis tools can introduce security risks in the implemented FSM by inserting additional don't-care states and transitions [11] [12]. Authors in [11] proposed architectural changes in the FSM to address the vulnerabilities introduced by don't-care states and transitions. Here, a modified T flip-flop based design which prevents normal states to access protected states (states which grant critical/secure functionality) was proposed. However, this solution fails to provide adequate protection against fault injection attack as it does not address the unauthorized state transition between protected states. Moreover, this architecture has adverse effects on timing, test coverage, and verification (e.g., equivalence checking) and therefore, is not suitable for industrial ASIC design flow.

In this paper, we propose a security-aware FSM design flow for ASIC designs to make the FSM fully secured against fault injection attacks. Our proposed flow introduces three additional steps into the current FSM design flow: (1) security-aware encoding of state assignments, (2) vulnerability analysis of synthesized FSM, and (3) security-aware architecture development for FSM design. An overview of our proposed security-aware FSM design flow is shown in Figure 1. Note that, the security-aware FSM architecture needs to be applied only when the security vulnerabilities cannot be addressed by

*Adib Nahiyan and Farimah Farahmandi contributed equally to this work.

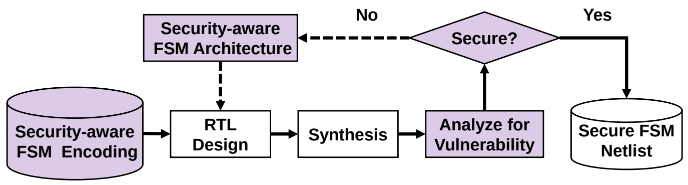
Security-aware FSM Architecture
No
Secure?
Yes
Security-aware FSM Encoding
RTL Design
Synthesis
Analyze for Vulnerability
Secure FSM Netlist

Fig. 1: Security-aware FSM design flow. The violet marked steps are the three additional steps introduced in the current FSM design flow by our approach. The dotted line represents that the security-aware FSM architecture needs to be applied only when the security vulnerabilities cannot be addressed by the security-aware encoding scheme.

the security-aware encoding scheme.

We make following major contributions:

- We investigate the traditional FSM design practices and demonstrate that these design practices (e.g., encoding schemes) can introduce vulnerabilities in the FSM. To the best of our knowledge, this is the first time that the security of different encoding schemes (binary, one-hot, and gray) have been examined and compared.
- We investigate the error detection based fault tolerant FSM architectures and demonstrate with experimental results that these approaches cannot provide adequate protection to FSMs against fault injection attacks.
- We propose two security-aware FSM encoding schemes; one based on securing protected states and the second based on securing protected transitions. The elegance of our proposed encoding schemes is that they inherently make the FSM more resistant to fault attacks without the need of any extra circuitry. This differentiates our proposed schemes from error detection-based approaches that require complex logic for detecting errors. We also validate with experimental results that most security vulnerabilities can be addressed by our proposed encoding schemes.
- We propose a technique, called Analyzing Vulnerabilities in FSM (AVFSM) to quantitatively analyze and evaluate how susceptible an FSM design is against fault injection attacks. To the best of our knowledge, this is the first systematic approach to analyze such vulnerabilities present in an FSM.
- We propose a security-aware FSM architecture to address the vulnerabilities introduced by the synthesis process (these vulnerabilities are detected using AVFSM approach) and to protect the FSM from advanced laser-based fault injection attacks.
- We demonstrate potential vulnerabilities to fault attacks in five controller benchmark circuits. We also compare our proposed security-aware FSM design flow with traditional FSM design practices in terms of security, cost, and performance.

The remainder of the paper is organized as follows. Section II provides a background on fault tolerant FSM design. Section III presents preliminaries and definitions that we use in this paper. Section IV shows how fault injection attack can be performed against FSMs using a motivational example. Section V presents our threat model. Section VI presents the vulnerabilities introduced by traditional and error detection code (EDC) based FSM design practices. Section VII discusses our proposed security-aware FSM design flow. Section VIII presents our results. Finally, Section IX concludes the paper.

II. Related Work

Sunar et al. [3] has proposed linear error detection techniques to protect the FSM from fault injection attacks. However, as described in Section VI-A, linear EDC based techniques do not take into account the non-uniform path delay distribution of a FSM or the vulnerabilities introduced by synthesis tools and therefore, are susceptible to fault injection attacks. We demonstrated with experimental results in Section VIII-D that these techniques are vulnerable to set-up time violation based fault injection attack.

Karpovsky et al. [13] proposed nonlinear error detection codes to minimize the fraction of undetectable errors. Contrary to linear codes, the error detection probability of nonlinear codes is dependent on state encoding. Therefore, if an adversary knows the states of FSM then, he/she can compute an undetectable error pattern and inject an undetectable fault. Thus, the security of this coding scheme mainly depends on the assumption that the attacker cannot observe the next state values of the FSM in the same clock cycle as fault injection [3]. However, this assumption is not valid as an attacker can make intelligent guess about the next state for a particular FSM. For example, in the AES controller FSM (see Section VIII-A), an attacker can guess the Initial Round' and Final Round' by observing when the encryption module loads the input plaintext and when the output ciphertext in available.

Kahraman et al. [14] proposed to address the limitations of applying non-linear codes in FSM by embedding randomization to achieve unpredictability and uniformity. Although it can provide some guarantee to detect fault injection attacks, the following challenges exist in their approach,

- It may require impractical area and delay overhead for many applications (see Section VIII-E).
- It requires the implementation of a tamper-proof clock and random number generator [15]. These requirements are quite challenging and expensive to meet when considering the fact that the attacker has physical access to the device.
- It essentially push the security threat from the FSMs to the decoder circuit which generates control signals based on the current state and requires visibility of the state. Therefore, the bare form of the state and output control signals become vulnerable points for possible attacks [16].

III. Preliminaries and Definitions

An FSM is formally defined as a 5-tuple $(S, I, O, \varphi, \lambda)$, where S is a finite set of states, I is a finite set of input symbols, O is a finite set of output symbols, $\varphi : S \times I \rightarrow S$ is the next-state function and $\lambda : S \times I \rightarrow O$ is the output function.

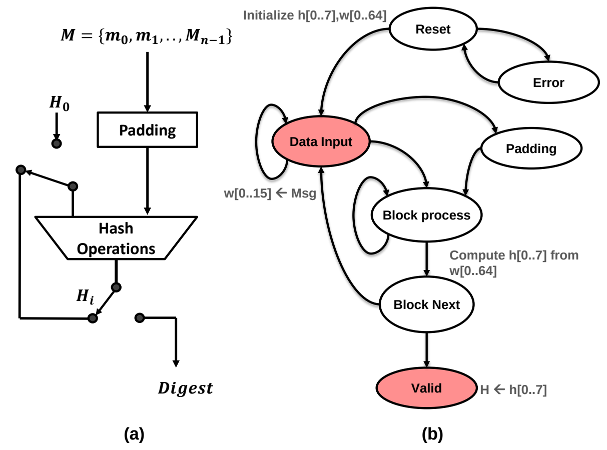

Fig. 2: (a) High-level diagram of SHA-256, (b) FSM of SHA-256 digest engine. The red marked states represent protected states of SHA-256 FSM.

For convenience, an FSM is typically represented as a directed graph where each vertex represents a state $s \in S$ and an edge represents the transition $t = T(s_i, s_j)$ from current state s_i to its next state s_j. This graph is referred to as a $state\ transition\ graph\ (STG)$. In the STG each state can be accessed from a set of states which we define as the *accessible set of states*,

$$A(x) = \{s_j \mid s_j\ is\ accessible\ from\ s_i\} \tag{1}$$

In this paper, we define two more sets, P and L, which are both specified by the designer. P is the set of protected states and L is a set of authorized states that are allowed to access a protected state p, that is $A(L) = \{p \mid p \in P\}$. *If any state p is accessed by any state apart from states in L, then the security of the FSM can be compromised.*

In the behavioral specification of the FSM, there are don't-care conditions where the next state or the output of a transition are not specified. During the synthesis process, the synthesis tool tries to optimize the design by introducing deterministic states and transitions for the don't-care conditions. Let us consider the FSM F' implemented by the synthesis tool from the behavioral description of the FSM F. Let, S and S' represent the set of states and T and T' represent the set of transitions in F and F', respectively. The set of don't-care states and transitions (S_D and T_D) introduced by the synthesis process are defined as follows,

$$\begin{aligned} S_D &= \{s' \mid (\ s' \in S') \cap (s' \notin S)\} \\ T_D &= \{t' \mid (t' \in T') \cap (t' \notin T)\} \end{aligned} \tag{2}$$

Table I presents the symbols and notations that we use throughout this paper.

IV. Motivating Example

In this section, we explore the possible fault injection attacks against an FSM and show how these attacks can compromise the security of the overall system. We use the controller circuit of a SHA-256 digest engine [17] as an example to demonstrate the feasibility and effectiveness of fault attacks. The FSM in the controller circuit of SHA-256 digest engine is shown in Fig. 2(b). The FSM is composed of 7 states: 'Reset', 'Data Input', 'Padding', 'Block Process', 'Block Next', 'Valid', and 'Error'. Each of these states controls specific operations in the SHA-256 digest engine. The digest algorithm operates on two registers, $w[0..64]$ which is responsible for loading the message and $h[0..7]$ which stores the intermediate digest results. These two registers are initialized during 'Reset' state. The final digest (H) will be latched into the result register in 'Valid' state. In our SHA-256 FSM example, 'Valid' is a protected state and 'Block Next' is the authorized state to access the protected state 'Valid'.

TABLE I: Symbols and notations

Symbols	Definitions	Symbols	Definitions
P	set of protected states	VT	vulnerable transitions
L	set of authorized states	$Path_{FS}(i)$	maximum path delay of ith state FF
S_D	set of don't-care states	SF	susceptibility factor
T_D	set of don't-care transitions	PVT	percentage of vulnerable transitions
FF_S	state flip-flop (FF)	ASF	average susceptibility factor
S_{EN}	state encoding	VF_{FI}	vulnerability factor of fault injection

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. We will demonstrate two such attacks in the following,

- During the 'Data Input' state, the message M is loaded to register $w[0..15]$. These values are used in a subsequent operation to compute $w[16..63]$, $h[0..7]$, and digest H. Therefore, if an attacker can inject fault to bypass 'Data Input' state, then M will not be loaded into register $w[0..15]$. Therefore, H will be computed from the initialized value of $w[0..15]$. The outcome of this attack would be the same hash value as computed for any message. This attack would compromise the collision resistance property [18] of the digest engine i.e., for any two different messages m_1 and m_2, their hash value would be same, $hash(m_1) = hash(m_2)$. A digital signature algorithm based on hash functions are particularly vulnerable to hash collision attacks. In [19], authors have used a hash collision attack to produce a rogue certificate authority.
- When the hash value of the last message block is calculated, the FSM moves to 'Valid' state and the digest value H is captured by the result registers. Now, if an attacker can inject fault to bypass 'Block Process' and/or 'Block Next' states, then instead of H, initial values of $h[0..7]$ would be captured by the result registers and would be published as the digest value. This attack would compromise both the pre-image resistance [18] and the collision resistance property of the digest engine. A successful execution of this attack would allow the adversary to bypass the authentication mechanism provided by the digest engine.

V. Threat Model

In this section, we identify how the vulnerabilities are introduced in the FSM and how these vulnerabilities can be exploited to compromise the security of a SoC. We also briefly describe the potential adversaries, their objectives, and their capabilities. In our threat model, we have considered fault injection attacks which create multiple bit flips and also considered fault attacks performed by a focused laser beam.

Sources of Vulnerabilities: Most security vulnerabilities in an FSM are unintentionally created by designer mistakes or by CAD tools. Traditional FSM design practices are driven by cost and performance while security is largely ignored. For example, FSMs are generally encoded in binary, gray or one-hot from the performance perspective. In [12], it was shown that certain encoding schemes are more susceptible to fault injection attacks. Further, CAD tools can create additional vulnerabilities in an FSM. For example, in the RTL specification of an FSM, there are don't-care conditions where the next state or the output of a state is not specified. Synthesis tools optimize the FSM design by introducing deterministic states and transitions for don't-care conditions. This can create a vulnerability in the circuit by allowing a protected state to be illegally accessed through the don't-care states (S_D) and transitions (T_D).

Fault Injection Attack on FSM: In this paper, we focus on setup-time violation-based fault attack. This attack violates the setup time constraint of state flip-flops (FFs) to bypass a normal state transition and enter a protected state. Setup time violations can be performed by different fault injection methods, including overclocking, reducing the voltage, and/or heating the device [20]. These types of attacks pose the most serious threat to an FSM as they require relatively low-cost equipment and do not necessarily need the complete knowledge of the FSM design.

We illustrate the working principle of setup-time violation-based fault attack using the example of the FSM of SHA-256 digest engine (shown in Fig. 2). Fig. 3(a) shows the distribution of path delays located in the fan-in cone of the state FFs of SHA-256 FSM under normal operating condition. The path delays were extracted using the static timing analysis from the gate-level implementation of the FSM (using 90nm standard cell library). The delay distribution is shown in the form of box-plot. It is clear from Fig. 3(a) that the path delay distribution to the state FFs is non-uniform, meaning that it is possible to inject fault that violates setup-time of certain FFs (ones with longer path delay) while maintaining setup-time of other FFs (ones with shorter path delay). Such attacks are termed as biased fault attack [21]. For example, when the FSM is operating with the clock period (T_{clk}) of $T_{clk} > 2.25ns$, no setup-time violation occurs in any state FF. However, the setup-time of state FF1 can be violated by operating the FSM with $2.05ns < T_{clk} < 2.25ns$ (1-bit fault region shown by green marked area in Fig. 3(a)). If T_{clk} is further reduced ($1.70ns < T_{clk} < 2.05ns$), then setup-time of both state FF1 and FF0 will be violated (2-bit fault region shown by purple marked area in Fig. 3(a)).

In some cases, e.g., in embedded systems, an attacker may not have control over the clock signal. In such cases, the attacker can reduce the supply voltage and/or increase the temperature of the circuit to perform setup-time violation-based fault attack. Fig. 3(b) shows the distribution of path delays located in the fan-in cone of the state FFs of SHA-256 FSM for low voltage corner condition. If we consider the operating clock period to be $2.5ns$, then under low voltage condition the setup-time of both state FF1 and FF0 will be violated. This is shown in 2-bit fault region shown by purple marked area in Fig. 3(b). Therefore, biased fault attack can also be performed by reducing the supply voltage and/or increasing the operating temperature.

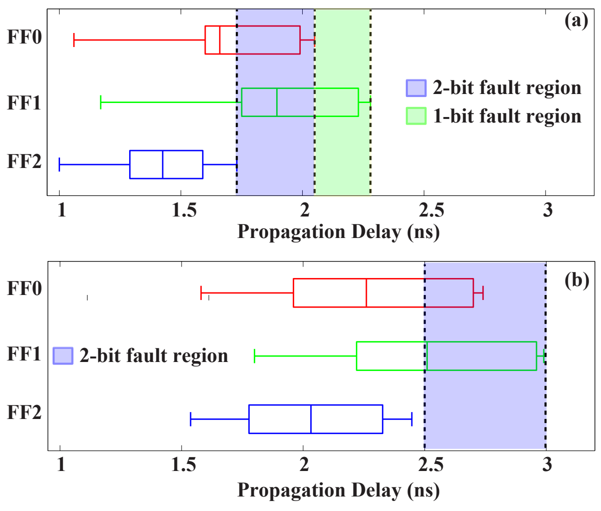

Fig. 3: (a) The distribution of path delays located in the fan-in cone of the state FFs of SHA-256 FSM under normal operating condition. (b) The distribution of path delays located in the fan-in cone of the state FFs of SHA-256 FSM for low voltage corner condition. The green and purple marked areas represent the 1-bit and 2-bit fault regions, respectively.

In summary, the non-uniform path delay distribution of an FSM enables an attacker to violate setup-time of certain FFs while maintaining setup-time of other FFs. Thereby allowing the attacker to bypass normal state transitions and get access to a protected state directly. Note that, it is also not feasible to equalize the non-uniform path delay distribution of an FSM using the limited sizes of buffer offered by a standard cell library.

It is also possible to induce faults to flip bits in sections of a circuit with a precisely focused laser beam. Although such fault injection techniques require expensive equipment as well as knowledge of the FSM design, our threat model takes such attacks also into consideration. Note that our threat model does not consider the fault attack scenario where an attacker can individually set or reset a single FF without affecting any other gates. Such, attacks are only possible through focused ion beam (FIB) and those are out of scope of this work. The reason is that, an attacker with FIB capability would directly probe the design to extract secret information rather than inducing fault to cause information leakage.

Potential Adversaries: For this threat model, we assume the adversary has physical access to the device. The adversary can thereby manipulate the clock signal, supply voltage, or operating temperature of the FSM. This model is realistic for stolen devices and systems or in the case of smart cards where clock and voltage may be supplied by a malicious reader. Based on the capabilities of the adversaries, we divide them into two adversarial categories,

TABLE II: Different encoding scheme for SHA-256 FSM.

States	**Reset**	**Data Input**	**Block Process**	**Block Next**	**Padding**	**Valid**	**Error**
Binary	000	001	010	011	100	101	110
One-hot	0000001	0000010	0000100	0001000	0010000	0100000	1000000
Hamming (7,4)	1110000	1101001	0101010	1000011	1001100	0100101	0001111

- Strong adversarial model: Here, the attacker knows the state encoding and functionality of the FSM and therefore, can mount precise glitching attacks [21]. The attacker can obtain this information from an insider, by espionage, or by reverse engineering. These adversaries can also mount precise laser-based fault attack. Although this class of attacks is less likely, they pose a greater threat as they allow one to precisely inject fault to gain access to a protected state.
- Weak adversarial model: In this case, the attacker does not know the state encoding nor the functionality of the FSM. Here, the attacker will try to inject random faults in the FSM in the hope of gaining access to a protected state. Although these attacks pose a less serious threat, they are more likely to be performed.

VI. Limitations in Existing FSM Design Flow

In this section, we describe how vulnerabilities are introduced by traditional FSM encoding schemes. We also present the limitations of error detection code (EDC) based approaches in protecting the FSM from fault attack.

A. *Traditional FSM Encoding Practices*

Encoding the state assignments in FSMs is traditionally done based on design constraints such as area, power, and delay. Designers generally choose between binary, one-hot, and gray encoding to encode the FSMs. None of these encoding schemes take security into consideration and thereby, can introduce vulnerabilities in the FSM. In this section, we compare traditional encoding styles from security and performance points of view.

Binary Encoding: In binary encoding scheme, states are encoded as a binary sequence where the states are numbered starting from 0 and up. The number of state flip-flops (FF_S), q, required for binary encoding scheme is given by $q = log_2(n)$; where, n is the number of states. From this equation, it is evident that binary encoding scheme requires minimum number of state FFs. Binary encoding scheme ensures maximum utilization of state FFs, but it requires complex combinational logic for decoding each state. Therefore, binary encoding scheme is better suited for FSM with a fewer number of states [22]. However, in terms of security, the binary encoding scheme makes the FSM more susceptible to fault injection attack. We demonstrate this vulnerability using the FSM of the SHA-256 digest engine.

The binary encoding for SHA-256 FSM is shown in Table II. One possible attack is shown in Fig. 4(a). During state transition from 'Padding' (100) to 'Block Process' (010), it is possible to inject fault and bump into protected state 'Valid' (110). To successfully inject this fault, the attacker needs to violate setup time of state FF2 (the leftmost bit, representing

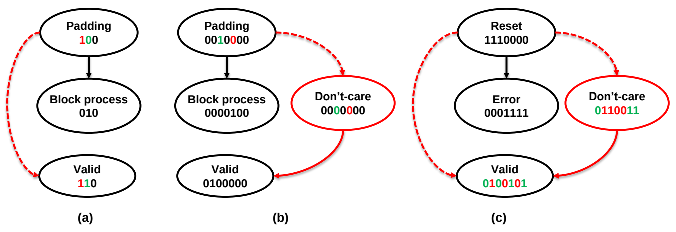

Fig. 4: Fault injection attack on the FSM of SHA-256 digest engine. Parts (a), (b) and (c) illustrate the fault injection attack on FSM encoded in binary, one-hot and Hamming code (7, 4), respectively. The red marked bits represent the state FFs for which setup time needs to be violated and the green marked bits represent the state FFs for which setup time needs to be maintained.

FF2, is marked in red in Fig. 4(a)) while maintaining setup-time of state FF1 (the middle bit, representing FF1, is marked in green in Fig. 4(a)).

One-hot Encoding: In one-hot encoding, only one bit of the state variable is '1' while all other state bits are zero. One-hot encoding requires as many state FFs as the number of states, and therefore, one-hot encoding requires more state FFs than binary. However, one-hot encoding possesses simpler combinational logic for decoding each state and, therefore, is more suitable for FSMs with more states. From the security perspective, it is also inherently less vulnerable to fault attacks. We will demonstrate this using the FSM of the SHA-256 digest engine. The one-hot encoding for SHA-256 FSM is shown in Table II. During state transition from 'Padding' (0010000) to 'Block Process' (0000100), it is not possible to inject faults and bump into protected state 'Valid' (0100000). The reason is that in order to go to 'Valid' state, the state FF(6) needs to be '1'; however, during the other state transitions, this bit remains unchanged as this bit is dedicated for 'Valid' state. Therefore setup-time violation based fault attack cannot be applied to change state FF(6) bit to '1'. This property makes the one-hot encoding inherently more secure against fault attacks.

Note that, one-hot encoding could result in many don't-care states. If any of these don't-care states has access to a protected state as a result of synthesis, then there will be a vulnerability in the FSM. For example, as shown in Fig. 4(b), the don't-care state '0000000' has access to 'Valid' state. Therefore, during state transition from 'Padding' to 'Block Process', an attacker can inject a fault to bump into the don't-care state and then access the 'Valid' state. Hence, there can still be vulnerabilities even in one-hot encoded FSMs.

Note that gray encoding is also used in traditional FSM design. Gray encoding has similar security characteristics to binary encoding scheme and therefore, we did not elaborate on gray encoding scheme.

B. *EDC Based Approaches*

Previously proposed fault tolerant FSM architectures [3], [5] are based on linear error detection code (EDC). The basic idea of these approaches is to encode the FSM's state using error detection codes and then detect any fault by checking the parity bits. Linear error detection codes are denoted as

$[n, k, d]$ where n is the code length, k is the message length and d is the minimum Hamming distance between any two code words. A linear error detection code can detect up to $d-1$ errors. Authors in [3], [5] assumed that the EDC based approach would force the attacker to be very precise when changing certain bits; otherwise the attack would be detected. Therefore, flipping multiple targeted bits without changing other bits would be a daunting task.

The main limitations of EDC based approaches are: (i) these approaches do not take into account biased fault model which allows an attacker to exploit the non-uniform path delay distribution of an FSM to inject multiple bit flips, (ii) none of the EDC based approaches address the vulnerabilities introduced by the synthesis tool. We illustrate these limitations by applying an EDC scheme on our SHA-256 FSM. Here, we use the Hamming (7,4) error detection approach [23], though any EDC based approach will share the same limitations.

The Hamming (7,4) encoding for SHA-256 FSM is shown in Table II. Note that, there are 16 valid states in Hamming (7,4) code; however, only 7 are used to encode SHA-256 FSM. Therefore, there are 9 valid don't-care states which may pose a threat as these states can bypass the error detection mechanism. We illustrate the limitation of Hamming (7,4) code using the following two examples. During state transition from 'Reset' (1110000) to 'Error' (0001111), it is possible to inject fault and bump into protected state 'Valid' (10100101). To successfully inject this fault, the attacker needs to violate setup time of state FF(1), FF(3) and FF(5) (marked in red in Fig. 4(c)) while maintaining setup-time of rest of the state FFs (marked in green in Fig. 4(c)). Also, during this transition, one can inject a fault to bump into the valid don't-care state '0110011' and, if this don't-care state has access to 'Valid' state, then it is a security vulnerability. The above examples illustrate that the EDC based approaches cannot provide adequate protection to FSMs against fault injection attacks.

VII. Security-aware FSM Design Flow

An overview of our proposed security-aware FSM design flow to protect the FSM is shown in Figure 1. Our proposed flow starts with designing the FSM RTL with security-aware encoding. We propose two encoding schemes which make the FSM inherently fault tolerant with little or no additional cost and performance overhead. However, these encoding schemes cannot address the vulnerabilities introduced by the synthesis process and cannot provide protection against advanced laser-based fault attacks. We analyze the synthesized FSM netlist for potential vulnerabilities using our proposed AVFSM technique. If any vulnerabilities exist or protection against laser-based fault attacks is required, we propose a security-aware FSM architecture to mitigate them. Our proposed FSM architecture ensures that protected states are only accessed from authorized states and they cannot be accessed via unauthorized states and don't-care states. Therefore, our proposed FSM architecture provides protection against laser-based fault attacks and also addresses the vulnerabilities introduced by CAD tools.

Algorithm 1 Secure Encoding - Scheme I

```
1: procedure
2: Input: Protected States $\mathbb{P}$, Normal States $\mathbb{N}$, Initial State $I$
3: Output: FSM Encoding Map, $S_{EN}$
4:     $P \leftarrow |\mathbb{P}|$, $N \leftarrow |\mathbb{N}|$
5:     l = $P + log(N)$                                   // Encoding Bit Length
6:     for $P_i \in \mathbb{P}$ do
7:         $E_i = OneHotEncoding(i, P)||(00..0)_{l-p}$    // concat (l-p) zeros with one-hot encoding
8:         $S_{EN}.add(P_i, E_i)$
9:     end for
10:    for $N_i \in \mathbb{N}$ do
11:        $E_i = binaryEncoding(i, N)||(00..0)_{l-N)}$
12:        $S_{EN}.add(N_i, E_i)$
13:    end for
14:    $S_{EN}.add(I, (00..0)_l)$                          // initial state
15:    return $S_{EN}$
16: end procedure
```

A. Security-aware FSM Encoding

In this section, we present two security-aware FSM encoding techniques. The first approach (Scheme I) is based on a conservative model where we make the protected states resilient to fault attack during all transition. However, fault injection during certain transitions may not create any security issues. In our second approach, we make the protected states resilient to fault attack during state transitions which cause security concerns.

Scheme I: One-hot encoding is more resilient to fault injection attacks in comparison with other encoding styles as discussed in Section VI-A. Our first encoding scheme exploits the benefits of one-hot style while reducing the number of don't-care states. Algorithm 1 shows the proposed encoding. The algorithm takes as input from the designer, the states specified as three different categories: the initial state, normal states, and protected states. Our primary goal is to make the protected states more resilient against fault attacks. Therefore, the algorithm uses one-hot scheme for protected states while it uses binary scheme for normal states. If the FSM contains one initial state, N normal states and P protected states, the algorithm uses $log(N) + P$ bits for encoding (line 5). The algorithm dedicates P upper bits to one-hot scheme while it pads zero for the rest of $log(N)$ bits in order to encode a protected state (lines 7-9). To encode a normal state, the algorithm pads zero for the N upper bits and uses binary encoding for $log(N)$ lower bits (lines 10-12). It always encodes the initial state with all zeros (line 13). This encoding approach decreases the number of don't-care states (as compared to one-hot) while making sure that it will be impossible for an attacker to access to a protected state from a normal state with fault attacks since during normal state transitions, P upper bits are fixed to zeros.

Example 1: The FSM shown in Figure 2(b) can be encoded with Algorithm 1 as following: Reset="00000", Block Process="00001", Block Next="00010", Padding="00011", Error="00100", Data Input="01000", Valid="10000".

Scheme II: Note that, every access to a protected state from an unauthorized state does not necessarily introduce a security threat based on the attack objective. For example, it can be observed from Figure 2(b) that an unauthorized access to *Data Input* state from *Block Process* may not be a security threat if the attacker's objective is to bypass the digest operation. In other words, an FSM may be secured against fault attacks if the state encoding provides protection for only prohibited

Algorithm 2 Secure Encoding - Scheme II

```
1:  procedure
2:  Input: State Names $\mathbb{S}$, Initial State $I$, Prohibited Transitions $\mathbb{T}$
3:  Output: FSM Encoding Map, $S_{EN}$
4:      for $log(N) \leq l \leq N-1$ do
5:          for all of possible combinations do
6:              $S_{EN} = \{\}$
7:              $S_{EN}.add(I, (00..0)_l)$                    // initial state
8:              $S_{EN} = findEncoding(\mathbb{S}, l)$   // random encoding with length $l$
9:              for $T_i \in \mathbb{T}$ do
10:                 m=generateMask($S_{EN}.get(src(T_i)), S_{EN}.get(dest(T_i))$)
11:                 for prohibited states $t_i$ of $T_i$ do
12:                     checkForConflicts($S_{EN}.get(t_i)$, m);
13:                 end for
14:             end for
15:             if (There is no conflict) then
16:                 return $S_{EN}$
17:             end if
18:         end for
19:     end for
20: end procedure
```

transitions instead of every transition to the protected states. This property enables us to introduce another FSM encoding scheme which is similar to binary scheme, but also tries to reduce the number of don't-care states that exist in the previously proposed encoding (Algorithm 1).

Algorithm 2 shows the second proposed encoding approach. The algorithm takes an initial state, state names, and a list of the prohibited transitions as inputs and generates an optimal length encoding as output. A list of prohibited transitions includes state(s) that should be prohibited during a transition from state u to v using fault attacks. Moreover, it can contain information about which transitions should not be bypassed. If there are n states, the algorithm searches different encoding lengths (l) where $log(n) \leq l \leq n-1$ and tries almost all of the combinations to find a secure encoding (lines 4-8). The goal is to find an encoding that does not have any conflict with the list of prohibited transitions. The initial state is encoded with all zeros. To check whether an attacker can inject a fault during a transition from state u to v and gain access to state t, a mask is generated from the temporary encodings of states u and v to identify which bits have changed during this transition (line 10). The changed bits are marked with 'x' and the fixed bits are kept as they are in the generated mask (e.g., "0101" $\rightarrow$ "1001" $: mask =$ "$xx01$"). The encoding of state t is compared with the generated mask. If the encoding has one-bit difference from the fixed bits of the mask, the temporary assignment is safe (since reaching to t requires changes in the fixed bits of transition $u \rightarrow v$). Otherwise, the assigned encodings are not safe and another combination should be tried (lines 11-12). The algorithm returns an encoding as a result when there is no conflict with the list of prohibited transitions (lines 13-14). Note that we also employ some heuristics to efficiently reduce the computation cost of the algorithm (e.g., using one-hot scheme in l bits and assign it to l states to limit the search space). If there is an optimal encoding, this algorithm will find it. In the worst case, it uses one-hot scheme for all of the states except the initial state like the previous approach. However, this approach requires more inputs from the designer.

Example 2: Using Algorithm 2, the FSM shown in Figure 2 can be encoded as: Reset="0000", Block Process="1000", Block Next="0100", Padding="0010", Error="0111", Data Input="0001", Valid="1110".

Algorithm 3 State Transition Graph Extraction

```
1:  procedure
2:  Input: Gate-level netlist of the FSM, FSM synthesis report
3:  Output: $STG$ Modified netlist for ATPG-based FSM extraction
4:      $FF_S \leftarrow$ Identify state FFs
5:      $S_{EN} \leftarrow$ Get state encoding
6:      $N_M \leftarrow$ Produce the modified netlist
7:      for each $s \in S_{EN}$ do
8:          Apply the logical value of s as constraint on $PI_{XOR}$
9:          Remove all faults and add stuck-at-1 fault at $PO_{OR}$
10:         Generate ATPG patterns n times
11:         Extract the present state values and conditions
12:     end for
13:     end for
14: end procedure
15: end procedure
```

B. AVFSM: FSM Vulnerability Analysis

Our proposed security-aware FSM encoding schemes inherently make the FSM more resistant to fault attacks without the need of any extra circuitry. However, our proposed encoding schemes do not address the vulnerabilities introduced by the synthesis process (after state encoding assignments). For example, synthesis tools can introduce don't-care states that have access to the protected states (as shown in Fig. 4(b)), allowing the attacker to access the protected states via these don't-care states. Also, an attacker can use a precise laser beam to induce bit flips causing the FSM to go to a protected state from an unauthorized state. This type of advanced fault injection attacks cannot be mitigated through our encoding schemes. In order to check whether there is any FSM security concern introduced after synthesis, we propose an FSM Vulnerability Analysis Framework (AVFSM).

Our proposed AVFSM takes as input (i) gate-level netlist of the design, (ii) FSM synthesis report, and (iii) user given inputs. AVFSM then extracts the STG of the FSM from the gate-level netlist and reports a quantitative measure of the vulnerability of FSM to fault injection attack. These procedures are discussed in details in the following subsections.

Extraction of STG: To analyze vulnerabilities in the FSM, we first need to extract the STG from the synthesized gate-level netlist. The extracted STG must incorporate the don't-care states and transitions which were introduced by the synthesis process. Existing work in literature only focuses on FSM reverse engineering from gate-level netlist [24], [25]. However, none of these techniques can extract the STG with the don't-care states and transitions.

One straightforward approach would be to perform a functional simulation of the FSM with all possible input patterns and produce the STG. However, this technique also cannot extract the don't-care states and transitions as these don't-care states cannot be accessed under the normal operating conditions of the FSM. It is because the synthesis tool introduces these don't-care states in such a way that these states cannot be accessed from the normal states (states mentioned in the RTL code); otherwise the original functionality of the FSM will be altered.

We propose an automatic test pattern generation (ATPG) based FSM extraction technique which can produce the STG with the don't-care state and transitions from the synthesized netlist. Our proposed extraction technique takes the gate-level netlist and the FSM synthesis report as inputs, and automatically generates the STG. Here, our assumption is that this vulnerability analysis will be performed by the designer, who has access to the RTL code, gate-level netlist, synthesis report and therefore has knowledge of the FSM's functionality.

The algorithm for STG extraction is shown in Algorithm 3. It first identifies the state flip-flops (FF_S) using the FSM synthesis report generated by the synthesis tool (procedure I, line 4). In this work, we use the Synopsys *dc_shell's report_fsm* [26] command to generate the report. The report contains names of the state registers (FF_S) and the state encoding information (S_{EN}). The naming of the registers is conserved during the synthesis process and we can identify the FF_S using the FSM synthesis report.

After identifying the FF_S, our algorithm searches if there are any non-state FFs (FF_{NS}) present in the input cone of the FF_S (see Fig. 5(a)). These non-state FFs are typically counters and they influence the state transitions in the FSM. We need to determine the logic values of the FF_{NS} which cause a transition in the STG.

Next, Algorithm 3 generates the modified netlist for the ATPG analysis. The modified netlist is shown in Fig. 5(b) and the original FSM is shown in Fig. 5(a). In the modified netlist, the output nets of the FF_S (defines the present state) and the FF_{NS} (defines conditions for state transition) are connected as primary inputs, PI_{PS}, and PI_{FFNS}. Also, XOR gates are placed at each input net of FF_S, and the other input of the XOR gate is connected as primary input, PI_{XOR}. The output pins of the XOR gates are ORed together, and the output pin of the OR gate is added as primary output, PO_{OR}. This modified netlist will be used to generate STG of the FSM.

Finally, Algorithm 3 determines the present states and input conditions that cause a transition to a particular state $s \in S_{EN}$. The basic idea is to first apply the logical values of s as constraints on PI_{XOR} and generate test patterns for stuck-at-1 fault at PO_{OR} (line 8-10). To generate patterns for this fault, the ATPG tool must produce 0 at PO_{OR} which requires the logic values at the input of the XOR gates to match with the constraints (s) applied on PI_{XOR}. In other words, the ATPG tool will generate the logic values of present states (PI_{PS}) and input conditions (i.e., input pins of the FSM and PI_{FFNS}) which cause transitions to state s. We generate the test patterns n number of times using *Tetramax's n-detect* [26] option to get all possible present states and input conditions which cause a transition to s. Although this option does not guarantee the generation of all possible patterns for a specific fault, in our experiments we have verified that by specifying a reasonably large value of n, we can extract the whole STG.

Vulnerability Analysis of Fault Attacks: In this section, we use the extracted STG to analyze how susceptible the FSM is to fault injection attacks. In our analysis, we consider the weak adversarial model (see section V) where faults are injected by violating the setup timing constraints using overclocking, voltage starving, and/or heating the device [27]. These types of attacks require low-cost equipment and pose the most serious threat [28].

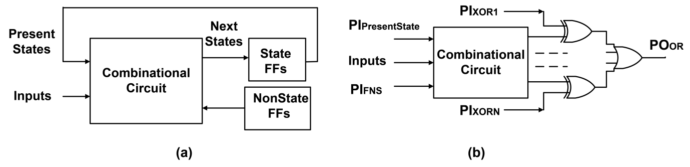

Fig. 5: (a) Original FSM (b) Modified FSM for ATPG-based STG extraction.

Estimating the vulnerability of hardware cryptosystems against timing violation attacks have been recently proposed in [20]. However, their proposed technique can only be applied to the data path and not to the FSM. Unlike data path, the FSM presents some unique challenges in vulnerability analysis of fault injection attacks (e.g., existence of don't-care states and transitions). Here, we propose a technique which analyzes each transition of the STG and based on a proposed metric quantitatively measures how susceptible that transition is to a fault injection attack. Based on the result, AVFSM will automatically report overall vulnerability measures of the FSM to fault attacks.

Our vulnerability analysis is based on the observation shown in Fig. 6. Let us consider the state transition $T(00, 10)$ where the current state is $'00'$ and the next state is $'10'$. During this transition, one cannot perform time violation based fault injection attacks to go to state $'01'$ (see Fig. 6(a)). It is because during $T(00, 10)$, the LSB bit of both the current state and the next state remains 0 and therefore, a setup time violation based fault cannot be injected at this bit position to change the bit value to 1. On the other hand during $T(10, 01)$, one can inject a fault to go to state $'11'$ (see Fig. 6(b)). To successfully inject this fault, the setup time constraint of MSB state FF needs to be violated whereas the setup time constraint of LSB state FF needs to be maintained. In other words, delay of the logic path of MSB state FF needs to be greater than the delay of the logic path of LSB state FF. We formulate these conditions by C and SF_T as shown in Algorithm 4.

To perform the fault vulnerability analysis, Algorithm 4 looks into each state transition of the extracted STG and analyzes if a fault can be injected during this transition to gain access to a protected state. It first computes the condition, C (line 11) for each transition and if $C == 1$, then it considers the respective transition as *Vulnerable Transition*, VT (line 12, 13). VT is defined as a set of transitions during which a fault can be injected to gain access to a protected state. For each VT, Algorithm 4 reports the conditions that need to be satisfied to perform a setup time violation based fault attack which are shown below,

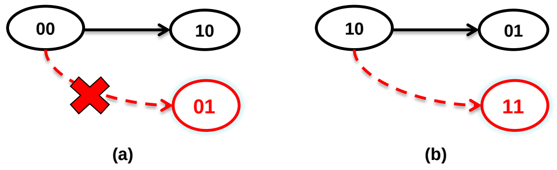

Fig. 6: Setup time violation based fault injection attacks. Fault injection is not possible in case of (a), whereas it is possible in case of (b).

Algorithm 4 Conditions for fault injection attack

```
 1: procedure
 2: Input: Extracted State Transition Graph
 3: Input: set of protected state $P$                          // User given input
 4: Output: Conditions for successful fault attacks
 5:     $T(x, y) \leftarrow$ Extracted $STG$
 6:     for each $T(x, y)$ do                                  // Transition from x to y
 7:         $S_x = [b_{x(n-1)} .....b_{x1}b_{x0}]$              // $S_x$ is state encoding of $x$
 8:         $S_y = [b_{y(n-1)} .....b_{y1}b_{y0}]$              // and $b$ represents each bit of $S$
 9:         $P = [b_{p(n-1)} .....b_{p1}b_{p0}]$
10:         Compute $C = \prod_{i=0}^{(n-1)}((b_{xi} \oplus b_{yi})||(b_{xi} \oplus b_{pi}))$
11:         if ($C == 1$) then
12:             Fault attack possible for $T(x, y)$
13:             $VT(x, y) \leftarrow T(x, y)$
14:             for $i = 0$ to $(n - 1)$ do
15:                 if ($b_{xi} \oplus b_{yi}$) then
16:                     if ($b_{xi} == b_{pi}$) then
17:                         $PV_{x,y}(i) = \{P_{Fs}(i)\}$
18:                     else
19:                         $PO_{x,y}(i) = \{P_{Fs}(i)\}$
20:                     end if
21:                 end if
22:             end for
23:             Compute $SF_{T(x,y)} = \frac{min(PV_{x,y})-max(PO_{x,y})}{avg(P_{FS})}$
24:         else
25:             Fault attack not possible for $T(x, y)$
26:         end if
27:     end for
28: end procedure
29: end procedure
```

- Path Violated: Setup time constraints of the state FFs in these paths need to be violated (line 17), i.e., $P_{Fs} \geq Clock_{period}$. The path delay of these state FFs are represented as PV.
- Path OK: Setup time constraints of the state FFs in these paths need to be maintained (line 19), i.e., $P_{Fs} < Clock_{period}$. The path delay of these state FFs are represented as PO.
- Path No-Effect: State logic bit in this path does not change during the transition and therefore, this path has no impact on the vulnerability analysis.

Apart from the protected states, Algorithm 4 also considers the don't-care states that have access to the protected states and reports the transitions as VT which can give access to these don't-care states. These don't-care states are defined as *Dangerous Don't-Care States* ($DDCS$) and mathematically can be represented as,

$$DDCS = \{s' \mid (A(s') = P) \cap (s' \in S_D)\} \tag{3}$$

Now, each VT may not pose the same level of threat to the implemented FSM. To quantify how susceptible each VT is to fault injection attacks, Algorithm 4 uses $Synopsys's\ Primetime$ tool [26] (for static timing analysis (STA)) to get the maximum path delay of each state FFs. We propose the *susceptibility factor* metric, SF_T to quantitatively measure the vulnerability of each transition to fault injection attacks,

$$SF_T = \frac{min(PV) - max(PO)}{avg(P_{FS})}, \tag{4}$$

Here, $avg(P_{FS})$ is calculated by taking the mean value of all the P_{FS}. $min(PV)$ is the minimum value of delays in *Path Violated*, and $max(PO)$ is the maximum value of delays in *Path OK*. Now, if $min(PV) < max(PO)$, then SF is negative. It means that the delay of a path in *Path OK* is higher than the delay of a path in *Path Violated* and fault injection for this transition is not feasible in the implemented circuit. Therefore, the transitions with negative SF is removed from the set of vulnerable transitions VT.

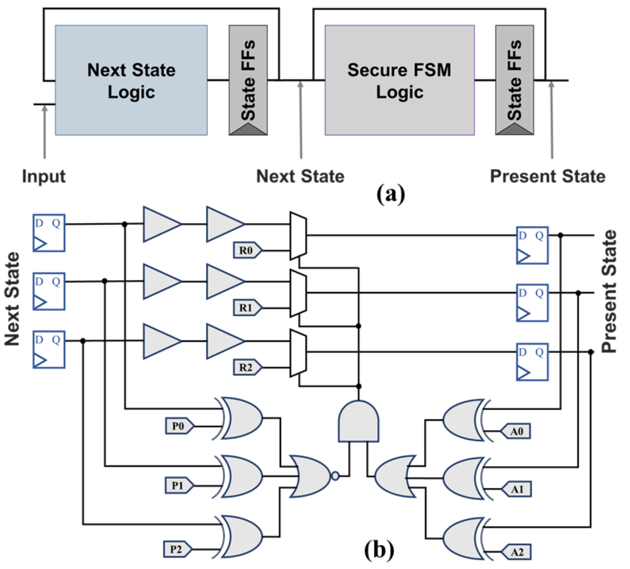

Fig. 7: (a) Security-aware FSM architecture. (b) Detailed implementation strategy. A, P and R represents each bit of 'Authorized', 'Protected' and 'Reset' state.

We propose the following overall metric, *vulnerability factor of fault injection* (VF_{FI}) to measure the overall vulnerability of the FSM to fault injection attacks. VF_{FI} is defined as follows:

$$VF_{FI} = \{PVT(\%), ASF\} \tag{5}$$

where,

$$PVT(\%) = \frac{\sum VT}{\sum T}, \; ASF = \frac{\sum SF}{\sum VT}$$

The metric VF_{FI} is composed of two parameters $\{PVT(\%), ASF\}$. $PVT(\%)$ indicates the percentage of number of vulnerable transitions ($\sum VT$) to total number transitions ($\sum T$), whereas ASF signifies the average of SF. The greater the values of these two parameters are, the more susceptible the FSM is to fault attacks. Note that we consider the P_{FS} to be normally distributed with the mean value of maximum path delay (reported by STA) and a variance value of 5% of $avg(Path_{FS})$ to take into account of process variation.

C. Security-aware FSM architecture

To address the security vulnerabilities identified by AVFSM, we propose a security-aware FSM architecture to ensure that protected states are only accessed from authorized states and cannot be accessed from unauthorized states and don't-care states. Thereby, our proposed architecture can mitigate laser and fault injection attacks, and also addresses the vulnerabilities introduced by the don't-care states. The security-aware FSM architecture requires some additional circuitry that need to be incorporated in the design at RTL level.

The operation of FSM is as follows. During each state transition, the next state logic bits appear at the input of the state FFs. At the positive edge of the clock signal the present state bits (output of the state FFs) are updated to the next state logic values. Because a state transition occurs at every clock edge, it does not provide any opportunity to analyze the next and the present state and verify if the next state transition is authorized. We propose to get around this limitation by incorporating another set of state FFs along with a 'secure FSM logic' and cascade it with the original FSM (see Figure 7 (a).

The 'secure FSM logic' implements the following function and ensures that only the authorized states can access the protected states.

```
**if** ((next_state = Protected) and (present_state /= Authorized)) **then**
    present_state = Reset
**else**
    present_state = Protected
**end if**
```

Note that 'secure FSM logic' can have unbalanced and non-uniform paths if we implement it using a synthesis tool. Therefore, we custom design the 'secure FSM logic' so that it has equal and uniform delay. The detailed implementation is shown in Figure 7 (b). This implementation ensures that all paths from 'Next State' to 'Present State' have nearly uniform and equal delays. Also, the buffers in the architecture ensure that signals from the next state FFs do not reach the present state FFs before the secure FSM logic determines whether the protected state is being accessed by the authorized state. Therefore, an adversary can no longer exploit the non-uniform path delay distribution to perform fault injection attacks. Although, we try to make delays of all paths in secure FSM logic to be uniform and equal, some non-uniformity will be introduced by the process variation during the fabrication process. However, the non-uniformity introduced by the process variation would be much smaller and therefore, would be extremely difficult for an attacker to exploit.

This architecture only adds one cycle latency but does not affect the delay of the FSM which is desirable for most controller circuits. The area overhead associated with our proposed architecture is the extra set of state FFs and the 'secure FSM logic'. Given the simple function implemented by 'secure FSM logic', the associated area overhead is low (see Section VIII-F). Also, our proposed technique does not require any gate-level modification, making it feasible in practice.

VIII. Results and Discussions

In this section, we evaluate our proposed secure FSM design flow using five controller benchmark circuits. First, we give a brief description of the five controller benchmark circuits and illustrate how setup-time violation based fault attack can be mounted on these benchmarks to compromise their security. Then we present a case study using one of these benchmarks (AES controller module) and show that our proposed VF_{FI} metric can effectively capture the probability of successful fault injection attacks on controller circuit. We also encode each benchmark circuit using traditional and proposed encoding schemes, and compare their results in terms of security, performance, and cost. Finally, we validate our security-aware FSM architecture and analyze its area and performance overhead.

All benchmark circuits of our experiemnts were synthesized using Synopsys Design Compiler [26] with 180nm GSCLib Library from Cadence. Note that, our framework is technology independent and therefore, is compatible with any standard cell library. Also note that, our flow is developed for ASIC designs. Therefore, we verified our results through post synthesis simulation using VCS.

A. Fault Attack on Controller Circuits

Here, we discuss how fault injection attacks on FSM can compromise security using five controller benchmark circuits, SHA-256, AES, memory controller, MIPS microprocessor, and RSA (Montgomery ladder algorithm). All these benchmarks are collected from OpenCores [17]. We have already presented fault attacks on SHA-256 controller circuit in Section IV and the rest are discussed below.

AES controller module: The FSM of AES controller circuit is composed of 5 states: 'Wait Key', 'Wait Data', 'Initial Round', 'Do Round' and 'Final Round'. During 'Wait Data' state, the plaintext is loaded into the AES datapath while during 'Initial Round' and 'Do Round' states, ten rounds of AES occur. After ten rounds, the 'Final Round' state is reached and result is latched to the result registers. Two possible attacks can be mounted against this controller circuit. If an attacker can inject a fault and gain access to the 'Final Round' without going through the 'Initial Round' and/or 'Do Round' states, then premature results will be stored, potentially leaking the secret key. If the 'Wait Data' state can be bypassed, the same ciphertext will be generated for every plaintext resulting in a DoS attack. Therefore, for this FSM we consider 'Final Round' and 'Wait Data' as protected states.

Memory controller module: This module allows an external bus master to access the memory bus if the external bus master grants access through 'mc_gnt' signal. We assume that the host CPU authenticates the external bus master and asserts 'mc_gnt' signal. The attacker's objective would be to inject a fault and bump into the state which allows access to memory bus without going through the 'mc_gnt' assertion.

MIPS microprocessor controller: This controller module generates the control bits for the multiplexers, the data memory and ALU control signals for the MIPS processor. It takes the given opcode, as well as the function code from the instruction, and translates it to the individual instruction control signals which are needed for the remaining stages. The FSM of this controller module includes memory read and write states that can only be accessed by memory instructions. Here, we assume that privilege control is implemented in software which analyzes the memory instructions from user kernel and asserts if it does not access memory locations which are dedicated to system kernel. However, if an attacker can inject a fault during non-memory instructions (e.g., add) and access the memory read and/or write states, he/she can potentially bypass the privilege control protection and gain access to system memory locations. Here, we consider the memory read and/or write states as protected states.

RSA controller module: RSA controller module implementing the Montgomery ladder algorithm presented in [3]. The FSM of this controller module consists of 7 states, 'Idle', 'Init', 'Load1', 'Load2', 'Multiply', 'Square', '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 either the key or premature result of RSA encryption. Therefore, 'Result' is the protected state.

B. Case Study on AES

We apply our proposed AVFSM analysis on two implementations of AES encryption module's controller circuit and compare each implementation's vulnerability. First, we have used two different encoding schemes for the FSM of the AES encryption module. We use $\{$ *Wait Key, Wait Data, Initial Round, Do Round, Final Round*$\}$ = $\{$ *000, 001, 010, 011, 100*$\}$ and $\{$ *Wait Key, Wait Data, Initial Round, Do Round, Final Roun*$\}$ = $\{$ *000, 100, 011, 101, 111*$\}$ for schemes I and II, respectively. We then synthesize each scheme with medium area effort and apply the AVFSM analysis to evaluate the vulnerabilities of each implemented FSM. Our analysis returns the following assessment: $VF_{FI} = \{0, 0\}$ for scheme I and $VF_{FI} = \{23\%, 0.38\}$ for scheme II. As explained in Section VII-B, a greater the value of VF_{FI} represents the FSM to be more vulnerable fault attacks. Therefore, we can expect scheme II to be vulnerable to fault attack, whereas scheme I is not.

We validate the AVFSM analysis by simulating a setup-time violation based fault attack on scheme I and II. We perform this simulated fault attack by increasing the clock frequency to cause setup time violation. Note that, reducing the voltage and/or increasing the temperature will have the same effect. For scheme II, the fault attack causes the FSM to bump to 'Final Round' without going through the round operations, potentially leaking the key. This attack works as follows, during the transition from 'Wait Data' (100) to 'Initial Round' (011), the setup time of state FF(2) is violated while the setup time of state FF(1) and state FF(0) is maintained. Therefore, instead of going to 'Initial Round', the FSM bumps to 'Final Round'. When the 'Final Round' is reached, the 'finished' signal is asserted to 1 causing the expanded key to be captured by the result register.

However, we were unable to induce a successful fault attack on scheme I. This case study validates that our AVFSM technique can correctly capture the probability of successful fault injection attack.

C. Comparison of Different Encoding Schemes

We compare the security, cost and performance of our proposed encoding schemes with traditional encoding schemes, i.e., binary and one-hot on the benchmark controller circuits described in Section VIII-A. We quantitatively analyze the security of the FSMs using the VF_{FI} metric. We utilize the Design Compiler tool to obtain the area and the maximum delay of each benchmark.

Table III summarizes all results. The area and delay shown in columns 6 and 7 reflect the cost and performance (critical path delay) of the controller circuit, respectively. Note that, the controller circuit of SHA, AES and RSA contain FSMs with less than 8 states. It may appear that our analysis was performed on smaller benchmark circuits. Actually, most implementations of cryptographic algorithms tend to have a small number of states (based on our review of crypto designs at opencore [17]). These crypto controller circuits would most likely be targeted for fault attack. We also apply our analysis on larger controller circuits, e.g., memory controller FSM with 66 states to show the scalability and applicability of our approach.

From Table III we can make the following important observations: (i) for all the benchmark circuits, binary encoding makes the FSM vulnerable to fault attack. Also, the greater value of VF_{FI} for binary encoding reflects binary encoding-based FSMs are more susceptible to faults attacks. (ii) One-hot encoding requires more area for all the benchmark circuits; though it is better compared to binary encoding from a security perspective. (iii) Our proposed encoding schemes offer much better security while having little or no cost and performance overhead. (iv) Only in one benchmark FSM (MIPS microprocessor), our proposed encoding schemes are vulnerable to fault attack. These security vulnerabilities are created by synthesis tools in the form of don't-care states and the encoding schemes cannot address these vulnerabilities as mentioned in Section VII. For this FSM, we need to apply our proposed FSM architecture to make it secure against fault attack. (v) Scheme II has better cost and performance compared to Scheme I. However, Scheme II requires more designer involvement and therefore, may increase time-to-market w.r.t. to Scheme I.

D. Security and Cost of Linear EDC

We analyze the security and cost of linear EDC based approaches using the benchmark controller circuits described in Section VIII-A. We first encode the FSMs of AES, SHA and RSA controller modules using Hamming (7,4) code and encode the FSMs of MIPS micro-processor and memory

TABLE III: Results for Different FSM Encoding Schemes

	Encoding scheme	# states	# state FFs	# Don't -care states	Area (μm^2)	Delay (ns)	Security concern?	$VF_{FI} = ASF, PVT$
AES	Binary	5	3	3	3068	0.62	Yes	0.23, 0.38
	One-hot		5	27	4380	0.7	No	0, 0
	Scheme I		4	11	3768	0.64	No	0, 0
	Scheme II		3	3	3146	0.63	No	0, 0
SHA	Binary	7	3	1	4495	2.69	Yes	0.10, 0.35
	One-hot		7	121	6701	3.12	Yes	0.10, 0.07
	Scheme I		4	9	5551	2.92	No	0, 0
	Scheme II		3	1	4976	3.31	No	0, 0
MIPS Processor	Binary	19	5	13	9346	1.6	Yes	0.42, 0.09
	One-hot		19	5.2e3	19816	1.52	Yes	0.26, 0.07
	Scheme I		9	496	12357	1.55	Yes	0.08, 0.003
	Scheme II		5	13	9330	1.58	Yes	0.11, 0.06
Memory Contrl.	Binary	66	7	62	60039	1.47	Yes	0.09, 0.01
	One-hot		66	7.3e19	68904	1.45	No	0, 0
	Scheme I		8	190	57624	1.54	No	0, 0
	Scheme II		8	190	57624	1.54	No	0, 0
RSA	Binary	7	3	1	3099	0.55	Yes	0.09, 0.12
	One-hot		7	121	5519	0.69	No	0, 0
	Scheme I		4	9	3632	0.57	No	0, 0
	Scheme II		3	1	3204	0.52	No	0, 0

controller modules using Hamming (15,11) code. We then place the error detection circuitry at the output of the state FFs. We quantitatively analyze the security of the FSMs using the VF_{FI} metric [12]. Table IV summarizes the results.

From Table IV, we can observe that EDC based approaches cannot provide adequate protection as indicated by the VF_{FI} metric. For all controller benchmark circuits except the memory controller, the FSMs are susceptible to fault attack even with error detection protection in place. The reason is that EDC based approaches do not take into account the biased fault model, where an attacker can exploit the non-uniform path delay distribution of an FSM to flip multiple targeted bits and bypass normal state transitions to bump into a protected state directly. Also, there are valid don't-care states in EDC based countermeasure as shown in column 5 in Table IV. If these valid don't-care states have access to a protected state, they present a security vulnerability; as these don't-care states can be exploited to access the protected state without triggering the error detection mechanism.

It is clear from Table IV that the EDC based countermeasures have much higher area overhead (on average 54.72% w.r.t. Scheme I and 75.59% w.r.t. Scheme II) and timing overhead (on average 45.99% w.r.t. Scheme I and 46.57% w.r.t. Scheme II) compared to our proposed security-aware encoding schemes. The reason is that, unlike our proposed encoding schemes which make the FSM inherently resilient to fault attack, EDC approaches require additional circuitry for error detection. However, our proposed security-aware encoding offers better security (fault attack resilience) compared with EDC based approaches (validated by VF_{FI} metric).

Note that, there are other EDC based approaches (e.g., [3], [5], [10], [8]) which have higher error detectability than Hamming code. However, none of these approaches address the vulnerabilities introduced by the synthesis tool and therefore, cannot provide adequate protection to fault attack.

E. Security and Cost of Non-Linear EDC

Non-linear EDC-based techniques [3], [13], [14] utilize the 'Error Detectability' metric to evaluate the security of the algorithm. 'Error Detectability' states the probability of detecting any error due to faults by an EDC algorithm. For example, 'Error Detectability' of $1-2^{-16}$ means a fault error can be detected with a probability of 0.99998. Therefore, a non-linear EDC-based technique with adequate 'Error Detectability' metric an provide protection for any bit-flips due to fault. Therefore, these techniques can potentially detect fault injection attacks which can individually set or reset a single FF without affecting any other gates utilizing FIBs.

However, the non-linear EDC-based techniques require impractical area and delay overhead. For example, to protect the RSA controller module which consists of 80 gates and has a delay of 0.64 ns with a minimum error detectability of $1-2^{-2}$, the technique proposed in [16] requires 8,305 gates (area overhead 10,281%) and has a delay of 29.99 ns (delay overhead 4,586%) [16]. If the minimum error detectability is raised to $1-2^{-16}$, this technique requires 96,096 gates (area overhead 120,020%) and has a delay of 294.57 ns (delay overhead 45,926%) [16]. Even considering the overall design, these area and delay overhead are prohibitively expensive for most applications.

TABLE IV: Results for EDC Based Approaches.

	Encoding scheme	# states	# state FFs	# Valid don't -care states	Area (μm^2)	Delay (ns)	Security concern?	VF_{FI} = ASF, PVT
AES	Hamming (7,4)	5	7	11	6606	1.07	Yes	0.23, 0.07
SHA	Hamming (7,4)	7	7	9	7955	3.64	Yes	0.10, 0.70
MIPS Processor	Hamming (15,11)	19	15	2029	12534	1.71	Yes	0.11, .003
Memory Contrl.	Hamming (15,11)	66	15	1982	85976	2.32	No	0, 0
RSA	Hamming (7,4)	7	7	9	7197	1.01	Yes	0.19, 0.02

We evaluate the security of our proposed encoding schemes using the VF_{FI} metric. It is similar in spirit to the 'Error Detectability' as it provides a mathematical probabilistic measure of successful fault injection attack. For example, VF_{FI} of (0,0) means a protected state cannot be accessed by an unauthorized state and the respective FSM is not vulnerable biased fault attacks. Table III shows that our proposed approach can provide adequate security for most FSMs. Also, our approach requires negligible area and delay overhead. Note that, our proposed technique does not cover FIB-based fault attack as discussed in Section V.

F. Results of Our Proposed FSM Architecture

We first validate that our proposed security-aware FSM architecture has equal and uniform path delay distribution. Table V shows the minimum and maximum delay of our 'Secure FSM Logic' for the binary encoded AES controller circuit where the Final Round' is the protected state. NS and PS in the table represent the Next State and Present State bits, respectively. Table V shows that all paths from 'Next State' to 'Present State' have almost equal and uniform delay. Therefore, an adversary can no longer exploit the non-uniform path delay distribution to perform fault injection attack. We implemented the security-aware FSM architecture for different benchmarks with different number of protected states. Our experiments show that our proposed architecture ensures almost uniform and equal delay for these cases as well.

We also validated the security provided by our proposed security-aware FSM architecture, by applying it to the fault attack scenario presented in Section VIII-B. We place our secure FSM architecture on the scheme II of the AES controller circuit and apply the same fault attack environment. However, we were unable to perform a successful fault attack. The reason is that when the protected state 'Final Round' (111) is tried to be accessed from an unauthorized state 'Wait Data' (100), the secure FSM logic changes the present_state to '000' state; thereby protecting the FSM from fault attack. Also, we have performed simulated fault attacks by introducing glitches in the clock signal to cause setup time violation of state FFs and our results show this architecture can effectively prevent these attacks as well.

TABLE V: Delay distribution of 'Secure FSM Logic'

Path	Min. delay (ns)	Max. delay (ns)
$NS[0] \longrightarrow PS[0]$	0.34	0.35
$NS[1] \longrightarrow PS[1]$	0.34	0.36
$NS[2] \longrightarrow PS[2]$	0.32	0.35

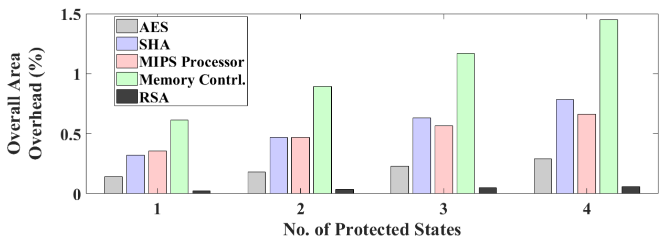

Fig. 8: Security-aware FSM architecture. Area overhead analysis w.r.t. overall design.

Fig. 8 presents the area overhead as a function of the number of protected states when our proposed FSM architecture is applied to the five controller circuit. The overhead shown is with respect to the overall design (including controller and datapath). Fig. 8 shows that the area overhead increases linearly with respect to the number of protected states. The reason is that, the area corresponding to the 'secure FSM logic' increases linearly with the number of protected states. Also, we can observe that the area overhead is significantly small ($< 1.5\%$) compared to the overall design. Moreover, our proposed architecture has no delay overhead; it adds only one cycle latency to the design. Note that, the area overhead for RSA is much smaller compared to the Memory Controller. The reason is that the overall area of RSA is much larger than the Memory Controller.

Our flow can serve as an important component of the recent academic and industrial initiative for developing CAD frameworks for automated security vulnerability assessment of hardware designs[29], [30]. The FSM extraction of our proposed flow can be used for different applications, e.g., extracting trigger sequence [31] for sequential Trojan [32], [33], [34].

IX. Conclusion

In this paper, we presented a security-aware FSM design flow to make the FSM fully secure against fault injection attacks. To establish this flow, we proposed two security-aware FSM encoding schemes. Our encoding schemes inherently make the FSM tolerant to fault attacks without the need of any extra circuitry. We also proposed a framework that systematically analyzes and evaluates vulnerabilities in the FSM against fault injection attacks. Our proposed flow allows the designer to find security vulnerabilities in the FSM at an early design stage. Our approach also enables the designer to quantitatively compare the security of different implementations of the same design. If vulnerabilities exist in the design then our proposed mitigation technique can be applied to make the FSM secure against such attacks. To eliminate the vulnerabilities identified by AVFSM, we propose a security-aware FSM architecture. Our proposed FSM architecture provides protection against laser-based fault attack and addresses the vulnerability introduced by CAD tools which are not covered by the encoding schemes. The proposed approach can easily be incorporated to the current ASIC flow.

X. Acknowledgments

This work was supported in part by Semiconductor Research Corporation (SRC) under task #2724, Cisco Systems, Inc. (Cisco) and NSF (1441667).

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] B. Yuce, N. F. Ghalaty, C. Deshpande, C. Patrick, L. Nazhandali, and P. Schaumont, "Fame: Fault-attack aware microprocessor extensions for hardware fault detection and software fault response," in *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*. ACM, 2016, p. 8.
[3] B. Sunar, G. Gaubatz, and E. Savas, "Sequential circuit design for embedded cryptographic applications resilient to adversarial faults," *IEEE Transactions on Computers*, vol. 57, no. 1, pp. 126–138, 2008.
[4] M. Berg, "Fault tolerant design techniques for asynchronous single event upsets within synchronous finite state machine architectures," in *7th International Military and Aerospace Programmable Logic Devices (MAPLD) Conference. NASA (September 2004)*, 2004.
[5] A. Krasniewski, "Concurrent error detection for finite state machines implemented with embedded memory blocks of sram-based fpgas," *Microprocessors and Microsystems*, vol. 32, no. 5, pp. 303–312, 2008.
[6] F. Farahmandi and P. Mishra, "Fsm anomaly detection using formal analysis," in *2017 IEEE 35th International Conference on Computer Design (ICCD)*. IEEE, 2017, pp. 313–320.
[7] S. Baranov, I. Levin, O. Keren, and M. Karpovsky, "Designing fault tolerant fsm by nano-pla," in *On-Line Testing Symposium, 2009. IOLTS 2009. 15th IEEE International*. IEEE, 2009, pp. 229–234.
[8] Z. Wang and M. Karpovsky, "Robust fsms for cryptographic devices resilient to strong fault injection attacks," in *On-Line Testing Symposium (IOLTS), 2010 IEEE 16th International*. IEEE, 2010, pp. 240–245.
[9] 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.
[10] Z. Wang, M. Karpovsky, and B. Sunar, "Multilinear codes for robust error detection," in *On-Line Testing Symposium, 2009. IOLTS 2009. 15th IEEE International*. IEEE, 2009, pp. 164–169.
[11] C. Dunbar and G. Qu, "Designing trusted embedded systems from finite state machines," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 5s, p. 153, 2014.
[12] A. Nahiyan, K. Xiao, K. Yang, Y. Jin, D. Forte, and M. Tehranipoor, "Avfsm: a framework for identifying and mitigating vulnerabilities in fsms," in *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*. IEEE, 2016, pp. 1–6.
[13] 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.
[14] K. D. Akdemir, Z. Wang, M. Karpovsky, and B. Sunar, "Design of cryptographic devices resilient to fault injection attacks using nonlinear robust codes," in *Fault Analysis in Cryptography*. Springer, 2012, pp. 171–199.
[15] Z. Wang, "Nonlinear robust codes and their applications for design of reliable and secure devices," Ph.D. dissertation, BOSTON UNIVERSITY, 2011.
[16] K. D. Akdemir, "Error detection techniques against strong adversaries," Ph.D. dissertation, Worcester Polytechnic Institute, 2010.
[17] *OpenCores*, http://opencores.org.
[18] D. R. Patel, *Information Security: Theory and Practice*. PHI Learning Pvt. Ltd., 2008.
[19] D. Molnar, M. Stevens, A. Lenstra, B. de Weger, A. Sotirov, J. Appelbaum, D. A. Osvik, D. Day, and R. Saal, "Md5 considered harmful today," in *Proceedings of 25th Chaos Computer Congress*, 2009.
[20] B. Yuce, N. F. Ghalaty, and P. Schaumont, "Tvvf: Estimating the vulnerability of hardware cryptosystems against timing violation attacks," in *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 72–77.
[21] N. F. Ghalaty, B. Yuce, and P. Schaumont, "Analyzing the efficiency of biased-fault based attacks," *IEEE Embedded Systems Letters*, vol. 8, no. 2, pp. 33–36, 2016.
[22] K. Kuusilinna, V. Lahtinen, T. Hämäläinen, and J. Saarinen, "Finite state machine encoding for vhdl synthesis," *IEE Proceedings-Computers and Digital Techniques*, vol. 148, no. 1, pp. 23–30, 2001.

[23] T. K. Moon, "Error correction coding," *Mathematical Methods and Algorithms. Jhon Wiley and Son*, 2005.

[24] L. Yuan, G. Qu, T. Villa, and A. Sangiovanni-Vincentelli, "An fsm reengineering approach to sequential circuit synthesis by state splitting," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 6, pp. 1159–1164, 2008.

[25] Y. Shi, C. W. Ting, B.-H. Gwee, and Y. Ren, "A highly efficient method for extracting fsms from flattened gate-level netlist," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*. IEEE, 2010, pp. 2610–2613.

[26] *Synopsys*, Available at http://www.synopsys.com/.

[27] L. Zussa, J.-M. Dutertre, J. Clédiere, B. Robisson, A. Tria *et al.*, "Investigation of timing constraints violation as a fault injection means," in *27th Conference on Design of Circuits and Integrated Systems (DCIS), Avignon, France*, 2012.

[28] 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.

[29] A. Nahiyan, K. Xiao, D. Forte, and M. Tehranipoor, "Security rule check," in *Hardware IP Security and Trust*. Springer, 2017, pp. 17–36.

[30] K. Xiao, A. Nahiyan, and M. Tehranipoor, "Security rule checking in ic design," *Computer*, vol. 49, no. 8, pp. 54–61, 2016.

[31] A. Nahiyan, M. Sadi, R. Vittal, G. Contreras, D. Forte, and M. Tehranipoor, "Hardware trojan detection through information flow security verification," in *Test Conference (ITC), 2017 IEEE International*. IEEE, 2017, pp. 1–10.

[32] F. Farahmandi, Y. Huang, and P. Mishra, "Trojan localization using symbolic algebra," in *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*. IEEE, 2017, pp. 591–597.

[33] J. Cruz, F. Farahmandi, A. Ahmed, and P. Mishra, "Hardware trojan detection using atpg and model checking," in *VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID), 2018 31st International Conference on*. IEEE, 2018, pp. 91–96.

[34] T. Hoque, S. Narasimhan, X. Wang, S. Mal-Sarkar, and S. Bhunia, "Golden-free hardware trojan detection with high sensitivity under process noise," *Journal of Electronic Testing*, vol. 33, no. 1, pp. 107–124, 2017.

Adib Nahiyan obtained his BS in Electrical Engineering from Bangladesh University of Engineering and Technology (BUET) in 2014. Currently he is a PhD student in the Electrical and Computer Engineering department at University of Florida, Gainesville, USA. His PhD studies are funded by Semiconductor Research Corporation (SRC) and Cisco. During his graduate studies he worked as research intern with Cisco, NC. His research interest includes Hardware Security, Secure VLSI Design and Finding Vulnerabilities in ASIC Design.

Farimah Farahmandi is a Ph.D candidate at the Department of Computer and Information Science and Engineering (CISE) at the University of Florida. She received her B.S. and M.S. from University of Tehran, Iran in 2010 and 2013 respectively. Her current research interests include formal verification, hardware security and post-silicon validation and debug. She was a research intern at Cisco advanced security research group. She is the recipient of IEEE System Validation and Debug Technology Committee Student Research Award, 2017.

Prabhat Mishra is a Professor in the Department of Computer and Information Science and Engineering at the University of Florida. His research interests include embedded and cyber-physical systems, energy-aware computing, hardware security and trust, system-on-chip verification, bioinformatics, and post-silicon validation and debug. He received his Ph.D. in Computer Science and Engineering from the University of California, Irvine. He has published five books and more than 150 research articles in premier international journals and conferences. His research has been recognized by several awards including the NSF CAREER Award, IBM Faculty Award, three best paper awards, and EDAA Outstanding Dissertation Award. Prof. Mishra currently serves as the Deputy Editor-in-Chief of IET Computers & Digital Techniques, and as an Associate Editor of ACM Transactions on Design Automation of Electronic Systems, IEEE Transactions on VLSI Systems, and Journal of Electronic Testing. He has served on many conference organizing committees and technical program committees of premier ACM and IEEE conferences. He is currently serving as an ACM Distinguished Speaker. Prof. Mishra is an ACM Distinguished Scientist and a Senior Member of IEEE.

Domenic Forte is an Assistant Professor with the Electrical and Computer Engineering Department at University of Florida, Gainesville, FL where he leads multiple efforts within the Florida Institute for Cybersecurity Research (FICS Research). His research covers the entire domain of hardware security from nanoscale devices to printed circuit boards (PCBs). Topics include hardware security primitives, hardware Trojan detection and prevention, security of the electronics supply chain, security-aware design automation tools, reverse engineering, and anti-reverse engineering. He also performs research in biometric system security, reliability, and implementation with specialization in physiological signals such as electrocardiogram (ECG) and photoplethysmograph (PPG).

Dr. Forte is a recipient of the NSF Faculty Early Career Development Program (CAREER) Award (2017), Army Research Office (ARO) Young Investigator Award (2016), Northrop Grumman Fellowship (2012), and George Corcoran Memorial Outstanding Teaching Award (2008). His research has been also recognized through best paper awards and nominations from multiple organizations and conferences. He is a member of the Institute of Electrical and Electronics Engineers (IEEE) and IEEE Circuits and Systems Society (CAS). He serves on the organizing committees of top conferences in hardware security such as IEEE International Symposium on Hardware Oriented Security and Trust (HOST) and Asian HOST. He also serves on the technical program committees of including Design Automation Conference (DAC), International Conference on Computer-Aided Design (ICCAD), Network and Distributed System Security Symposium (NDSS), IEEE International Test Conference (ITC), and International Symposium for Testing and Failure Analysis (ISTFA). He is a co-author of the book Counterfeit Integrated Circuits-Detection and Avoidance, co-editor of the book Hardware Protection through Obfuscation, and co-editor of the book " Security Opportunities in Nano Devices and Emerging Technologies ". He is an Associate Editor of Journal of Hardware and Systems Security (HaSS) and was the Guest Editor of the IEEE Computer 2016 Special Issue on Supply Chain Security for Cyber-Infrastructure.

Mark Tehranipoor (S'02M'04SM'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 400 journal articles and refereed conference papers and has given more than 175 invited talks and keynote addresses. He has published 10 books and more than 20 book chapters. 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 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, 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.