Journal of Electronic Testing (2022) 38:527–546
https://doi.org/10.1007/s10836-022-06028-5

AFIA: ATPG-Guided Fault Injection Attack on Secure Logic Locking

Yadi Zhong[1] · Ayush Jain[1] · M. Tanjidur Rahman[2] · Navid Asadizanjani[2] · Jiafeng Xie[3] · Ujjwal Guin[1]

Received: 10 June 2022 / Accepted: 3 October 2022 / Published online: 3 November 2022

Abstract

The outsourcing of the design and manufacturing of integrated circuits has raised severe concerns about the piracy of Intellectual Properties and illegal overproduction. Logic locking has emerged as an obfuscation technique to protect outsourced chip designs, where the circuit netlist is locked and can only be functional once a secure key is programmed. However, Boolean Satisfiability-based attacks have shown to break logic locking, simultaneously motivating researchers to develop more secure countermeasures. In this paper, we present a novel fault injection-based attack to break any locking technique that relies on a stored secret key, and denote this attack as ***AFIA***, ***A***TPG-guided ***F***ault ***I***njection ***A***ttack. The proposed attack is based on sensitizing a key bit to the primary output while injecting faults at a few other key lines that block the propagation of the targeted key bit. AFIA is very effective in determining a key bit as there exists a stuck-at fault pattern that detects a stuck-at 1 (or stuck-at 0) fault at any key line. The average complexity of the number of injected faults for AFIA is linear with the key size $\mathcal{K}$ and requires only $\mathcal{K}$ test patterns to determine a secret key K. AFIA requires fewer injected faults to sensitize a bit to the primary output, compared to $2\mathcal{K}-1$ faults for the differential fault analysis attack illustrated in our previous work.

Keywords Logic locking · Differential fault analysis · Fault injection · IP Piracy · IC overproduction

1 Introduction

Over the last few decades, the impact of globalization has transformed the integrated circuit (IC) manufacturing and testing industry from vertical to horizontal integration. The continuous trend of device scaling has enabled the designer to incorporate more functionality in a system-on-chip (SoC) by adopting lower technology nodes to increase performance and reduce the overall area and cost of a system. Currently, most SoC design companies or design houses no longer manufacture chips and maintain a foundry (fab) of their own. This is largely due to the increased complexity in the fabrication process as new technology development is being adopted. The cost for building and maintaining such foundries is estimated to be a multi-million dollar investment [99]. As modern integrated circuits (ICs) are becoming more complex, parts of the design are reused instead of designing the whole from scratch. As a result, the design house integrates intellectual properties (IP) obtained from different third-party IP vendors and outsources the manufacturing to an offshore foundry. Due to this distributed design and manufacturing flow, which includes designing SoCs using third-party IPs, manufacturing, testing, and distribution of chips,

Responsible Editor: N, Karimi

✉ Ujjwal Guin
ujjwal.guin@auburn.edu

Yadi Zhong
yadi@auburn.edu

Ayush Jain
ayush.jain@auburn.edu

M. Tanjidur Rahman
mir.rahman@ufl.edu

Navid Asadizanjani
nasadi@ufl.edu

Jiafeng Xie
jiafeng.xie@villanova.edu

[1] Department of Electrical and Computer Engineering, Auburn University, Auburn, AL 36849, USA

[2] Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611, USA

[3] Electrical and Computer Engineering, Villanova University, Villanova, PA 19085, USA

various threats have emerged in recent years [1, 17, 85]. The research community has also been extensively involved in proposing countermeasures against these threats [19, 29, 35, 36, 49, 55, 59].

Logic locking has emerged as the most prominent method to address the threats from untrusted manufacturing [1, 18, 43, 59]. In logic locking, the netlist of a circuit is locked with a secret key so that the circuit produces incorrect results in regular operation unless the same key is programmed into the chip. Figure 1a shows an abstract view of logic locking where the key is stored in a tamper-proof memory and is applied to the locked circuit to unlock its functionality. The key needs to be kept secret, and care must be taken during the design process so that this secret key is not leaked to the primary output directly during the operation. The common logic locking techniques insert additional logic elements like XOR gates [59], multiplexers (MUXs) [56], and look-up tables (LUTs) [11] to lock the circuit functionality, and are shown in Fig. 1b. A SAT attack, by Subramanyan et al. [81], was among the first to efficiently attack a range of locking schemes. With this SAT analysis, the key of a locked circuit is determined in a short period of time. The SAT attack requires the locked netlist, recovered through reverse engineering, and a functional working chip. Since then, several SAT-resistant locking techniques have emerged [7, 31, 37, 38, 50, 58, 66, 72, 82, 90, 92, 93, 98] and many of them were broken soon after they have been proposed [23, 24, 26, 39, 40, 44, 51, 71, 103]. The majority of the research has been directed towards SAT attack resiliency. *However, can we reliably state that a logic locking technique is completely secure even if we achieve complete SAT resistivity?*

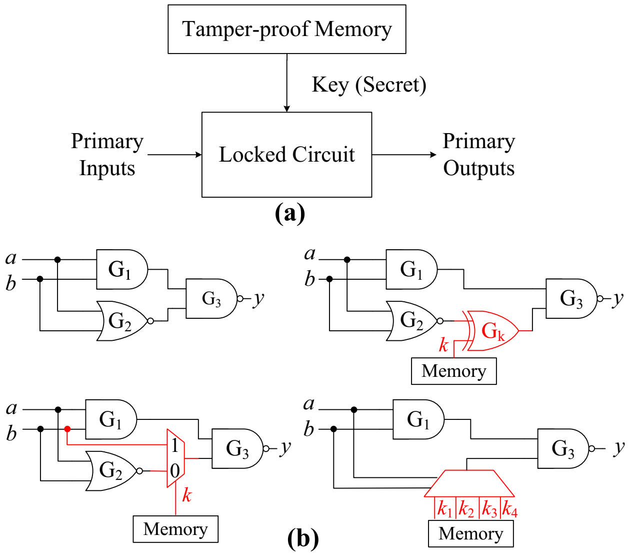

Fig. 1 Logic Locking: **a** An abstract view of the logic locking. **b** Different types of logic locking techniques with XOR/XNOR, MUX and LUT

An untrusted foundry can be treated as an adversary as logic locking is proposed to protect designs from untrusted manufacturing. The adversary has many more effective means to determine the secret key without performing SAT analysis. A few of these attacks can be in the form of probing [53, 54], inserting a hardware Trojan in the design [34], and analyzing the circuit topology [76, 101, 102]. Countermeasures are also developed to partially prevent these attacks [3, 39, 75, 77, 89, 100–102].

Unlike cryptosystems, not all input patterns for a locked circuit are valid for propagating the incorrect key values to the primary outputs. Instead, only a few patterns may exist to carry the values of key bits to the output, similar to the identification of hard-to-detect faults. This is especially true for Post-SAT solutions [68, 70, 94, 96, 98], where they minimize the output corruptibility for incorrect keys. For logic locking, some key bits can block the propagation of the target key bit, i.e. SLL [55] and Post-SAT designs. This is different from the fault injection attack in cryptography, where an entirely new output can be observed under any input pattern even though there is a single bit change in the key as the plaintext goes through many transformations (e.g., Shift Rows, Mix Columns and key addition for AES) [13, 25, 45]. It is trivial for a cryptosystem to change one key bit and apply a random pattern. Unfortunately, this is not the case for a circuit locked with a secret key. It is hard to observe the output change with the change of a single key bit by applying a random pattern. The novelty of this paper is that we apply the methodology in ATPG to efficiently derive the desired input pattern, which guarantees the change in output under different keys and helps launch the fault injection attack.

This paper shows how an adversary can extract the secret key from a locked netlist, even if all the existing countermeasures are in place. An adversary can determine the secret key by injecting faults at the key registers [53, 54], which hold the key value during normal operation, and performing differential fault analysis. In this paper, we present ***AFIA***, key sensitization-based ***A***TPG-guided ***F***ault ***I***njection ***A***ttack, to break any locking scheme. The entire process can be performed in three steps. First, we process the locked netlist and converted it into a directed graph to extract all logic cones and construct a key-cone association matrix that records the distribution of keys among different cones. This structural analysis facilitates total fault reduction for subsequent test pattern generation. Second, it is necessary to select an input pattern that produces an incorrect response for the target key bit only while keeping its dependent keys at faulty states. This can be achieved by using a constrained automatic test pattern generation (ATPG) [15] to generate such a test pattern, which is widely popular for testing VLSI circuits. It is a simple yet effective way to determine a 1-bit key by generating a test pattern that can detect the stuck-at fault (*saf*) at the target key (corresponds to that key bit) while keeping its dependent keys at logic 1 (or logic 0). Dependencies are often inserted [97] to prevent direct sensitization of a key bit to the output by test patterns due to other key lines blocking its path. In our proposed approach, the pattern which detects a stuck-at 1 (*sa1*) fault at one key line with logical constraints for the recovered key lines is sufficient to determine that key bit. One can also use stuck-at 0 (*sa0*) fault to derive such pattern. Note that the fault-free and faulty responses are always the complements under the test pattern that detects that fault, which helps to derive the key bit value. The same process needs to be applied for other key bits to generate such input patterns, and this results in most $\mathcal{K}$ patterns for determining the entire key of size $\mathcal{K}$. Note that one test pattern can detect multiple key bits when they are placed in different logic cones (no dependencies). Third, we apply these test patterns to only one instance of unlocked chip obtained from the market and collect the responses. Faults can be injected at the blocking key registers using laser fault injection equipment (see Sect. 5.1 for details) and obtain the key value by comparing the output responses with test patterns' generated by constrained ATPG. This is a significant improvement compared to our previous conference paper [33] where differential fault analysis requires injection of faults twice.

The contributions of this paper are described as follows:

- We propose a novel attack to break secure logic locking techniques using fault injection-based method. The basic idea behind the attack is the availability of an input pattern that sensitizes a key bit to the primary output. If there are interdependencies among keys, fault injection is necessary only for the dependent key bits in order to propagate the desired ones to the output. Multiple key bits can be sensitized to the outputs if they are placed in different logic cones during locking. ***To the best of our knowledge, we are the first to demonstrate that the stuck-at fault patterns can be used to determine the secret key of a locked circuit with fault injections on interdependent keys.***
- The proposed attack can be launched very efficiently with the minimum number of injected faults. It is necessary to inject faults only to ensure the proper key propagation, whereas our prior work [33] requires $2\mathcal{K} - 1$ faults ($\mathcal{K} - 1$ faults for C_A and $\mathcal{K}$ faults for C_F) to determine one key bit. In addition, our proposed cone analysis approach can find key bits which are located in different cones in parallel. As fault injection is an expensive process, we propose to generate test patterns that reduce the number of injected faults. Each key bit is targeted one at a time to minimize the number of faults. Note that fault injection is necessary when a group of key bits blocks the propagation of a targeted key bit to the output.
- We demonstrate the feasibility of our proposed fault injection attack using Hamamatsu PHEMOS-1000, a laser fault injection equipment, on a Kintex-7 FPGA [95]. We have performed extensive simulations on different benchmarks with secure locking techniques. Constrained ATPG using the Synopsys TetraMAX tool [84] is used to generate test patterns to simulate the attack. The simulation result shows a significant reduction of total fault count for AFIA compared to DFA [33] in breaking the same locked benchmark.

The rest of the paper is organized as follows: An overview of different logic locking techniques and existing attacks along with fault injection techniques is provided in Sect. 2. We describe our previously published attack [33] in Sect. 3. The proposed attack and its methodology to extract the secret key from any locked circuit are described in Sect. 4. We present the results for the implementation of the proposed attack on different locked benchmark circuits in Sect. 5. Finally, we conclude our paper in Sect. 7.

2 Prior Work

The prior work related to logic locking and fault injection techniques is described in this section.

2.1 Logic Locking

As mentioned in Section 1, the objective of logic locking is to obfuscate the functionality of the original circuit by inserting a lock (secret key). The key-dependent circuit makes it difficult for the adversary to pirate or analyze the original circuit directly. In this context, various traditional logic locking techniques were based on different location selection algorithms for key gate placement, such as random (RLL) [60], fault analysis-based (FLL) [55], and strong interference-based logic locking (SLL) [56]. To demonstrate the capabilities of an adversary, Subramanyan et al. [81] developed a technique using Boolean Satisfiability (SAT) analysis to obtain the secret key from a locked chip. This oracle-guided SAT attack iteratively rules out incorrect key values from the key space by using distinguishing input patterns and the corresponding oracle responses.

In post-SAT era, resiliency against the SAT attack became one of the crucial metrics to demonstrate the effectiveness of newly proposed schemes [40]. Sengupta et al. proposed stuck-at-fault based stripping of original netlist and reconstruction to form the locked netlist, where incorrect results are produced only for chosen input patterns [66, 98]. Simultaneously, researchers have adopted a different direction to tackle the SAT attack, including restricting access to the internal states of a circuit through scan-chains. Guin et al. proposed a design that prevents scanning out the internal states of a design after a chip is activated and the keys are programmed/stored in the circuit [30, 31]. The concept of scan locking gained significant interest from the researchers, which led to the development of various scan-chain-based locking schemes [41, 51, 90]. Alrahis et al. attack scan-chain-based locking schemes by unrolling the sequential circuit to a combinational one, which is then provided to the SAT solver to extract the secret key [4]. Sisejkovic et al. [77] proposed an oracle-less structural analysis attack on MUX-based (SAAM) logic locking to exploit insertion flaws in MUX-based key gates. Deceptive multiplexer-based (D-MUX) logic locking is proposed to achieve functional secrecy [73] against both SAAM and oracle-less machine learning-based attacks. As combinational feedback loops are not translatable to SAT problems, cyclic-based locking [58] is resistant to the initial SAT attack [81]. In addition, there has extensive efforts in the proposal of non-functional logic locking techniques, such as scan-chain-based [6, 51], timing-based locking [7, 50, 82, 93], and routing-based locking [37–39].

As the research community explores new directions to understand an attacker's latent qualities, new attacks on logic locking have been proposed. An adversary may perform direct or indirect probing on the key interconnects or registers [54]. An attacker is not required to understand the complete functionality of the circuit to perform these attacks. In this, Rahman et al. demonstrated how an attacker could target the key registers and perform optical probing to gain knowledge regarding the fixed value for those registers. Following this, tampering attacks can also become an attacker's primary choice. Jain et al. exploited this notion to extract the secret key by implementing hardware Trojans inside the locked netlist [34]. Without an oracle, the attribute of repeated functionality in the circuit can also be used to compare the locked unit functions and their unlocked version to predict the secret key [101]. This makes it necessary to lock all instances of unit functions in the entire netlist to achieve a secured logic locking scheme. CLIC-A [23, 24], an ATPG-based attack, can break keys by applying constraint-based ATPG to propagate the target key bit to the primary output but suffers scalability in the dependent key count. Cyclic-based locking has suffered from modified SAT-based attack [72], where cyclic-based constraints are placed to avoid infinite loops. Several non-functional-based locking can be broken by sequential-based attacks with limited scan access [26, 40, 44] or SMT attack [5].

2.1.1 Comparison of AFIA with CLIC-A [23]

There is a major difference between our proposed *AFIA* and *CLIC-A*. First, the worst-time complexity for test generation regarding the total test pattern count for solving the key-dependent faults between *AFIA* and *CLIC-A* differs significantly. Our worst-case complexity of solving an n-bit key of non-mutable convergent key gates inside a single logic cone is at most n test patterns with $\frac{n\cdot(n-1)}{2}$ injected faults (see Theorem 3). This is because AFIA determines each key bit by directly comparing the output response with the generated test pattern. On the other hand, *CLIC-A* applies constraint-based ATPG by assigning constraint on the $(n-1)$-bit key and setting a stuck-at 0 at the target key line since placing don't cares (X) on other key bits will not produce the desired test patterns for non-mutable convergent key gates. However, the simulated output from the constraint-based ATPG likely agrees with the oracle simulation under the same test pattern hardness of logic locking. Note that it does not mean that the constraints placed on the $(n-1)$-bit key is the correct key values, and it only indicates that, under the particular test pattern, the output from the netlist with constraints and the stuck-at fault matches with the oracle output. Instead, *CLIC-A* has to perform additional constraints in ATPG and check the output against the oracle to ensure that key values are correct. The worst-case complexity in the total test pattern count for *CLIC-A* is exponential $\boldsymbol{O}(2^{(n-1)})$.

Let us consider an example of an unlocked circuit in Fig. 2a locked with 6 dependent XOR/XNOR key gates, as

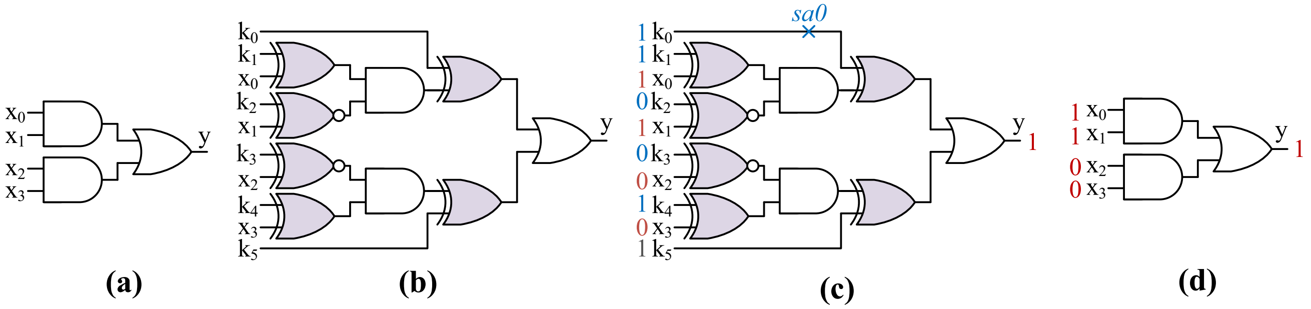

Fig. 2 The inefficiency of CLIC-A attack. **a** Original circuit. **b** Locked circuit with 6 dependent key gates, where correct key $\{k_0, ..., k_5\} = \{001100\}$. **c** Key assignment and input pattern returned by Constraint-based CLIC-A. **d** The oracle response $y = 1$

shown in Fig. 2, whose correct key $\{k_0, ..., k_5\} = \{001100\}$. As none of the keys can be sensitized to the output without knowing the correct value for the other five, CLIC-A runs constraint-based ATPG and sets *sa0* to k_0. Suppose ATPG returns a test pattern with key vector $\{k_0, ..., k_5\} = \{110011\}$ and input vector $\{x_0, ..., x_3\} = \{1100\}$, along with the simulated fault-free output `atpg-sim`$(x_0, ..., x_3, k_0, ..., k_5) = 1$, as shown in Fig. 2c. Although the output of the simulated netlist matches with the oracle response $y = 1$, the key value returned by constraint-based ATPG is incorrect. There is no method for CLIC-A to check whether the key vector is the actual key other than appending it as a constraint to ATPG so that the test pattern returned at the next iteration would be different from the current one. The worst-case complexity for CLIC-A to fully determine the 6-bit key is to iterate through all possible combinations of the remaining 5-bit key (excluding k_0 with *sa0*), resulting in a 2^5 test pattern count to break the locking scheme with dependent keys. On the other hand, AFIA only requires 6 test patterns to determine all 6-bit keys, which is much more efficient than CLIC-A. In summary, test pattern generation for CLIC-A becomes infeasible if there are a large number of dependent keys in a logic cone.

2.1.2 Comparison of AFIA with Key Sensitization Attack [55]

There is a similarity between our proposed AFIA and sensitization attack [55]. The similarity between these approaches is the sensitization, i.e., the propagation, of the key to the output. However, our approach is more general for the following reason. First, the sensitization attack does not need fault activation as the key gates are XOR/XNOR gates, and the key can propagate to the key gate output for both input 0 and 1. However, this may not hold for non-XOR-based locking techniques. For example, MUX-based locking has keys connected to the input of AND gate instead of the XOR gate, where one needs to set the other AND input to the non-controlling value 1 for fault activation. Besides, it is common practice for recent locking techniques to synthesize the locked benchmark after key insertion. The synthesis tool can optimize the key gate with other gate types, which results in keys directly connected to non-XOR gates like AOI, NAND, etc. To propagate the key value to the primary output, having only key sensitization without the activation would not work for synthesized locked circuits. For example, we can break *SFLL-hd* [98], *SFLL-flex* [98], and *SFLL-rem* [69] with n patterns for a n-bit key (see Sect. 4.7), where sensitization attack requires brute force attack ($\boldsymbol{O}(2^n)$) to all the non-mutable keys in the *SFLL* restoration circuitry. Second, our proposed fault injection can break non-mutable convergent key gates from strong logic locking, which is the countermeasure proposed in a sensitization attack. AFIA only needs at most n (see Theorem 1) test patterns for a n-bit pairwise non-mutable convergent keys, but it would take $\boldsymbol{O}(2^n)$ in the worst case to brute force the correct key under sensitization attack [55].

2.1.3 Dissimilarities Between Logic Locking and Cryptosystems

There has been considerable efforts [12, 98] in the proposal of formal analysis on logic locking through introducing similar concepts used in cryptography. However, logic locking techniques differ from various cryptosystems in two aspects. First, the objective for logic locking and cryptosystem is different. The cryptographic algorithm ensures that the secret key is fully integrated with the input plaintext (i.e., the addRoundKey in all ten rounds of AES encryption). Logic locking, however, focuses on perturbing the output, commonly by XORing a 1-bit key with a wire in the circuit, under certain input patterns, where no repeated insertion of the same key bit or its derived value to elsewhere. Second, the output of a locked circuit and the ciphertext of a cryptosystem behaves differently under input combinations. A locked circuit under an incorrect key may behave identically to the oracle (or locked circuit with the correct key) under multiple input patterns. This is particularly true for Post-SAT locking solutions, i.e., SARLock [96], Anti-SAT [94], SFLL [68, 98], CAS-Lock [70], where the output corruptibility for incorrect keys is reduced to the bare minimum. This means that a locked circuit with an incorrect key behaves exactly as an unlocked circuit under an exponential number of input combinations.

The cryptographic algorithms, especially the block ciphers, are built on confusion and diffusion properties recommended by Claude Shannon in his classic 1949 paper [74]. This results in a large number of output bit changes in the output (ciphertext) even for a single bit change in the input (plaintext) [25, 45]. For example, AES has 10/12/14 rounds of diffusion and confusion operations depending on the key size of 128/192/256 bits. It is thus trivial to launch differential fault analysis as it will guarantee the change in the output, where one can compare the faulty and fault-free responses by injecting a fault into a key register, one at a time. On the contrary, digital circuits generally do not have repeated layers of operations like block ciphers. Digital circuits, except crypto accelerators, are designed to meet the user specification of speed, power, and area, and the functionality (change in output) depends on the user's needs. It is well understood and verified that digital circuits have lots of don't cares (*Xs*) in the inputs. The VLSI test community adopted test compression [57, 86] to reduce the test pins and resultant test times. As there exists a large number of *Xs* in the test pattern, it is infeasible to apply a random pattern and expect it to propagate the target key bit (e.g., a stuck-at fault at the key line) to output. For example, if there are 70% Xs in a test pattern with a 100 input cone [which is very common], the probability of a random pattern propagating the key to the output is $2^{30}/2^{100} \approx 0$. The effect of some keys in a locked circuit can even be muted due to the circuit's structural and functional behavior [55], which is in direct contrast to cryptosystems, where every output is influenced by all key bits [45].

2.2 Fault Injection Methods

Over the years, several threats and methods have emerged to break a cryptosystem without performing mathematical analysis or brute force attacks. Using these attacks, an adversary can subvert the security of protection schemes, primarily through extracting or estimating the secret key using physical attacks. Fault injection attacks intentionally disturb the computation of cryptosystems in order to induce errors in the output response. To achieve this, external fault injection is performed through invasive or non-invasive techniques. This is followed by the exploitation of erroneous output to extract information from the device.

Fault-based analysis on cryptosystems was first presented theoretically by Boneh et al. on RSA [14]. This contribution initiated a new research direction to study the effect of fault attacks on cryptographic devices. The comparison between the correct and faulty encryption results has been demonstrated as an effective attack to obtain information regarding the secret key [22, 42, 47]. These can be realized into different categories:

- *Clock Glitch*: The devices under attack are supplied with an altered clock signal which contains a shorter clock pulse than the normal operating clock pulse. For successfully inducing a fault, these clock glitches applied are much shorter than the circuit's tolerable variation limit for the clock pulse. This results in setup time violations in the circuit and skipping instructions from the correct order of execution [27, 64].
- *Power Variation*: This technique can be further bifurcated into two subcategories: either the malicious entity may choose to provide a low power supply to the system (also abbreviated as underfeeding), or the adversary may choose to influence the power line with spikes. This adversely affects the set-up time and influences the normal execution of operations. The state elements in the circuit are triggered without the input reaching any stable value, causing a state transition to skip operations or altering the sequence of execution [8, 9, 28].
- *Electromagnetic Pulses/Radiation*: The eddy current generated by an active coil can be used to precisely inject faults at a specific location in the chip. This method does not require the chip to be decapsulated in order to inject the fault. However, the adversary is required to possess information regarding specific modules and their location inside the chip [21, 63].
- *Laser*: Fault injection using lasers is also regarded as a very efficient method because it can precisely induce a fault at an individual register to change its value [10]. For optical fault injection, the laser can be focused on a specific region of the chip from the backside or front side. However, due to the metal layers on the front side, it is preferred to perform the attack on the backside of the chip. Skorobogatov and Anderson [80] first demonstrated the effectiveness of this method by using a flashgun to inject fault to flip a bit in the SRAM cell. Several other research groups also utilized and proposed different variants of this method to study the security of cryptographic primitives [16, 48, 65, 79].
- *Focused-ion Beam (FIB)*: The most effective and expensive fault injection technique is devised with focused ion beam (FIB) [87]. This method enables cutting/connecting wires and even operates through various layers of the IC fabricated in the latest technology nodes [91].
- *Software Implemented Fault Injection*: This technique produces errors through software that would have been produced when a fault targeted the hardware. It involves the modification of programs running on the target system to provide the ability to perform the fault injection. It does not require dedicated complex hardware, a gate-level netlist, or RTL models that are described in hardware description languages. The faults are injected into accessible memory cells such as registers and memories through software that represent the most sensitive zones of the chip [32, 78, 88].

3 Background

In this section, we present a differential fault analysis (DFA) attack introduced in [33]. Our attack method is inspired by VLSI test pattern generation. One test pattern is able to detect a single stuck-at fault with the propagation of this fault to the primary output. Since key values from tamper-proof non-volatile memory are loaded to key registers, these registers are the potential locations for stuck-at-faults. With an active chip at hand, the adversary could target these registers and extract the secret key.

3.1 Threat Model

The threat model defines the capabilities of an adversary and its standing in the IC manufacturing and supply chain. It is very important to know an attacker's ability and the available resources/tools to estimate its potential to launch the attack. The design house or entity designing the chip is assumed to be trusted. The attacker is assumed to be the untrusted foundry or a reverse engineer having access to the following:

- The locked netlist of a circuit. An untrusted foundry has access to all the layout information, which can be extracted from the GDSII or OASIS file. Also, this locked netlist can be reconstructed by reverse engineering the fabricated chip in a layer-by-layer manner with advanced technological tools [87].
- An unlocked and fully functional chip is accessible to the adversary since the chip is publicly available from the market.
- A fault injection equipment is essential to launch the attack. It is not mandatory to use high-end fault injection equipment. The main operation is to inject faults at the locations of key registers (all the flip-flops) on a de-packaged/packaged chip. Precise control is not necessary as we target all the flip-flops simultaneously. An adversary can also choose the software methods to inject faults at these flip-flops. Once the register is at the faulty state, the scan enable (*SE*) signal needs to be assigned to put the chip in test mode.
- The attacker has the know-how to determine the location of the tamper-proof memory. Then, it will be trivial for an adversary to find the location of the key register in a netlist, as it can easily trace the route from the tamper-proof memory.

Notations To maintain uniformity across the entire paper, we represent frequently used terms with the defined notations, and they will be referred to with these notations in the following subsections.

- $\mathcal{K}$ denotes key length or key size, i.e., the number of bits in the key.
- K denotes the keyspace; $K = \{k_0, k_1, \dots k_{\mathcal{K}-1}\}$.
- The locked netlist of a circuit is abbreviated as C_L. The unlocked and fully functional chip/circuit, whose tamper-proof memory has been programmed with the correct key, is denoted by C_O. The two versions of fault-injected circuits are described as follows:
    - C_F represents a locked circuit where all the key lines ($\mathcal{K}$) are injected with logic 1 (or logic 0) faults. We call it the circuit with faulty key registers for differential fault analysis (DFA).
    - C_A represents the same locked circuit in which ($\mathcal{K} - 1$) key lines are injected with the same logic 1 (or logic 0) faults, leaving one key line fault-free. We denote this circuit as a fault-free circuit for DFA.

  For any given circuit, we assume the primary inputs (*PI*) of size |*PI*|, primary outputs (*PO*) of size |*PO*|, and secret key (K) size of $\mathcal{K}$. We also use key lines or key registers alternatively throughout this paper as their effects are the same on a circuit.
- Stuck-at fault (*saf*): For any circuit modeled as a combination of Boolean gates, stuck-at fault is defined by permanently setting an interconnect to either 1 or 0 in order to generate a test vector to propagate the fault value at the output. Each connecting line can have two types of faults, namely, stuck-at-0 (*sa0*) and stuck-at-1 (*sa1*). Stuck-at faults can be present at the input or output of any logic gates [15].
- Injected fault: A fault is injected at the key register using a fault injection method (see details in Sect. 2).

Note that *saf* is an abstract representation of a defect to generate test patterns, whereas an injected fault is the manifestation of a faulty logic state due to fault injection.

3.2 Differential Fault Analysis (DFA) Attack Methodology

This fault injection attack relies on differential fault analysis. The captured output response of the circuit with faulty key registers with the corresponding fault-free circuits can reveal the key. Applying any fault injection methods (see the details in Sect. 2.2), the attacker can create the faulty chip/circuit. Figure 3 shows an abstract representation of DFA. The fault-free circuit (C_A) is an unlocked chip (C_O) bought from the market whose key bits need to be retrieved. Except for the key-bit targeted to be extracted, all remaining key registers are fixed to a particular faulty value of either 0 or 1 corresponding to the selected fault. A circuit with faulty key registers (C_F) uses the same chip, and it is injected with a particular fault to keep all the key registers or interconnects to a faulty value of logic 1 or 0. One input pattern is first applied to C_A, and its response is collected. The same input pattern is then applied to the

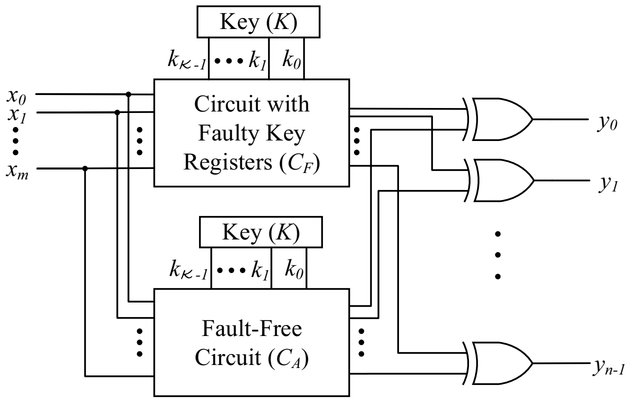
Key (K)
$k_{\mathcal{K}-1}$ • • • k_1 k_0
x_0
x_1
x_m
Circuit with Faulty Key Registers (C_F)
Key (K)
$k_{\mathcal{K}-1}$ • • • k_1 k_0
Fault-Free Circuit (C_A)
y_0
y_1
y_{n-1}

Fig. 3 The abstract representation of our DFA attack

C_F to collect the faulty response. By XORing the corresponding circuit response, any output discrepancy between fault-free circuit (C_A) and the circuit with faulty key registers (C_F) is revealed. If both the circuits differ in their responses, the XORed output will be 1; otherwise, it will be 0. If we find an input pattern that produces a conflicting result for both C_A and C_F only for one key bit, the key value can be predicted. The key value is the same as the injected fault value if the XORed output is of logic 0; otherwise, the key value is a complement to the injected fault.

The attack can be described as follows:

- *Step-1*: The first step is to select an input pattern that produces complementary results for the fault-free (C_A) and faulty (C_F) circuits. The input pattern needs to satisfy the following property – it must sensitize only one key bit to the primary output(s). In other words, only the response of one key bit is visible at the *PO*, keeping all other key bits at logic 1s (or 0s). If this property is not satisfied, it will be impractical to reach a conclusion regarding the value of a key bit. *Now the question is, how can we find if such a pattern exists in the entire input space* (ξ). To meet this requirement, our method relies on stuck-at faults (*saf*) based constrained ATPG to obtain the specific input test patterns (see details in Sect. 3.4). Considering the fact that the adversary has access to the locked netlist, it can generate test patterns to detect *sa1* or *sa0* at any key lines and add constraints to other key lines (logic 1 and 0 for *sa1* and *sa0*, respectively). A single fault, either *sa0* or *sa1* on a key line, is sufficient to determine the value of that key bit. Therefore, we have selected *sa1*, and the following subsections are explained considering this fault only. This process is iterated over all the key bits to obtain $\mathcal{K}$ test patterns. The algorithm to generate the complete test pattern set is provided in Algorithm Sect. 3.4.

- *Step-2*: The complete set of generated test patterns is applied to the fault-induced functional circuit with faulty key registers (C_F). The circuit is obtained by injecting logic 1 fault on the key registers if *sa1* is selected in the previous step; else, the circuit is injected with logic 0 faults for *sa0*. The responses are collected for later comparison with fault-free responses. For C_A, test patterns are applied such that it matches the fault modifications in the circuit. For example, the test pattern for the first key is applied to the circuit when the circuit instance does not pertain to any fault on its corresponding key register and holds the correct key value while the remaining key registers are set to logic 1 (for *sa1*) or 0 (for *sa0*). For the next key-bit, (C_A) instance is created by excluding this selected key bit from any fault while keeping all other key registers to logic 1 (for *sa1*) or 0 (for *sa0*). This process is repeated for all key bits, and their responses are collected for comparison in the subsequent step.

- *Step-3*: The adversary will make the decision regarding the key value from the observed differences in the output responses of (C_A) and (C_F). For any test pattern corresponding to a particular key bit, when the outputs from both circuits are the same, it implies that the injected fault on the key lines in a C_F circuit is the same as the correct key bit; only then will the outputs of both ICs be same. Otherwise, when C_F and C_A differ in their output response, it concludes the correct key bit is a complement to the induced fault. This process is repeated for all key bits. In this manner, the key value can be extracted by comparing the output responses of both circuits for the same primary input pattern.

3.3 Example

We choose a combinational circuit as an example for simplicity to demonstrate the attack. The attack is valid for sequential circuits, as well, as it can be transformed into a combinational circuit in the scan mode, where all the internal flip-flops can be reached directly through the scan-chains [15].

Figure 4 shows the test pattern generation on a circuit locked with a 3-bit secret key, where the propagation of k_0 is dependent on k_1 and vice versa. First, we target to find out the value of k_0. A test pattern P_1 is generated to detect a *sa1* fault at k_0 with constraint $k_1 = 1$ and $k_2 = 1$ (adding faults on all the key lines except the target key bit). As the value of k_1 is known during the pattern generation, the effect of the *sa1* at k_0 will be propagated to the primary output y_0. For a fault value $\overline{D}$ at k_0, if $[x_0\ x_1] = [1\ 1]$ then D propagates to n_2. To propagate the value at n_2 to the output of G_3, its other

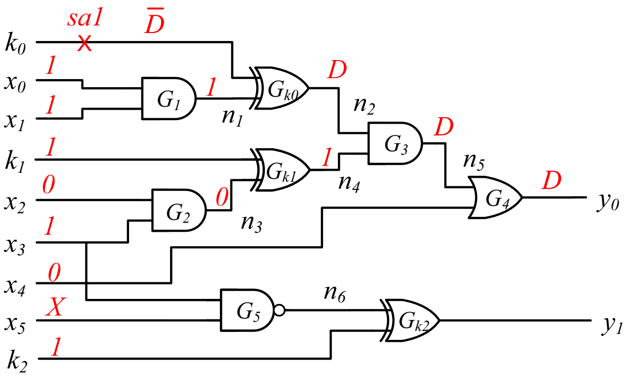

Fig. 4 Test pattern generation considering a *sa1* at key line k_0 with constraint $k_1 = 1$ and $k_2 = 1$. Test pattern, $P_1 = [11010X]$ can detect a *sa1* at k_0

input (n_4) needs to attain logic 1. Since $k_1 = 1$ due to injected fault which is set as a constraint in ATPG tool, $n_4 = 1$ for $n_3 = 0$ which implies $[x_2\ x_3] = [0\ 1]$. At last, $x_4 = 0$ propagates D propagates the value at n_5 to the primary output y_0. The output y_0 can be observed as D for the test pattern $P_1 = [1\ 1\ 0\ 1\ 0\ X]$. Finally, to perform the DFA, this pattern P_1 needs to be applied to both C_A and C_F to determine the value of k_0. Similar analysis can be performed for the other two key bits, k_1 and k_2.

3.4 Test Pattern Generation

To generate the test pattern set, an automated process relying on constrained ATPG is performed. The detailed steps to be followed are provided in Algorithm 1. Synopsys Design Compiler [83] is utilized to generate the technology-dependent gate level netlist and its test protocol from the RTL design. A test protocol is required for specifying signals and initialization requirements associated with design rule checking in Synopsys TetraMAX [84]. Automatic test generation tool TetraMAX generates the test patterns for the respective faults along with constraints for the locked gate level netlist.

Algorithm 1: Test pattern generation for constrained ATPG in DFA

```
Input  : Locked gate-level netlist ($C_L$), test protocol ($T$), and
         standard cell library
Output: Test pattern ($P$) set
1  Read the locked netlist ($C_L$) ;
2  Read standard cell library ;
3  Run design rule check with test protocol generated from
     design compiler ;
4  Determine key size $\mathcal{K}$ from $C_L$ ;
5  for $i \leftarrow 0$ to $(\mathcal{K} - 1)$ do
6      Add a $sa1$ fault at key line $k_i$ ;
7      for $j \leftarrow 0$ to $(\mathcal{K} - 1)$ do
8          if $i \neq j$ then
9              Add constraint at $k_j$ to logic 1 ;
10         end
11     end
12     Run ATPG to detect the fault ;
13     Add the test pattern, $P_i$ to the pattern set, $P$ ;
14     Remove all faults ;
15     Remove all constraints ;
16 end
17 Report the test pattern set, $P$ ;
```

The inputs to the algorithm are the locked gate-level netlist (C_L), Design Compiler generated test protocol (T), and the standard cell library. The algorithm starts with reading the locked netlist and standard cell library (Lines 1-2). The ATPG tool runs the design rule check with the test protocol obtained from the Design Compiler to check for any violation (Line 3). Only upon the completion of this step is the fault model environment set up in the tool. The size of the key ($\mathcal{K}$) is determined by analyzing C_L (Line 4). The remaining key lines are selected one by one to generate test patterns (Line 5). A stuck-at-1 fault is added at the i^{th} key line to generate P_i (Line 6). The ATPG constraints (logic 1) are added to other key lines (Lines 7-11). A test pattern P_i is generated to detect the $sa1$ at the i^{th} key line (Lines 12-13) and added to the pattern set, P. All the added constraints and faults are removed to generate the $(i + 1)^{th}$ test pattern (Lines 14-15). Finally, the algorithm reports all the test patterns, P (Line 17).

4 AFIA: ATPG-Guided Fault Injection Attack

The objective of an adversary is to reduce the number of injected faults to launch an efficient attack. The DFA presented in Sect. 3.2 requires $2\mathcal{K} - 1$ faults to determine a single key bit, where $\mathcal{K}$ denotes the secret key size. This severely limits the adversary's capability as injecting a large number of faults is challenging from the fault injection equipment's perspective. All these faults need to be injected when applying the test pattern to evaluate one key bit. In this section, we present an efficient attack and denoted as ***AFIA***, an ***A***TPG-guided ***F***ault ***I***njection ***A***ttack based on key sensitization. This new attack only requires injecting the fault on a key register if there is a dependency among keys. The threat model remains the same as DFA. We consider an untrusted foundry to have access to the gate-level netlist and can generate manufacturing test patterns.

4.1 Overall Approach

The proposed attack AFIA evaluates one key bit at a time iteratively and can be summarized by the following steps:

- *Step-1*: First, AFIA analyzes the locked circuit C_L and its logic cones. Some cones are completely independent (e.g., LC_0 in Fig. 5), some cones share few inputs (e.g., LC_1, ..., LC_{p-1}), and the others share the same inputs (e.g., LC_p, ..., LC_{N-1}). It is necessary to determine keys from cones that are a subset of other larger cones (if any) first during the test pattern generation in order to reduce the number of injected faults. For an independent logic cone (say LC_0), we can propagate the keys one at a time without injecting faults at keys of other cones. If the two cones are overlapped, it is beneficial to sensitize keys to a cone with fewer unknown keys.

- *Step-2*: Similar to DFA, it requires an input pattern to derive a correct key bit. We denote this key bit as the target key bit. Constraints are set on the recovered key lines, where no fault injections are needed. The attacker performs fault injection (*Step-3*) solely on keys (in the

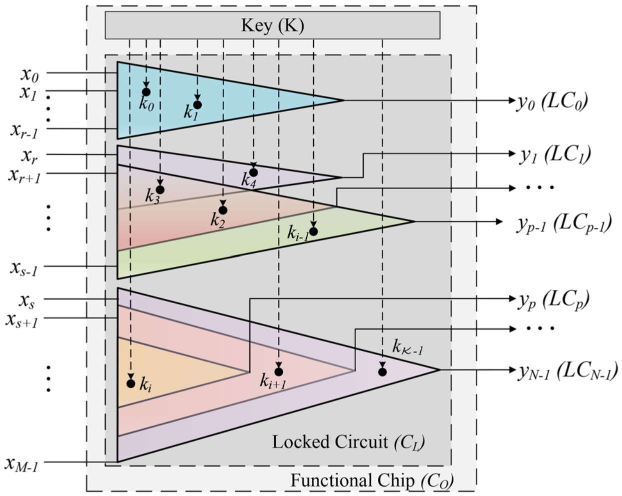

Fig. 5 An abstract view of a locked circuit

same cone) that block the propagation of the targeted key bit. The blocking key set is determined by the returned test patterns from ATPG TetraMAX [84]. Once a key bit is determined, AFIA targets the next key bit of the same cone by putting the previously obtained keys as constraints during the test pattern generation.

- *Step-3*: The last step applies fault injections on functional chip C_O using the generated test patterns of *Step-2*. The targeted key value can be extracted by comparing the fault-injected output against the output pattern computed by ATPG. When the value of all the targeted key bits in one text pattern has been identified, we can constrain these bits with their actual values in ATPG in the subsequent pattern.

AFIA is an iterative method, where *Step-2* is performed to generate test patterns, and *Step-3* injects fault and applies that pattern to determine the targeted key bit. Once this targeted key is determined, it will be used as a constraint in *Step-2*. The following subsections present these three steps in detail.

4.2 Cone Analysis

The goal of this proposed attack is to apply minimal fault injections to recover the complete key set. It is ideal for the adversary to inject faults at key registers only when necessary. In general, not all keys prevent the propagation of the target key bit, as many of the keys are often distributed across the netlist and reside in different logic cones. A logic cone is a part of the combinational logic of a digital circuit that represents a Boolean function and is generally bordered by an output and multiple inputs [15]. Thus, cone analysis can effectively separate the dependence of different groups of key bits, where one group does not block the propagation of the key bits in other groups. We propose to analyze the internal structure of the locked netlist C_L by creating a directed graph G from it. We denote that both the inputs and logic gates' outputs are nodes. A directed edge exists from Node n_1 to Node n_2 if and only if they are associated with a logic gate. Intuitively, a circuit with N outputs has N logic cones, as in Fig. 5. Note that the number of cones can be only primary outputs (POs) for a combinational circuit or the sum of POs and pseudo primary outputs (PPOs) for a sequential circuit [15]. All the inputs and logic gates whose logical values affect y_j belong to logic cone LC_j. The graph representation of logic cone LC_j with sink y_j is a subgraph of G.

Two possible scenarios might occur during the locking of a netlist. Key bit(s) can be placed uniquely in a logic cone and cannot be sensitized to any other POs/PPOs except the cone's output. Other key bits can be placed in the intersection of multiple cones and can be sensitized through any of these. We observe that the majority of the key bits are inside the intersections with multiple cones. What should be the best strategy to propagate a key bit to one of the POs/PPOs when there exist multiple sensitization paths? Our objective is to reduce the number of faults to sensitize a key bit to a PO/PPO, and it is beneficial to select a cone with the minimum number of keys. Note that the keys in a cone can block the propagation of a targeted key in that same cone only and requires fault injection to set a specific value to these blocking keys. It is, thus, necessary to construct a key-cone association matrix A to capture the correlation between the logic cones and the key bits. The matrix A not only provides insight on which keys (and how many of them) are inside a logic cone but also offers a structured view of whether a key belongs to multiple logic cones, and is presented as follows:

$$A = [a_{i,j}]_{\mathcal{K}\times N}$$

$$= \begin{array}{c} \\ k_0 \\ k_1 \\ k_2 \\ \vdots \\ k_{\mathcal{K}-1} \end{array} \begin{array}{c} \begin{array}{cccc} LC_0 & LC_1 & \dots & LC_{N-1} \end{array} \\ \begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,N-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,N-1} \\ a_{2,0} & a_{2,1} & \cdots & a_{2,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{\mathcal{K}-1,0} & a_{\mathcal{K}-1,1} & \cdots & a_{\mathcal{K}-1,N-1} \end{pmatrix} \end{array},$$

where, $a_{i,j} \in \{0, 1\}$, and $a_{i,j} = 1$ if key k_i is present in cone LC_j, otherwise, $a_{i,j} = 0$.

It is straightforward for the attacker that, if he/she picks cone LC_j and key bit k_i (if its value is still unknown) in this cone, only keys (other than k_i) residing in LC_j could potentially impede the propagation of k_i to the output y_j. This is advantageous to the attacker because the keys outside of cone LC_j would not, by any means, affect the propagation of k_i to y_j. Thus, he/she can safely ignore these keys, and it does not matter whether he/she already has the correct logical values for them or not.

For example, the directed graph representation of locked netlist c432-RN320 with a 32-bit key [61] is shown in Fig. 6. Output nodes are in red, key registers in green (at the left-most level), key gates in cyan, remaining input (at the left-most level), and gates in blue. The top two logic cones with the fewest keys are LC_{N223} of output $N223$ and LC_{N329} of output $N329$. Logic cone LC_{N223} has only one key (*keyIn_0_4*, with key gate highlighted) (all other nodes and edges are in magenta and light green). Logic cone LC_{N329} is the superset of LC_{N223}, and it contains additional thirteen keys (all other nodes and edges exclusively in LC_{N329} are in purple and orange). With AFIA, the only key in LC_{N223} is determined first, followed by the remaining thirteen keys in LC_{N329}. Because of the only key in LC_{N223}, no fault injection is necessary for this key's propagation to $N223$.

4.3 Test Pattern Generation

Once the cone analysis is performed, it is required to generate test patterns so that a targeted key can be sensitized to one of the PO/PPO. The test pattern generation process is similar to the DFA presented in Sect. 3.2 except with a much lesser number of ATPG constraints. We treat undetermined keys as inputs during the test pattern generation and the recovered keys as ATPG constraints. As the secret key remains the same in an unlocked chip, it is unnecessary to inject faults at the recovered key bits as their values are known during the test pattern generation. On the other hand, we need to inject faults at unknown and yet to be determined key lines. However, it is not necessary to inject faults at all of them. We use the ATPG tool to determine whether one or more unknown key bits do not block the propagation of the targeted key bit. As we treat unknown keys as inputs, the ATPG tool can generate a pattern that might contain $X's$ at some of the key lines (using *set_atpg -fill X* [84]), and we do not need to inject faults at these bits. This allows an adversary to reduce the number of fault injections further. Similar to DFA, a stuck-at fault, *sa1* (or *sa0*), is placed on the target key bit with constraints on recovered key bits during the ATPG. When TetraMAX [84] returns a test pattern, the attacker applies the pattern and injects faults (presented in Sect. 4.4) to sensitize the target key bit at the PO/PPO. After recovering one key bit, AFIA sets ATPG constraints on the recovered key lines, generates another test pattern, and applies it to sensitize the next key.

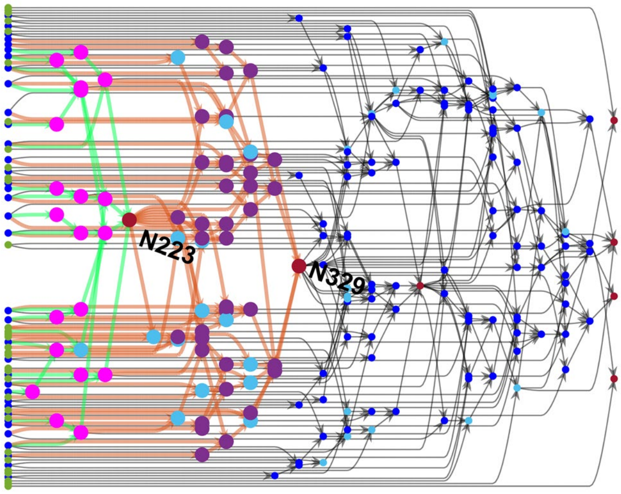

Fig. 6 Directed graph of locked c432-RN320 netlist with a 32-bit key

4.4 Fault Injection

The final step applies fault injections on functional chip C_O using generated test patterns from Sect. 4.3. Faults are injected at the key registers with any appropriate fault injection techniques described in Sect. 2.2. No fault injection is necessary at the key bits whose values are already determined as their values are no different from those already programmed in the chip C_O. If we receive a faulty response by applying the test pattern developed in *Step-2*, the value of the secret key will be 1 as we have sensitized a *sa1* fault during the ATPG; otherwise, the secret key is 0. If we generate a test pattern considering a *sa0* fault, the faulty response results in the secret key of 0, and vice versa. *Step-2* in Sect. 4.3 and *Step-3* in Sect. 4.4 are repeated until the entire secret key is found. Consequently, fewer faults are injected compared with the DFA since injections happen only at key locations (of the same logic cone) that block the propagation of the to-be-determined key bits.

4.5 Proposed Algorithm for AFIA

Algorithm 2 describes the implementation details of AFIA. The adversary first constructs a directed graph G from the locked netlist C_L (Line 1), as elaborated in Sect. 4.2. Aside from converting netlist to graph, `netlist2Graph(.)` returns the key list K and output list Y. By exploiting directed graph structure, logic cone LC_j can be easily extracted by flipping all edges in graph G (Line 2) and run breadth-first-search (BFS) or depth-first-search, (DFS) [20], on output nodes y_j. The key-cone association matrix A is declared as an empty array, where the cone and key information will be added (Line 3). Function `extractCone(.)` is implemented with BFS. It returns the directed subgraph of logic cone LC_j and a logical (**true/false**) vector LK_j of dimension $\mathcal{K} \times 1$. If key bit k_q is inside cone LC_j, $LK_j[q] = \textbf{true}$; else,

$LK_j[q]$ = **false**. Matrix A is updated by concatenating all vectors LK_j's together (Line 6) so that the complete A has $\mathcal{K}$ rows and N columns, as explained in Sect. 4.2.

Algorithm 2: AFIA: ATPG-guided Fault Injection Attack.

```
   Input  : Locked gate-level netlist ($C_L$)
   Output: Secret key ($KEY$)
1  // ———————- Cone Analysis
2  $[K, Y, G] \leftarrow$ netlist2Graph($C_L$);
3  $Gflip \leftarrow$ flipEdges($G$);
4  $A \leftarrow [\,]$;
5  for each $y_j$ in Y do
6      $[LK_j, LC_j] \leftarrow$ extractCone($Gflip$, $K$, $y_j$);
7      $A \leftarrow$ append vector $LK_j$ as the last column;
8  end
   // ————- ATPG test pattern generation
9  Recovered key bits from Step-3 of AFIA, $K^R \leftarrow \varnothing$;
10 while ($A$ != false) do
11     $[K^U_{LC}] \leftarrow$ fConeWMinKeys($A$,$K$);
12     if $K^U_{LC} != \varnothing$ then
13         for $l \leftarrow 0$ to $(|K^U_{LC}| - 1)$ do
14             Add a $sa1$ fault at key line $K^U_{LC}[l]$;
15             Add constraints at recovered key bits to $K^R$;
16             Test pattern $P_l \leftarrow$ run ATPG (set_atpg -fill X);
17             Remove all faults;
18             Remove all constraints;
               // ————- Fault Injection
19             Invoke Step-3 of AFIA with $P_l$;
20             Add recovered key $K^U_{LC}[l]$ to $K^R$;
21             Assign false to all entries in key $K^U_{LC}[l]$'s row in
                 $A$;
22         end
23     end
24 end
25 Report the secret key, $KEY \leftarrow \{K, K^R\}$;
```

AFIA invokes fConeWMinKeys(.) (Line10) and obtains a vector K^U_{LC} of all unknown keys in the logic cone with the fewest (positive) unknown keys. For simplicity, K^U_{LC} records the row indices of the unknown keys, as in matrix A. For every key bit in K^U_{LC}, the $sa1$ is set on the to-be-determined key (Line 13). The recovered key values in K^R are appended as constraints (Line 14). Test pattern P_l (Line 15) is generated after invoking ATPG. All the stuck-at faults (Line 16) and constraints (Line 17) are removed. When P_l and fault injections (Line 18) are applied on the working chip C_O, $K^U_{LC}[l]$ bit is recovered by referencing the ATPG's predicted output of the corresponding P_l. Afterward, the correct bit value is added to the recovered key list K^R (Line 19). Since this bit is recovered, it is no longer an unknown key, and AFIA updates the association matrix A to assign logical zero to all entries on key $K^U_{LC}[l]$'s row (Line 20). This is conceptually equivalent to deleting $K^U_{LC}[l]$ from the unknown key list as fConeWMinKeys(.) will only count the number of non-zero entries per column. When all key bits in K^U_{LC} are determined, the adversary moves on to the subsequent logic cone (Line 10). Finally, when all cones are covered, the secret key KEY is returned (Line 24).

4.6 Example

Here, we use the same circuit as in Fig. 4 as an example to illustrate how AFIA works. The circuit has six inputs, two outputs, and three key bits. With two outputs, this circuit has two logic cones, as in Fig. 7. The same D-Algorithm [15] is applied to show the propagation of stuck-at-faults. Based on cone analysis in Sect. 4.2, logic cone LC_0 contains two key bits, k_0, k_1, cone LC_1 has only one key k_2. Thus, the association matrix A can be represented as:

$$A = \begin{array}{c c} & \begin{array}{cc} LC_0 & LC_1 \end{array} \\ \begin{array}{c} k_0 \\ k_1 \\ k_2 \end{array} & \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array}.$$

AFIA picks a logic cone with the fewest number of unknown keys to solve (Line 10, Algorithm 2). Since all keys

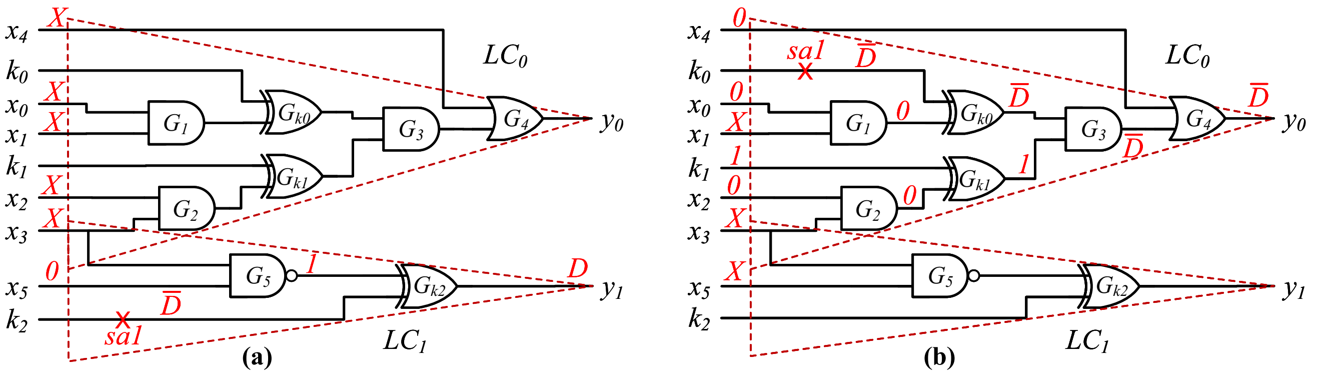

Fig. 7 Test Pattern Generations for AFIA. **a** Test Pattern $P_0 = [XXXXX0]$ for $sa1$ at k_2. **b** Test Pattern $P_1 = [0X0X0X]$ for $sa1$ at k_0 with injected fault $k_1 = 1$

are unknown at this time, `fConeWMinKeys(.)` function selects logic cone LC_1 and returns $K_{LC}^U = [2]$. This cone has one key bit k_2, to which we assign *sa1*. Using D-Algorithm, fault value $\overline{D}$ is marked on this key line. Here, the output y_1 is directly connected to XOR key gate G_{k_2}, and we can propagate this fault $\overline{D}$ to output $y_1 = D$ with logic 1 for the other input of this XOR gate, as in Fig. 7a. Test pattern $P_0 = [x_0 x_1 \dots x_5] = [XXXXX0]$ can detect *sa1* for key k_2. Here, the value of the recovered key is 1 when the output is faulty. Otherwise, the recovered key is 0 as we have sensitized a *sa1* fault during the ATPG. Note that no fault injection is necessary to determine this key. Matrix A is updated with all zeros on the k_2's row,

$$A = \begin{array}{c c} & \begin{array}{cc} LC_0 & LC_1 \end{array} \\ \begin{array}{c} k_0 \\ k_1 \\ k_2 \end{array} & \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} \end{array}.$$

In the next iteration (Line 10), there is only one logic cone (also the cone with the least unknown keys), LC_0, left in matrix A that has unknown keys. Function `fConeWMinKeys(.)` identifies LC_0 and yields $K_{LC}^U = [0\ 1]^T$, which captured the indices of unknown keys k_0, k_1. With two keys k_0 and k_1, AFIA chooses k_0 first randomly (Algorithm 2 Line 13). By adding *sa1* at k_0, test pattern $P_1 = [x_0 x_1 \dots x_5] = [$ *0X0X0X*$]$ with logic *1* fault on k_1 can propagate the faulty response $\overline{D}$ in k_0 to y_0, as shown in Fig. 7b. Fault injection is performed at k_1 by setting its value to 1, and apply P_1 to determine k_0. AFIA, then, flushes out all the entries on row k_0 of matrix A,

$$A = \begin{array}{c c} & \begin{array}{cc} LC_0 & LC_1 \end{array} \\ \begin{array}{c} k_0 \\ k_1 \\ k_2 \end{array} & \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} \end{array}.$$

After k_0 is recovered, AFIA moves on to determining the other key in LC_0, k_1, (Line 12). We add a *sa1* at k_1 (Line 13), along with constraining on k_0, k_2 to their determined values (Line 14). If the correct logical value for k_0 is *0* (i.e., the stored key), test pattern $P_2 = [x_0 x_1 \dots x_5] = [110X0X]$ can sensitize the *sa1* of k_1 to the output y_0. If the stored secret key bit is $k_0 = 1$, the test pattern P_2 will be different, and its value will be $[0X0X0X]$, which one can verify using the same D-Algorithm. Note that no fault injection is necessary to determine k_1.

Finally, the matrix A will be updated to all zeros and the AFIA recovers the entire key.

4.7 AFIA Complexity Analysis

The average complexity of the AFIA attack is linear with the key size (K). In this section, we show that AFIA is very effective at breaking any logic locking technique. However, the fault injection time may vary depending on the effectiveness of the equipment. It is practically instantaneous to obtain the secret key once the responses are collected from C_O.

Lemma 1 *One input pattern is sufficient to recover one key bit.*

Proof A single test pattern is sufficient to detect a *saf* if such a fault is not redundant [15]. A redundant fault results from a redundant logic that cannot be exercised from the inputs. As the key gates are placed to modify the functionality, it cannot be a redundant logic. As there exists one test pattern to detect a *saf* at the key line, it can be used to recover one key bit.

Theorem 1 *AFIA recovers the entire secret key, K using at most $\mathcal{K}$ number of test patterns, i.e.,*

$$TP_{AFIA}[f_K(C_L) = f(C_O)] \leq \mathcal{K}. \tag{1}$$

where $f_K()$ represents the functionality with K as the key.

Proof A C_L with a $\mathcal{K}$-bit key is injected with a *saf* fault on every key line. As AFIA requires one test pattern to obtain one key bit (see Lemma 1), the upper bound of the number of test patterns is $\mathcal{K}$. However, a single pattern can detect two or more stuck-at faults on the key lines if their effect is visible in different logic cones (e.g., different outputs). As a result, the required number of test patterns to recover the entire key (K) can be less than $\mathcal{K}$.

Theorem 2 *AFIA is applicable to strong logic locking* [55], *where pairwise key gates are inserted to block the propagation of one key by the other.*

Proof In strong logic locking, the propagation of one key is blocked due to the other key. However, $(\mathcal{K} - 1)$ faults are injected at $(\mathcal{K} - 1)$ key lines, worst-case scenario, except for the one whose value needs to be determined. Once an external fault is injected into the functional chip, the key value is fixed and no longer remains unknown. Hence, AFIA is applicable to strong logic locking.

Theorem 3 *The worst-case complexity for the total number of faults injected in AFIA is $\boldsymbol{O}(\mathcal{K}^2)$.*

Proof Let us consider a circuit with a single logic cone locked with a secret key vector $\{k_0, \dots, k_{\mathcal{K}-1}\}$. Suppose all key bits are pairwise non-mutable convergent, i.e., the propagation of one key bit depends on all the other keys. To sensitize the 1^{st} key bit, we need to add $\mathcal{K} - 1$ faults during the fault injection process. The 2^{nd} key bit requires $\mathcal{K} - 2$ faults as the value of the 1^{st} key bit is known. Similarly, the 3^{rd} key bit requires $\mathcal{K} - 3$ faults, and so on. Thus, the total number of faults is:

$\sum_{i=1}^{\mathcal{K}}(\mathcal{K} - i) = \frac{\mathcal{K}\cdot(\mathcal{K}-1)}{2}$.

Thus, the worst-case complexity for the total number of faults injected is $\boldsymbol{O}(\mathcal{K}^2)$.

Theorem 4 *The average-case complexity for the total number of faults injected in AFIA is* $\boldsymbol{O}(\mathcal{K})$.

Proof Consider a circuit with N logic cones, each cone LC_j has negligible or no overlap with its neighboring cones, LC_{j-1} and LC_{j+1}, and $\mathcal{K}$ keys are evenly distributed (amortized) among the N cones. For each cone, it has an average $a = \frac{\mathcal{K}}{N}$ keys). Since negligible overlap between cones, there is no preference between the order of execution on deciphering keys in logic cones, and each cone needs to inject $\frac{\mathcal{K}/N\cdot(\mathcal{K}/N-1)}{2}$ faults. Overall, by summing up all faults for every logic cone, the required number of fault injections is $N \cdot \frac{\mathcal{K}/N\cdot(\mathcal{K}/N-1)}{2}$.

Thus, the average-case complexity is $N \cdot \frac{\mathcal{K}/N\cdot(\mathcal{K}/N-1)}{2} = \frac{a-1}{2} \cdot \mathcal{K} = \boldsymbol{O}(a\mathcal{K}) = \boldsymbol{O}(\mathcal{K})$.

4.8 AFIA on Fault-Tolerant Circuit

Fault-tolerant circuits and circuits with redundancy may prevent the injected faults from being revealed at the output. However, it does not affect our proposed AFIA. As the objective of logic locking is to produce incorrect output for wrong key combinations under certain input patterns, these input patterns ensure the differential output behavior for keys. Thus, the key cannot be inserted inside the region of redundancy, where no input pattern can ever produce differential output. Any key bit placed at these locations cannot corrupt the output so that either logic 0 or logic 1 is its correct value. The SoC designer would not place a key bit in such a way that both logic values gives the correct output since it contradicts the principle of logic locking. In summary, redundancies are not a countermeasure against AFIA attack for a well-designed locked circuit.

4.9 AFIA on Non-Functional-Based Locking Techniques

Our fault injection-based attack can also be extended to non-functional logic locking techniques [38, 51]. The dynamically obfuscated scan-chain (DOSC) technique [51] has three secrets stored in the tamper-proof memory, which are the functional obfuscation key, the LFSR seed, and the control vector. AFIA can break the functional obfuscation key if the obfuscated scan-chain becomes transparent to the attacker. To achieve that, the attacker needs to inject faults at all the *Scan Obfuscation Key* registers directly to get a known shift out state from the functional IP. For the routing-based locking technique [38], our proposed attack is applicable to breaking the key-configurable logarithmic-based network (CLN) as the switch-boxes (SwB) consist of MUX-based key gates. Once a fault is injected into a key register, the selection path for the corresponding MUX is determined. We can target these keys one at a time with test patterns generated from the ATPG tool and inject faults on dependent key registers.

5 Experimental Results

This section provides the feasibility of fault injection to break secure logic locking. Extensive simulations are performed on different benchmarks with different locking techniques to demonstrate the effectiveness of the proposed fault injection attack for breaking a secure locking technique. We have shown a significant reduction of total fault count for AFIA compared to DFA, presented in our conference paper, in breaking the same locked benchmark.

5.1 Laser Fault Injection

To demonstrate the laser fault injection attack, we selected a Kintex-7 FPGA [95], which is used as the device-under-test (DUT). Locked benchmark circuits are implemented in the Kintex-7 FPGA, where faults are injected into key registers. Figure 8 shows the laser fault injection (LFI) setup with a Hamamatsu PHEMOS-1000 FA microscope [46]. The

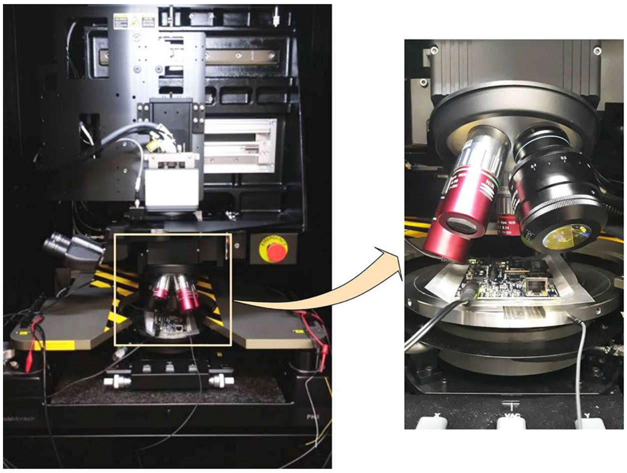

Fig. 8 The FPGA board placed under the lens for laser-fault injection at the target registers

equipment consists of a diode pulse laser source (Hamamatsu C9215-06) with a wavelength of 1064 nm. Three objective lenses were used during this work: 5x/0:14 NA, 20x/0:4 NA, 50x/0:76 NA. The 50x lens is equipped with a correction ring for silicon substrate thickness. The laser diode has two operation modes – a) low power (200 mW) pulse mode, and b) high power (800 mW) impulse mode. The high power impulse mode can be used for laser fault injection. The laser power can be adjusted from 2% to 100% in 0.5% steps.

Photon emission analysis [52] can be used to localize the implemented locked circuitry in the DUT. Thereafter, the DUT is placed under the laser source for LFI. A trigger signal is fed to the PHEMOS-1000 to synchronize the LFI with the DUT operation. Once the device reaches a stable state after power-on, the laser is triggered on the target key registers. After the fault injection, we need to guarantee that the device is still functioning as expected and has not entered into a completely dysfunctional state. The laser triggering timing can be checked by a digital oscilloscope for greater precision.

5.2 Fault Count Comparison

The differential attack methodology (DFA) introduced in Sect. 3 and in [33] requires $\mathcal{K}-1$ number of constraints per test pattern. The total number of faults that need to be injected to determine one key bit is $2\mathcal{K}-1$, as C_A and C_F require $\mathcal{K}-1$ and $\mathcal{K}$ faults, respectively. The total number of faults required to decipher $\mathcal{K}$ key bits is $(2\mathcal{K}-1)\cdot\mathcal{K} = 2\mathcal{K}^2-\mathcal{K}$. Compared to DFA, AFIA only requires injecting faults to key registers if these key bits are interdependent, where the propagation of one key is dependent on others.

Table 1 shows the number of faults to be injected for both the DFA (Algorithm 1) and AFIA (Algorithm 2). To demonstrate the feasibility of the fault injection attack on logic locking, we computed the number of faults after generating test patterns using constrained ATPG using the Synopsys TetraMAX tool [84]. Note that the successful generation of test patterns using constrained ATPG guarantees the successful attack on locking. We choose benchmark circuits with random logic locking (added '-RL' after the benchmark name) and strong logic locking (added '-SL') from TrustHub [61], SFLL-hd (added 'SFLL-hd'), SFLL-flex (added 'SFLL-flex'), and SFLL-rem (added 'SFLL_rem') benchmarks from [98], and GitHub [69]. Column 2 represents the secret key size, whereas Columns 3 and 4 represent the number of faults to determine the entire key for DFA and AFIA, respectively. Data in Column 4 is collected under *sa1* fault in test pattern generation (Algorithm 2). Finally, Column 5 shows the average number of faults to evaluate one key bit under AFIA. For example, with locked benchmark c432-RN320, the number of faults required for DFA is 2016, whereas AFIA requires only 48 faults to extract the 32 key bits, leading to 1.5 faults per key bit. For c1355-SL1280, the number of faults increased significantly to 32,640 for DFA. AFIA only requires 1,419 faults to determine the 128 key bits, or 11.09 faults per key bit.

Table 1 Comparison of Number of Injected Faults

Locked Benchmark	**Key Size**	**DFA**	**AFIA**	
	($\mathcal{K}$)	$\boldsymbol{F_T}$	$\boldsymbol{F_T}$	$F_T/\mathcal{K}$
c432-RN320	32	2016	48	1.5
c432-RN640	64	8128	165	2.58
c432-RN1280	128	32640	1085	8.48
c2670-RN1280	128	32640	520	4.06
c3540-RN1280	128	32640	268	2.09
c5315-RN1280	128	32640	282	2.20
c6288-RN1280	128	32640	268	2.09
c7552-RN1280	128	32640	334	2.61
c1355-SL1280	128	32640	1419	11.09
c1908-SL1280	128	32640	654	5.11
c5315-SL1280	128	32640	3469	27.10
c6288-SL1280	128	32640	368	2.88
c7552-SL1280	128	32640	188	1.47
b14_C_k8_SFLL-hd	8	120	28	3.5
b14_C_k16_SFLL-flex	16	496	120	7.5
b14_C_k32_SFLL-flex	32	2016	496	15.5
b14_C_k64_SFLL-flex	64	8128	2016	31.5
b14_C_k128_SFLL-flex	128	32640	8128	63.5
c432_k8_SFLL-hd	8	120	28	3.5
c432_k16_SFLL-flex	16	496	120	7.5
c432_k32_SFLL-flex	32	2016	496	15.5
c880_k8_SFLL-hd	8	120	28	3.5
c880_k16_SFLL-flex	16	496	120	7.5
c880_k32_SFLL-flex	32	2016	496	15.5
SFLL_rem_k128 [69]	128	32640	8128	63.5

Based on Theorem 4, if keys are uniformly distributed among logic cones, the number of fault injections for AFIA is linear with respect to key size, $\boldsymbol{O}(a\mathcal{K}) = \boldsymbol{O}(\mathcal{K})$, with variable a indicating the average key size per logic cone. If having the same key size, an RLL circuit with more logic cones, or a smaller a, (provided that the size of all logic cones are about the same), should, generally, has fewer fault injections than one with fewer logic cones. This is equivalent to having fewer injected faults in an RLL-based circuit that contains more output than the ones without (see definition of the number of logic cones in Sect. 4.2). Benchmark c432-RN1280 has a larger a than other 128-bit RLL circuits, for c432 has only seven outputs, while c2670 has 140 outputs, c3540 has 22, c5315 has 123, c6288 has 32, c7552 has 108 outputs respectively. (Note, not all logic cones will have keys inside, but the circuit with more output usually has more key-embedded cones than those with fewer outputs.) This is the reason that c432-RN1280 requires considerably more fault injections in total, 1085, than other locked netlist with same key size, where c2670-RN1280 needs 520 faults, c3540-RN1280 has 268, c5315-RN1280 has 282, c6288-RN1280 has 268, c7552-RN1280 has 334, see Table 1.

RLL randomly picks a location in the original unlocked circuit for key gate insertion, while SLL produces more blocking keys. In terms of theoretical complexity analysis, as long as the key gates in RLL locked circuit are distributed uniformly, the number of fault injections for SLL should be larger than RLL, under the same original unlocked benchmark and the same key size, e.g., c5315-RN1280 and c5315-SL1280, c6288-RN1280 and c6288-SL1280. For SFLL-hd and SFLL-flex, each locked circuit has a perturbation unit and a restoration unit. All keys reside in the functionality restoration unit, where every key passes through the output of the restoration subcircuit to reach the primary output [2, 76]. Because of this restoration unit, all key bits are interdependent. Hence, all SFLL-flex and SFLL-hd circuits belong to the worst-case scenario as in Theorem 3, in which the number of injected faults is $\frac{\mathcal{K}\cdot(\mathcal{K}-1)}{2}$. We also evaluated our proposed attack on the latest SFLL variant, SFLL-rem [67, 68]. Although SFLL-rem does not have the added perturb unit, the keys are present in the restoration unit only, and our attack can still break it.

6 Future Work

Although AFIA targets combinational logic circuits or sequential ones with scan-chain access, it can be extended to other clock-based and timing-based locking techniques that target output corruptibility in a different clock cycle [7, 50, 82]. These techniques require multiple clock cycles (typically two) to capture the key to a storage element and thus observe its effect on the circuit behavior (i.e., output corruptibility). Fortunately, the same fault injection-based attack proposed in this paper can be applied to these locking techniques as well. We, however, need to consider transition delay faults (*TDFs*) or path delay faults (*PDFs*) instead of stuck faults to propagate the effect of the targeted key on the output. The same Algorithm 2 can be applied to generate patterns to launch the attack. Note that the *TDFs* and *PDFs* require multiple captures (typically 2). By controlling the fault injection in a precise timing range, it is possible to observe the key through launch on shift (*LOS*) and launch on capture (*LOC*) schemes [15, 62].

7 Conclusion

This paper presents AFIA, a novel stuck-at fault-based fault injection attack that undermines the security of any logic locking technique. AFIA utilizes cone analysis to analyze the dependency of keys. Faults are injected only at the interdependent key bits, which is a significant improvement from the previously published attack DFA [33], dropping the total number of faults to the linear multiple of key size. With the automatic test pattern generation (ATPG) tool, we constructed a pattern set, which is used to apply to an unlocked chip. Each pattern is sufficient to determine a one-bit key. All key bits are derived by comparing collected responses from fault injections and the predicted response from test pattern generation. We performed laser fault injections on Kintex-7 FPGA with various locked benchmark circuits and state-of-the-art locking techniques, and our results have demonstrated the effectiveness of the proposed AFIA scheme. Our future work will focus on developing a locking technique to prevent AFIA.

Funding This work was supported by the National Science Foundation under Grant Number CNS-1755733.

Data Availability The authors declare that the data supporting the findings of this study are available within the article.

Declarations

Conflict of Interest The authors have no conflicts of interest to declare that are relevant to the content of this article.

References

1. Alkabani Y, Koushanfar F (2007) Active hardware metering for intellectual property protection and security. In: USENIX Security Symposium, pp 291–306
2. Alrahis L, Patnaik S, Khalid F, Hanif MA, Saleh H, Shafique M, Sinanoglu O (2021a) GNNUnlock: Graph Neural Networks-based Oracle-less Unlocking Scheme for provably secure logic locking. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, pp 780–785
3. Alrahis L, Patnaik S, Knechtel J, Saleh H, Mohammad B, Al-Qutayri M, Sinanoglu O (2021b) UNSAIL: Thwarting oracle-less machine learning attacks on logic locking. IEEE Trans Inf Forensics and Secur 16:2508–2523
4. Alrahis L, Yasin M, Limaye N, Saleh H, Mohammad B, Alqutayri M, Sinanoglu O (2019) ScanSAT: Unlocking static and dynamic scan obfuscation. Trans Emerg Topics Computing 9(4):1867-1882
5. Azar KZ, Kamali HM, Homayoun H, Sasan A (2019) SMT attack: Next generation attack on obfuscated circuits with capabilities and performance beyond the SAT attacks. In: IACR Transactions on Cryptographic Hardware and Embedded Systems, pp 97–122
6. Azar KZ, Kamali HM, Homayoun H, Sasan A (2021a) From cryptography to logic locking: a survey on the architecture evolution of secure scan chains. IEEE Access 9:73133–73151

7. Azar KZ, Kamali HM, Roshanisefat S, Homayoun H, Sotiriou CP, Sasan A (2021b) Data flow obfuscation: a new paradigm for obfuscating circuits. IEEE Trans Very Large Scale Integr Syst 29(4):643–656
8. Barenghi A, Bertoni GM, Breveglieri L, Pellicioli M, Pelosi G (2010) Low voltage fault attacks to AES. In: International Symposium on Hardware-Oriented Security and Trust (HOST), pp 7–12
9. Barenghi A, Bertoni GM, Breveglieri L, Pelosi G (2013) A fault induction technique based on voltage underfeeding with application to attacks against AES and RSA. J Syst Softw 1864–1878
10. Barenghi A, Breveglieri L, Koren I, Naccache D (2012) Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. Proceedings of the IEEE, pp 3056–3076
11. Baumgarten A, Tyagi A, Zambreno J (2010) Preventing IC piracy using reconfigurable logic barriers. IEEE Des Test Comput 27(1):66–75
12. Beerel P, Georgiou M, Hamlin B, Malozemoff AJ, Nuzzo P (2022) Towards a formal treatment of logic locking. Cryptology ePrint Archive
13. Blömer J, Seifert JP (2003) Fault based cryptanalysis of the advanced encryption standard (AES). In: International Conference on Financial Cryptography, Springer, pp 162–181
14. Boneh D, DeMillo RA, Lipton RJ (1997) On the importance of checking cryptographic protocols for faults. In: International Conference on the Theory and Applications of Cryptographic Techniques, pp 37–51
15. Bushnell ML, Agrawal VD (2004) Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits, Frontiers in Electronic Testing Series Volume 17. Springer Science & Business Media
16. Canivet G, Maistri P, Leveugle R, Clédière J, Valette F, Renaudin M (2011) Glitch and laser fault attacks onto a secure AES implementation on a SRAM-based FPGA. J Cryptol 247–268
17. Castillo E, Meyer-Baese U, García A, Parrilla L, Lloris A (2007) IPP@HDL: Efficient intellectual property protection scheme for IP cores. IEEE Trans VLSI (TVLSI) 578–591
18. Chakraborty RS, Bhunia S (2008) Hardware protection and authentication through netlist level obfuscation. In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design, pp 674–677
19. Charbon E (1998) Hierarchical watermarking in IC design. In: Proceedings of the IEEE Custom Integrated Circuits Conference, pp 295–298
20. Cormen T, Leiserson C, Rivest R, Stein C (2009) Introduction to algorithms. MIT Press, Computer Science
21. Dehbaoui A, Dutertre JM, Robisson B, Tria A (2012) Electromagnetic transient faults injection on a hardware and a software implementations of AES. In: Workshop on Fault Diagnosis and Tolerance in Cryptography, pp 7–15
22. Dusart P, Letourneux G, Vivolo O (2003) Differential fault analysis on AES. In: International Conference on Applied Cryptography and Network Security, pp 293–306
23. Duvalsaint D, Jin X, Niewenhuis B, Blanton R (2019a) Characterization of locked combinational circuits via ATPG. In: IEEE International Test Conference (ITC), pp 1–10
24. Duvalsaint D, Liu Z, Ravikumar A, Blanton RD (2019b) Characterization of locked sequential circuits via ATPG. In: 2019 IEEE International Test Conference in Asia (ITC-Asia), IEEE, pp 97–102
25. Dworkin MJ, Barker EB, Nechvatal JR, Foti J, Bassham LE, Roback E, Dray Jr JF (2001) Advanced Encryption Standard (AES). NIST Publication Series: Federal Information Processing Standards (NIST FIPS)-197, pp 1–51
26. El Massad M, Garg S, Tripunitara M (2017) Reverse engineering camouflaged sequential circuits without scan access. In: 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), IEEE, pp 33–40
27. Fukunaga T, Takahashi J (2009) Practical fault attack on a cryptographic LSI with ISO/IEC 18033-3 block ciphers. In: Workshop on Fault Diagnosis and Tolerance in Cryptography, pp 84–92
28. Guilley S, Sauvage L, Danger JL, Selmane N, Pacalet R (2008) Silicon-level solutions to counteract passive and active attacks. In: Workshop on Fault Diagnosis and Tolerance in Cryptography, pp 3–17
29. Guin U, Shi Q, Forte D, Tehranipoor MM (2016) FORTIS: a comprehensive solution for establishing forward trust for protecting IPs and ICs. ACM Transactions on Design Automation of Electronic Systems (TODAES), p 63
30. Guin U, Zhou Z, Singh A (2017) A novel Design-for-Security (DFS) architecture to prevent unauthorized IC overproduction. In: VLSI Test Symposium (VTS), pp 1–6
31. Guin U, Zhou Z, Singh A (2018) Robust Design-for-Security architecture for enabling trust in IC manufacturing and test. Transactions on Very Large Scale Integration (VLSI) Systems, pp 818–830
32. Hsueh MC, Tsai TK, Iyer RK (1997) Fault injection techniques and tools. Computer 30(4):75–82
33. Jain A, Rahman T, Guin U (2020) ATPG-guided fault injection attacks on logic locking. In: IEEE Physical Assurance and Inspection of Electronics (PAINE), pp 1–6
34. Jain A, Zhou Z, Guin U (2021) TAAL: tampering attack on any key-based logic locked circuits. ACM Trans Des Automation Electronic Syst 26(4):1–22
35. Jarvis RW, McIntyre MG (2007) Split manufacturing method for advanced semiconductor circuits. US Patent 7,195,931
36. Kahng AB, Lach J, Mangione-Smith WH, Mantik S, Markov IL, Potkonjak M, Tucker P, Wang H, Wolfe G (2001) Constraint-based watermarking techniques for design IP protection. IEEE Transactions on CAD of Integrated Circuits and Systems, pp 1236–1252
37. Kamali HM, Azar KZ, Gaj K, Homayoun H, Sasan A (2018) LUT-Lock: a novel LUT-based logic obfuscation for FPGA-Bitstream and ASIC-Hardware protection. In: 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), IEEE, pp 405–410
38. Kamali HM, Azar KZ, Homayoun H, Sasan A (2019) Full-lock: Hard distributions of sat instances for obfuscating circuits using fully configurable logic and routing blocks. In: Proceedings of the 56th Annual Design Automation Conference 2019, pp 1–6
39. Kamali HM, Azar KZ, Homayoun H, Sasan A (2020) Interlock: an intercorrelated logic and routing locking. In: 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD), IEEE, pp 1–9
40. Kamali HM, Azar KZ, Farahmandi F, Tehranipoor M (2022) Advances in logic locking: Past, present, and prospects. Cryptology ePrint Archive
41. Karmakar R, Chatopadhyay S, Kapur R (2018) Encrypt flip-flop: a novel logic encryption technique for sequential circuits. arXiv preprint: arXiv:1801.04961
42. Lee CY, Xie J (2019) High capability and low-complexity: Novel fault detection scheme for finite field multipliers over gf (2 m) based on mspb. In: 2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), IEEE, pp 21–30
43. Lee J, Tebranipoor M, Plusquellic J (2006) A low-cost solution for protecting IPs against scan-based side-channel attacks. In: 24th IEEE VLSI Test Symposium, IEEE, p 6
44. Limaye N, Sengupta A, Nabeel M, Sinanoglu O (2019) Is robust Design-for-Security robust enough? Attack on locked circuits with restricted scan chain access. arXiv preprint: arXiv:1906.07806
45. Paar C, Pelzl J (2009) Understanding cryptography: a textbook for students and practitioners. Springer Science & Business Media

46. PHEMOS-1000 Emission microscope, HAMAMATSU. https://www.hamamatsu.com/eu/en/product/semiconductor-manufacturing-support-systems/failure-analysis-system/C11222-16.html. Accessed 25 Sep 2022
47. Piret G, Quisquater JJ (2003) A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In: International Workshop on Cryptographic Hardware and Embedded Systems, pp 77–88
48. Pouget V, Douin A, Lewis D, Fouillat P, Foucard G, Peronnard P, Maingot V, Ferron J, Anghel L, Leveugle R et al (2007) Tools and methodology development for pulsed laser fault injection in SRAM-based FPGAs. In: Latin-American Test Workshop (LATW)
49. Qu G, Potkonjak M (2007) Intellectual property protection in VLSI designs: Theory and practice. Springer Sc. & Business Media
50. Rahman MS, Guo R, Kamali HM, Rahman F, Farahmandi F, Abdel-Moneum M (2022) O'Clock: Lock the clock via clock-gating for SoC IP protection. In: Design Automation Conf. (DAC), pp 1–6
51. Rahman MS, Nahiyan A, Rahman F, Fazzari S, Plaks K, Farahmandi F, Forte D, Tehranipoor M (2021) Security assessment of dynamically obfuscated scan chain against oracle-guided attacks. ACM Trans Des Automation Electronic Syst 26(4):1–27
52. Rahman MT, Asadizanjani N (2019) Backside security assessment of modern SoCs. In: International Workshop on Microprocessor/SoC Test, Security and Verification (MTV), pp 18–24
53. Rahman MT, Rahman MS, Wang H, Tajik S, Khalil W, Farahmandi F, Forte D, Asadizanjani N, Tehranipoor M (2020) Defense-in-depth: a recipe for logic locking to prevail. Integration 72:39–57
54. Rahman MT, Tajik S, Rahman MS, Tehranipoor M, Asadizanjani N (2020) The key is left under the mat: on the inappropriate security assumption of logic locking schemes. In: 2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), IEEE, pp 262–272
55. Rajendran J, Pino Y, Sinanoglu O, Karri R (2012) Security analysis of logic obfuscation. In: Proceedings of Annual Design Automation Conference, pp 83–89
56. Rajendran J, Zhang H, Zhang C, Rose GS, Pino Y, Sinanoglu O, Karri R (2015) Fault analysis-based logic encryption. IEEE Transactions on Computers, pp 410–424
57. Rajski J, Tyszer J, Kassab M, Mukherjee N (2004) Embedded deterministic test. IEEE Trans Comput Aided Des Integr Circ Syst 23(5):776–792
58. Roshanisefat S, Mardani Kamali H, Sasan A (2018) SRCLock: SAT-Resistant cyclic logic locking for protecting the hardware. In: Proceedings of 2018 Great Lakes Symposium on VLSI, pp 153–158
59. Roy JA, Koushanfar F, Markov IL (2008) EPIC: Ending Piracy of Integrated Circuits. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp 1069–1074
60. Roy JA, Koushanfar F, Markov IL (2010) Ending piracy of integrated circuits. Computer 30–38
61. Salmani H, Tehranipoor M, Trust-Hub [Online]. Available: https://trust-hub.org/home. Accessed 25 Sep 2022
62. Savir J, Patil S (1994) Broad-side delay test. IEEE Trans Comput Aided Des Integr Circ Syst 13(8):1057–1064
63. Schmidt JM, Hutter M (2007) Optical and EM fault-attacks on CRT-based RSA: Concrete results
64. Selmane N, Guilley S, Danger JL (2008) Practical setup time violation attacks on AES. In: Seventh European Dependable Computing Conference, pp 91–96
65. Selmke B, Heyszl J, Sigl G (2016) Attack on a DFA protected AES by simultaneous laser fault injections. In: Workshop on Fault Diagnosis and Tolerance in Cryptography, pp 36–46
66. Sengupta A, Ashraf M, Nabeel M, Sinanoglu O (2018a) Customized locking of IP blocks on a Multi-Million-Gate SoC. In: International Conference on Computer-Aided Design (ICCAD), pp 1–7
67. Sengupta A, Nabeel M, Yasin M, Sinanoglu O (2018b) ATPG-based cost-effective, secure logic locking. In: VLSI Test Symposium (VTS), pp 1–6
68. Sengupta A, Nabeel M, Limaye N, Ashraf M, Sinanoglu O (2020) Truly stripping functionality for logic locking: a fault-based perspective. IEEE Trans Comput Aided Des Integr Circ Syst 39(12):4439–4452
69. SFLL_rem. https://github.com/micky960/SFLL_fault. Accessed 25 Sep 2022
70. Shakya B, Xu X, Tehranipoor M, Forte D (2020) Cas-lock: a security-corruptibility trade-off resilient logic locking scheme. IACR Transactions on Cryptographic Hardware and Embedded Systems, pp 175–202
71. Shamsi K, Li M, Plaks K, Fazzari S, Pan DZ, Jin Y (2019a) IP protection and supply chain security through logic obfuscation: a systematic overview. ACM Transactions on Design Automation of Electronic Systems (TODAES) 24(6):1–36
72. Shamsi K, Pan DZ, Jin Y (2019b) IcySAT: Improved SAT-based attacks on cyclic locked circuits. In: 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), IEEE, pp 1–7
73. Shamsi K, Pan DZ, Jin Y (2019c) On the impossibility of approximation-resilient circuit locking. In: 2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), IEEE, pp 161–170
74. Shannon CE (1949) Communication theory of secrecy systems. Bell Syst Tech J 28(4):656–715
75. Shen H, Asadizanjani N, Tehranipoor M, Forte D (2018) Nanopyramid: an optical scrambler against backside probing attacks. In: Proceedings on International Symposium for Testing and Failure Analysis (ISTFA), p 280
76. Sirone D, Subramanyan P (2020) Functional analysis attacks on logic locking. IEEE Trans Inf Forensics Secur 15:2514–2527
77. Sisejkovic D, Merchant F, Reimann LM, Leupers R (2021) Deceptive logic locking for hardware integrity protection against machine learning attacks. IEEE Trans Comput Aided Des Integr Circ Syst pp 1-14
78. Skarin D, Barbosa R, Karlsson J (2010) GOOFI-2: a tool for experimental dependability assessment. In: IEEE/IFIP International Conference on Dependable Systems & Networks (DSN), pp 557–562
79. Skorobogatov S (2010) Optical fault masking attacks. In: Workshop on Fault Diagnosis and Tolerance in Cryptography, pp 23–29
80. Skorobogatov SP, Anderson RJ (2002) Optical fault induction attacks. In: International Workshop on Cryptographic Hardware and Embedded Systems, pp 2–12
81. Subramanyan P, Ray S, Malik S (2015) Evaluating the security of logic encryption algorithms. In: IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp 137–143
82. Sweeney J, Zackriya VM, Pagliarini S, Pileggi L (2020) Latch-based logic locking. In: 2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), IEEE, pp 132–141
83. Synopsys (2021a) Design compiler graphical: Create a better starting point for faster physical implementation. https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-graphical.html

84. Synopsys (2021b) TetraMAX ATPG: Automatic test pattern generation. https://www.synopsys.com/implementation-and-signoff/test-automation/testmax-atpg.html
85. Tehranipoor M, Wang C (2011) Introduction to hardware security and trust. Springer Science & Business Media
86. TestMAX DFT, Design-for-Test Implementation, Synopsys. https://www.synopsys.com/content/dam/synopsys/implementation &signoff/datasheets/testmax-dft-ds.pdf. Accessed 25 Sep 2022
87. Torrance R, James D (2009) The state-of-the-art in IC reverse engineering. In: International Workshop on Cryptographic Hardware and Embedded Systems, pp 363–381
88. Tsai T, Iyer R (1995) FTAPE - a fault injection tool to measure fault tolerance. In: Computing in Aerospace Conference, p 1041
89. Vashistha N, Lu H, Shi Q, Rahman MT, Shen H, Woodard DL, Asadizanjani N, Tehranipoor M (2018) Trojan scanner: Detecting hardware trojans with rapid SEM imaging combined with image processing and machine learning. In: Proceedings on International Symposium for Testing and Failure Analysis, p 256
90. Wang X, Zhang D, He M, Su D, Tehranipoor M (2018) Secure scan and test using obfuscation throughout supply chain. Transactions on Computer-Aided Design of Integrated Circuits and Systems 37(9):1867–1880
91. Wu H, Ferranti D, Stern L (2014) Precise nanofabrication with multiple ion beams for advanced circuit edit. Microelectron Reliab 1779–1784
92. Xie Y, Srivastava A (2016) Anti-SAT: Mitigating SAT attack on logic locking. In: International Conference on Cryptographic Hardware and Embedded Systems, pp 127–146
93. Xie Y, Srivastava A (2017) Delay locking: Security enhancement of logic locking against IC counterfeiting and overproduction. In: Proceedings of the 54th Annual Design Automation Conference, pp 1–6
94. Xie Y, Srivastava A (2019) Anti-SAT: Mitigating SAT attack on logic locking. IEEE Trans Comput Aided Des Integr Circ Syst 38(2):199–207
95. Xilinx (2021) Xilinx Kintex-7 FPGA KC705 evaluation kit. https://www.xilinx.com/products/boards-and-kits/ek-k7-kc705-g.html
96. Yasin M, Mazumdar B, Rajendran JJ, Sinanoglu O (2016a) SARLock: SAT attack resistant logic locking. In: IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp 236–241
97. Yasin M, Rajendran JJ, Sinanoglu O, Karri R (2016b) On improving the security of logic locking. Transactions on Computer-Aided Design of Integrated Circuits and Systems 35(9):1411–1424
98. Yasin M, Sengupta A, Nabeel MT, Ashraf M, Rajendran JJ, Sinanoglu O (2017) Provably-secure logic locking: from theory to practice. In: Proceedings of ACM SIGSAC Conference on Computer and Communications Security, pp 1601–1618
99. Yeh A (2012) Trends in the global IC design service market. DIGITIMES Research
100. Zhang J, Yuan F, Wei L, Liu Y, Xu Q (2015) VeriTrust: Verification for Hardware Trust. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 34(7):1148–1161
101. Zhang Y, Cui P, Zhou Z, Guin U (2019) TGA: an oracle-less and topology-guided attack on logic locking. In: Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop, pp 75–83
102. Zhang Y, Jain A, Cui P, Zhou Z, Guin U (2020) A novel topology-guided attack and its countermeasure towards secure logic locking. J Cryptogr Eng 1–14
103. Zhong Y, Guin U (2022) Complexity analysis of the SAT attack on logic locking. arXiv preprint: arXiv:2207.01808

Yadi Zhong is currently pursuing her Ph.D. in Computer Engineering from the Department of Electrical and Computer Engineering, Auburn University, AL, USA. She received her B.E. degree from the same university in 2020. Her research interest are logic locking, fault injection and hardware security, and post-quantum cryptography. She received Auburn University Presidential Graduate Research Fellowships in 2020.

Ayush Jain received his M.S. Degree from the Department of Electrical and Computer Engineering, Auburn University, AL, USA in 2020. He is currently working as SoC Design Engineer at Intel Corporation. He received his B.Tech. degree from the Electrical Engineering Department, Pandit Deendayal Petroleum University, Gujarat, India, in 2018. His current research interests include hardware security, VLSI design, and testing.

M. Tanjidur Rahman received his Ph.D. degree in electrical and computer engineering in 2021 from University of Florida. He obtained his BS (with honors) and MS in electrical and electronic engineering from Bangladesh University of Engineering and Technology (BUET) in 2012, and 2014, respectively. His research interests include hardware security and trust, physical assurance, configurable security architecture, and reliable VLSI design. Dr. Rahman has authored 7 technical journal and 8 conference papers. He has also published one book and one book chapter on physical assurance and chip backside security assessment. He has two patent applications under review.

Navid Asadizanjani received the Ph.D. degree in Mechanical Engineering from University of Connecticut, Storrs, CT, USA, in 2014. He is currently an Assistant Professor with the Electrical and Computer Engineering Department at University of Florida, Gainesville, FL, USA. His current research interest is primary on "Physical Attacks and Inspection of Electronics". This includes wide range of products from electronic systems to devices. He is involved with counterfeit detection and prevention, system and chip level reverse engineering, Anti reverse engineering, etc. Dr. Asadizanjani has received and was nominated for several best paper awards from International Symposium on Hardware Oriented Security and Trust (HOST) and International Symposium on Flexible Automation (ISFA). He was also winner of D.E. Crow Innovation award from University of Connecticut. He is currently the program chair of the PAINE conference and is serving on the technical program committees of several top conferences including International Symposium of Testing and Failure Analysis (ISTFA) and IEEE Computing and Communication Workshop and Conference (CCWC).

Jiafeng Xie received the M.E. and Ph.D. degrees from Central South University and University of Pittsburgh, in 2010 and 2014, respectively. He is currently an Assistant Professor in the Department of Electrical & Computer Engineering, Villanova University, Villanova, PA. His research interests include cryptographic engineering, hardware security, post-quantum cryptography, and VLSI implementation of neural network systems. Dr. Xie has served as technical committee member for reputed conferences such as HOST, ICCD, and ISVLSI. He is also currently serving as Associate Editor for Microelectronics Journal and IEEE Access. He was Associate Editor for IEEE Transactions on Circuits and Systems-II: Express Briefs. He received the IEEE Access Outstanding Associate Editor recognition for the year of 2019. He also received the Best Paper Award from HOST'19.

Ujjwal Guin received his PhD degree from the Electrical and Computer Engineering Department, University of Connecticut, in 2016. He is currently an Assistant Professor in the Electrical and Computer Engineering Dept. of Auburn University, Auburn, AL, USA. He received his B.E. degree from the Dept. of Electronics and Telecommunication Engineering, Bengal Engineering and Science University, Howrah, India, in 2004 and his M.S. degree from the Dept. of Electrical and Computer Engineering, Temple University, Philadelphia, PA, USA, in 2010. He has developed several on-chip structures and techniques to improve the security, trustworthiness, and reliability of integrated circuits. His current research interests include Hardware Security & Trust. He has authored several journal articles and refereed conference papers. He serves on the organizing committees of HOST, VTS, and PAINE, and technical program committees of DAC, HOST, VTS, PAINE, VLSID, GLSVLSI, ISVLSI, and Blockchain. He is an active participant in the SAE International G-19A Test Laboratory Standards Development Committee and G-32 Cyber-Physical Systems Security Committee. He is a member of ACM and senior member of IEEE.