Valkyrie: Vulnerability Assessment Tool and Attack for Provably-Secure Logic Locking Techniques

Nimisha Limaye, *Graduate Student Member, IEEE*, Satwik Patnaik, *Member, IEEE,* and Ozgur Sinanoglu, *Senior Member, IEEE*

***Abstract*—Protection of the design intellectual property (IP) has become a pertinent need owing to the globalized integrated circuit (IC) supply chain. Logic locking has been perceived as a holistic solution ensuring protection against multiple supply chain entities. The research community has proposed many logic locking techniques, out of which provably-secure logic locking (PSLL) techniques have gathered traction due to their algorithmic and mathematical security guarantees. However, there has been a perpetual cat-and-mouse game between the attackers and the defenders. Although these logic locking techniques are provably secure, they are typically short-lived due to the weaknesses in their hardware/structural implementation that attacks exploit. We attribute this cat-and-mouse game to the lack of a diagnostic tool for PSLL techniques for security-enforcing designers and raise the question, "*Can a designer proactively diagnose the hardware implementation of a PSLL technique for structural vulnerabilities before taking the design to silicon?*"**

In this work, we first review the recent PSLL techniques to extract generic properties, based on which we develop a first-of-its-kind security diagnostic tool (*Valkyrie*) that a security-enforcing designer can use to assess the structural vulnerabilities before taking the design to silicon. We also propose a generic circuit-recovery attack, validating the tool results to assure the community that if the tool identifies a vulnerability, it can always be exploited. Thus, our attack acts as a cautionary tale to the designer. We make these claims after verifying the efficacy of our tool and attack on 15 (seven broken and eight unbroken) PSLL techniques for different synthesis tools, technology libraries, and abstraction levels across a dataset of more than $20,000$ locked designs. *We observe* 100*% success in all these cases.* Our diagnostic tool (which we open-source) can thus serve as a vehicle to test the structural resilience of the hardware implementation of any newly developed PSLL technique. We envision *Valkyrie* bringing a much-needed control over the cat-and-mouse game that the PSLL research has been trapped in.

***Index Terms*—Hardware security, IP protection, logic locking, security diagnostic tool, circuit recovery attack**

I. Introduction

Manuscript received August 2, 2021; revised November 29, 2021; accepted January 13, 2022. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Ulrich Rührmair. (*Nimisha Limaye and Satwik Patnaik contributed equally to this work.*) (*Corresponding authors: Nimisha Limaye and Satwik Patnaik.*)

Nimisha Limaye is with the Department of Electrical and Computer Engineering, Tandon School of Engineering, New York University, Brooklyn, NY 11201, USA (email: nimisha.limaye@nyu.edu).

Satwik Patnaik was with the Department of Electrical and Computer Engineering, Tandon School of Engineering, New York University, Brooklyn, NY 11201, USA. He is currently with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: satwik.patnaik@tamu.edu).

Ozgur Sinanoglu is with the Division of Engineering, New York University Abu Dhabi, Abu Dhabi 129188, UAE (email: ozgursin@nyu.edu).

Digital Object Identifier 10.1109/TIFS.2022.3149147

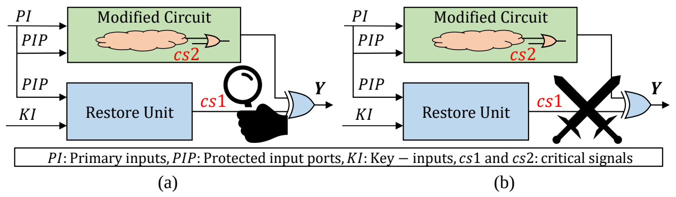

PI
PIP
Modified Circuit
cs2
PIP
KI
Restore Unit
cs1
Y
PI: Primary inputs, PIP: Protected input ports, KI: Key − inputs, cs1 and cs2: critical signals
(a)
(b)

Fig. 1. A high-level overview of our work. (a) *Valkyrie*[1] security diagnostic tool for a security-enforcing designer to inspect the hardware implementation (magnifying glass) for structural vulnerabilities (critical signals $cs1$ and $cs2$) in a provably-secure logic locking (PSLL) technique. (b) Attacker exploits the vulnerabilities (swords) diagnosed by the security diagnostic tool.

THE astronomical cost of setting up a foundry for advanced semiconductor technology nodes has led design companies such as Apple®, NVIDIA®, and Qualcomm® to adopt a fabless business model [1]. Reports indicate that TSMC® is pursuing an investment of $23 billion for the 3nm technology node [2]. Notwithstanding the cost involved in setting up such foundries, lithography concerns have also exacerbated the process of manufacturing integrated circuits (ICs). Recently, Intel® identified a defect mode in their manufacturing process, which delayed the manufacturing of their 7 nm chips [3]. Consequently, Intel® chalked out contingency options which include outsourcing manufacturing to "third-party foundries [4]." Outsourcing manufacturing to third-party, potentially *untrustworthy* foundries gives rise to the threat of intellectual property (IP) piracy, overproduction of ICs, and insertion of Trojans [5]. In this work, we focus on the threat of design IP piracy and the illegal overproduction of ICs.

A. Logic Locking

Over the past decade, researchers proposed several countermeasures for IP protection, out of which logic locking has gained significant traction due to its ease of implementation and straightforward integration within the supply chain.[2] Logic locking provides IP protection against several untrusted entities in the supply chain, such as foundry, test facilities, and end users [8]. It creates additional key-inputs in a design such that only the secret key loaded post-fabrication on a

[1]In Norse mythology, *Valkyrie* means the chooser of the slain. Analogous to that, the *Valkyrie* security diagnostic tool can establish which PSLL technique is resilient (will live) and which is not (will die) against *structural attacks*.

[2]Industrial and government agencies have shown a keen interest in adopting logic locking. On the industrial front, Mentor Graphics® has included logic locking in its *Trust Chain* framework [6], while government agencies such as Defense Advanced Research Projects Agency (DARPA) are investing in logic locking through programs such as Obfuscated Manufacturing of GPS (OMG) and Automatic Implementation of Secure Silicon (AISS) [7].

TABLE I
EFFICACY OF THE *Valkyrie* SECURITY DIAGNOSTIC TOOL IN IDENTIFYING STRUCTURAL VULNERABILITIES ACROSS 15 PSLL TECHNIQUES; THE TECHNIQUES HIGHLIGHTED IN ITALICS HAVE NOT BEEN BROKEN YET. ✓ DENOTES SUCCESSFUL IDENTIFICATION OF STRUCTURAL VULNERABILITIES

Defense →	**SARLock**	**Anti-SAT**	**TTLock**	**SFLL-HD**	***SFLL-flex***	**SFLL-rem**	**CASLock**	**MCAS**	***SAS***	***Gen-Anti-SAT***		***CAC***	***DTL***		
										Comp.	***Non-Comp.***		***SARLock***	***Anti-SAT***	***CAC***
Valkyrie	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

secure memory (a.k.a tamper-proof memory) that drives these key-inputs renders the fabricated chips functional [9]. In the process, it inserts additional gates (key-gates) in the design.

Pre-SAT Locking: Initial research focused on determining suitable locations for key-gate insertion to achieve high output corruption.[3] These locations were either chosen randomly (random logic locking [9]) or following some heuristics.

SAT-based attack [10]: The initial locking techniques were circumvented by an input-output (I/O) query-based attack, which leveraged Boolean satisfiability solvers (commonly known as the SAT-based attack). The attack leverages SAT solvers to compute *distinguishing input patterns (DIPs)*[4] and queries a working chip (a.k.a. *oracle*) with the key loaded in its memory with these DIPs. The attack iteratively generates DIPs and gradually prunes the key search space based on the responses of the oracle to the DIPs.[5] The SAT-attack resilience of any locking technique can be measured by the total time T=$\sum_{i=0}^{\lambda} t_i$ [13], where λ is the number of SAT iterations and t_i is the time taken per SAT iteration.

Post-SAT Locking: Researchers identified three directions to strengthen logic locking techniques which include (i) increasing the number of iterations, (ii) increasing the time taken per iteration, and (iii) inserting configurable and cyclic interconnection networks. Considering (i), researchers inserted *point-functions*[6] to prune out *exactly* one incorrect key per SAT iteration, thereby forcing the attack to make exponential calls (in the size of the key) to decipher the secret key [13, 14]. Considering (ii), researchers inserted SAT-hard structures (such as multipliers, look-up tables, routing structures) to increase the SAT complexity of the locked design, thereby increasing the time taken per iteration to recover the secret key [15, 16]. Finally, in (iii), researchers augmented the design with cyclic interconnection networks that create a logical combinational loop in the design [17]. The presence of loops hinders attackers from launching the SAT-based attack and other I/O query-based attacks as the circuit can no longer be represented as a directed acyclic graph, which is a fundamental requirement for most I/O query-based attacks.

In this work, we consider the techniques in (i) because of their algorithmic security guarantees against SAT-based (and other I/O query-based) attacks. Next, we discuss the cat-and-mouse game[7] that has ensued in the PSLL domain.

B. The (Never-Ending) Cat-and-Mouse Game in PSLL

In 2016, researchers proposed SARLock [14] and Anti-SAT [13], which add point-functions to thwart the SAT-based attack. Such a construction forces the SAT-based attack to make exponential calls to an oracle for deciphering the secret key. However, both techniques were circumvented by a *signal probability skew (SPS)* attack [23] that utilizes signal probabilities to identify and then remove the added point-function circuitry from the locked design. These techniques were also thwarted by the *bypass* attack [24]. The bypass attack first obtains DIPs for which the corrupted output port returns an incorrect value on the application of a wrong key. Then a “bypass circuitry” is constructed using these DIPs and appended to the locked design to invert the corrupted output port. This construction facilitates an attacker to successfully recover the original design (bypassing the locked design), even under the application of an incorrect key [24].

In 2017, researchers modified the original design for one or a few input patterns by hard-coding them (using an AND-tree) and then corrected this modified functionality using a key-controlled restore unit [25]. This construction thwarted SPS and bypass attacks. An attacker removing the restore unit obtains a modified design instead of the original design. However, in 2019, attackers identified traces of these hard-coded input patterns through graph connectivity and structural analysis, leading to key recovery attacks [26, 27].

In 2018, researchers utilized a fault-injection-based approach, where instead of inserting a hard-coded point function, they removed logic from the original design [28]. This technique (*SFLL-rem*) successfully thwarted all the then existing attacks and stood unbroken in a global community-led challenge [29]. However, in 2021, this technique was broken by an attack leveraging logic synthesis principles [30]. The attack analyzes the prime implicant table (PIT) to recover secrets based on a finding that a removed protected input pattern may not merge with prime implicants in the original PIT.

In 2020, researchers proposed CASLock [31] to thwart the SAT-based attack and the bypass attack. Mirrored-CAS (MCAS) was proposed to thwart SPS and removal attacks [31]. However, these techniques were broken using attacks that identified the locking units through structural analysis [32].

[3]Output corruption is achieved when an incorrect key produces an erroneous output response—when 50% of the output ports in a locked design are corrupted, we consider it as high output corruption.

[4]Inputs that help in eliminating incorrect keys from the key search space.

[5]The SAT-based attack was followed by *AppSAT* [11] and *Double-DIP* [12]. While *AppSAT* relaxed the exactness constraint and illustrated the trade-off between exact-attack resiliency and approximation resiliency, *Double-DIP* finds 2-DIP (doubly differentiating input pattern) in each attack iteration.

[6]Boolean function that evaluates to 1 at *exactly* one input pattern.

[7]A similar cat-and-mouse game exists in domains (ii) and (iii). For example, cyclic locking techniques have been challenged by *CycSAT* [18] and other variants [19, 20]. SAT-hard locking techniques [15, 16] have been successfully attacked by modeling-based attack [21] and neural network-guided attack [22].

TABLE II
EFFICACY OF THE PROPOSED CIRCUIT-RECOVERY (CR) ATTACK AGAINST THE PRIOR STATE-OF-THE-ART ATTACKS

Attack \ Defense	SARLock	Anti-SAT	TTLock	SFLL-HD	SFLL-flex	SFLL-rem	CASLock	MCAS	SAS	Gen-Anti-SAT		CAC	DTL		
										Comp.	Non-Comp.		SARLock	Anti-SAT	CAC
SPS/ATR [23]	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Bypass [24]	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
GNNUnlock [33]	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
CASUnlock [32]	✗	✓	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗
This Work	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

✓ Successful attack ✗ Unsuccessful attack

C. Motivation and Research Challenges

As discussed, there has been a (never-ending) cat-and-mouse game in PSLL research for half a decade. *All the attacks target the hardware implementation of the PSLL technique rather than the underlying provably-secure algorithm.* We note the similarity to the vulnerability of AES hardware implementation to side-channels (power, timing, etc.) despite its strong algorithmic security. The fact that this cat-and-mouse game uncovers vulnerabilities highlights an essential point that although the locking algorithms might be *provably-secure*, implementing these techniques in hardware using computer-aided design (CAD) tools leaves structural hints for an opportune attacker. *Thus, it is pivotal that we steer our efforts in formulating a structural-analysis-centric diagnostic tool that informs the designer about potential vulnerabilities in the hardware implementation of PSLL techniques.*

Thus, we identify the following research challenges (**RC**).

RC1 Can we identify properties that are generic and common to all PSLL techniques?

RC2 Can we develop a security diagnostic tool that can successfully identify (and inform the designer of) potential structural vulnerabilities in the hardware implementation of PSLL techniques before the design is taken to silicon?

RC3 Can we show that this security diagnostic tool is accurate, *i.e.,* adversaries can exploit any vulnerability it identifies? In other words, we would like this tool to be free of false positives. Can we formulate a generic attack, equivalent in strength, to the security diagnostic tool?

D. Scope of This Work and Our Research Contributions

In this work, we tackle the aforementioned research challenges and develop a security diagnostic tool *Valkyrie*, that diagnoses the hardware implementation of PSLL techniques for *structural vulnerabilities.* Additionally, we also showcase our proposed circuit recovery attack. Figure 1 illustrates the high-level idea of our work; (a) highlights the diagnosis part, and (b) highlights the exploit part. *Our work is analogous to bug-detection tools, where researchers first detect bugs and then present exploits to highlight the ramifications of the bugs.*

The primary contributions of our work are as follows.

1) We propose a holistic and generic security diagnostic tool (*Valkyrie*) for a security-enforcing designer that can identify vulnerabilities in the considered PSLL techniques from the standpoint of structural attacks. **To the best of our knowledge, this is the first work that proposes a security diagnostic tool for PSLL techniques.** To that end, we successfully identify vulnerabilities in 15 PSLL techniques, including eight unbroken techniques (highlighted in *italics* in Table I) using principles of graph theory, equivalence checking, and VLSI testing.
2) We propose a generic circuit-recovery (CR) attack to demonstrate that adversaries can exploit these identified vulnerabilities to recover the original functionality from the locked design. **To that end, we showcase the efficacy of our attack on 15 PSLL techniques, including eight unbroken ones**. Our CR attack advances the state-of-the-art by being generic as opposed to other attacks (e.g., Bypass [24], SPS [23]), which have been predominantly locking-technique specific (Table II).
3) We showcase the efficacy of *Valkyrie* and CR attack by conducting experiments on more than $20,000$ locked designs. We investigate the role played by different technology libraries and industry-standard and academic synthesis tools tuned with varying synthesis parameters and show that our tool and attack both are always successful *irrespective* of the underlying technology and flow. **Our analysis highlights the inadequacies of the security-agnostic CAD tools, especially for PSLL techniques.**
4) We develop two versions, one using academic tools and the other using industry-standard tools. We open-source the *Valkyrie* tool and other associated artifacts to enable the logic locking community towards the development of secure PSLL implementations devoid of structural vulnerabilities.[8]

The organization of the paper is as follows. First, we provide detailed background on adversary models and the considered PSLL techniques in Sec. II. We then propose the security diagnostic tool in Sec. III and expound on our proposed circuit-recovery attack in Sec. IV. We present detailed experimental results in Sec. V followed by a discussion in Sec. VI. Finally, we provide concluding remarks in Sec. VII.

II. BACKGROUND AND RELATED WORK

A. Adversary Model and Objectives

We will first outline the resources available to an adversary, which have been widely adopted by researchers in the logic locking community [8, 10–12, 26].

1) An adversary obtains the locked design either by reverse-engineering the layout and mask information (provided by the design house to the foundry for fabrication) or by reverse-engineering the chip (adversary is the end-user).

[8]https://github.com/LL-Tools/Valkyrie

2) An adversary has access to an activated chip (bought from open market) on which input patterns can be applied and corresponding outputs can be observed (adversary is the end-user). The activated chip is referred to as an *oracle*.

Using the resources mentioned above, we outline the adversary model as follows.

1) **Oracle-less model:** An adversary only uses the locked design to launch the attack.
2) **Oracle-guided model:** An adversary uses the locked design and the oracle to launch the attack.

Next, we outline some of the standard assumptions made by researchers in the logic locking community [8, 26].

1) The adversary knows the logic locking technique implemented in the design.
2) The adversary can distinguish between the regular primary inputs and the key-inputs.
3) An adversary is not restricted in the number of queries they can make to the oracle and they are not required to reveal any query in advance to the defender [34].

Adversarial Objective: An attacker aims to retrieve the original IP from the locked design. This is termed a circuit-recovery (CR) attack. There are two important steps in any CR attack; viz., (i) identifying the critical signal(s) (cs) separating the original design from the locking unit, and (ii) obtaining the logic value of the critical signal(s) (csv).

B. Provably-Secure Logic Locking (PSLL)

A crypto-system exhibits *provable security* if there exist mathematical proofs showing resilience to certain types of attacks [35]. Therefore, a logic locking technique exhibits provable security if it remains algorithmically secure against I/O query-based attacks (e.g., SAT-based attack or any key-pruning attack) under the aforementioned adversary model and the assumptions outlined below.

1) An adversary having access to an oracle and the reverse-engineered locked design must face exponential complexity in his/her efforts to retrieve the secret key with regards to the key-size, *i.e.,* he/she requires 2^k queries (to the oracle) to recover a k-bit key.
2) An adversary can inspect the locked netlist but is restricted from probing the oracle.

Thus, if a logic locking technique maintains resilience against the aforementioned adversary model and assumptions and demonstrates exponential complexity against I/O query-based attacks, we refer to it as a provably-secure logic locking (PSLL) technique [13, 14, 36]. These techniques are referred to as "provably secure" because they are algorithmically secure against I/O query-based (oracle-guided) attacks.

Next, we provide a brief background about the PSLL techniques considered in this work. First, we categorize them into *single-flip*, *double-flip*, and *multi-flip* locking techniques. Single-flip locking techniques (Fig. 2(a)) comprise a *standalone critical signal* (cs) generated by the locking circuit to corrupt the protected output for incorrect keys. On the other hand, double-flip locking techniques (Fig. 2(b)) include *two critical signals*, $cs1$ generated by the restore unit and $cs2$ generated by the perturb unit, which unconditionally corrupts the protected output and restores it for only the correct key. Multi-flip locking techniques contain more than two critical signals. The commonly used abbreviations in this work are shown in Table III.

TABLE III
COMMONLY USED ABBREVIATIONS

Term	Definition	Term	Definition
AIG	AND-INV-Graph	I/O	Input/output
ATPG	Automatic test pattern generation	K	Key-size
ATR	AND-tree removal	KI	Key-inputs
CAC	Corrupt-and-correct	MCAS	Mirrored CAS
CASLock	Cascaded locking	MFLT	Multi-flip locking technique
CNF	Conjunctive normal form	PI	Primary input
CR	Circuit recovery	PIP	Protected input port
cs	critical signal	PO	Primary output
csv	critical signal value	POP	Protected output port
CCS	Candidate critical signals	PP	Protected pattern
DFLT	Double-flip logic locking technique	PSLL	Provably-secure logic locking
DIP	Distinguishing input pattern	SARLock	SAT-attack resistant logic locking
D_{locked}	Locked design	SAS	Strong Anti-SAT
D_{orig}	Original design	SAT	Boolean satisfiability attack
DTL	Diversified tree logic	SFLL	Stripped functionality logic locking
Gen-Anti-SAT	Generalized Anti-SAT	SFLT	Single-flip locking technique
GNNUnlock	Graph neural network unlock	SPS	Signal probability skew
HD	Hamming distance	TTLock	Tenacious and traceless logic locking

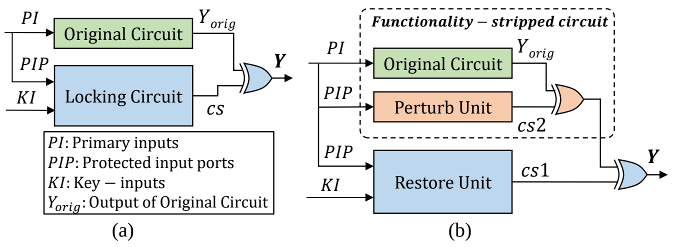


Fig. 2. High-level architecture of (a) single-flip locking technique (SFLT) (b) double-flip locking technique (DFLT). The nets cs, $cs1$, and $cs2$ are the critical signals.

C. Single-flip Locking Techniques (SFLT)

SARLock [14] was proposed to thwart the SAT-based attack [10]. The construction includes a comparator and a masking circuit connected to the original design via a single critical signal. The secret key is hard-coded in the masking circuit and compared with the key from tamper-proof memory.

Gen-Anti-SAT [37] includes two key-controlled functions, f and g, which are ANDed together to produce a single critical signal. Gen-Anti-SAT has two variants, (i) complementary and (ii) non-complementary. Note that f can be any Boolean function. g can either be the complement of f (in complementary Gen-Anti-SAT) or completely different (in non-complementary Gen-Anti-SAT). Such a construction introduces unpredictability in the locking structure, which cannot be exploited by the existing structural attacks [23, 24]. Without knowing the underlying functions f and g, the signal probability of the critical signal cannot be computed. Anti-SAT [13] and CASLock [31] are special cases of complementary Gen-Anti-SAT.

The construction of **Anti-SAT** [13] constitutes two functions, g (AND-tree) and $\overline{g}$ (NAND-tree), which are ANDed together to form a single critical signal. This signal is then merged with the original design. In **CASLock** [31], the g block contains cascaded AND-OR gates. Thus, the strength of CASLock stems from the introduction of OR gates in the g and $\overline{g}$ blocks. It can thwart the bypass attack by changing the location and number of AND/OR gates in g and $\overline{g}$ blocks.

D. Double-flip Locking Technique (DFLT)

TTLock [25] was the first DFLT proposed to thwart the SAT-based attack and removal attacks. It modifies the original design for *exactly* one input pattern. The modified design is then corrected using a key-controlled restore unit. Removing the restore unit results in a modified design instead of the original design in SARLock and Anti-SAT. Yasin *et al.* [25] generalized this concept into *stripped functionality logic locking (SFLL)*, a family of techniques, explained next.

SFLL-HD-h [25] modifies the original design for c input patterns, where c is $\binom{k}{h}$, k is the key-size and h is the Hamming distance (HD). Note that k and h dictate the resilience trade-off between the SAT-based and removal attacks. SFLL-HD-h protects a specific set of input patterns that are an HD of h away from the secret key.

SFLL-flex$^{c\times k}$ [25] provides the flexibility of protecting user-defined input patterns, which are hard-coded into the original design to produce the modified design. The restore unit stores these protected patterns (PPs) in c input cubes, each of k-bit size. These patterns essentially form a part of a (c*k)-bit secret key used to restore the modified functionality.

SFLL-rem [28] is another variant in the SFLL family. Unlike previous variants, the original design is modified by *removing* logic instead of adding hard-coded input pattern(s). A fault is inserted at a net connected to at least k input ports, where k is the key-size. The entire logic cone corresponding to this fault-infected net is removed. The test patterns detecting this fault are computed and one pattern amongst them is chosen as the PP. The modified functionality is restored for all the patterns except for this PP, which becomes the secret key. This modified functionality is then corrected using a key-controlled restore unit.

Corrupt-and-correct (CAC) [36] corrupts the original design using hard-coded AND-trees and corrects the modified functionality using a generic correction (restore) unit. One primary difference between TTLock and CAC is the inclusion of hard-coded AND-tree in the *correction* unit.

Diversified tree logic (DTL) [36] helps in increasing the on-set[9] size of the hard-coded structures, thereby hindering removal attacks. The designer can replace some of the gates in the AND-tree with OR/NAND/XOR gates to control this on-set size, which constructs the DTL structure. Replacing t gates in the first layer of the AND-tree with OR/NAND (XOR) gates increases the on-set from 1 to 3^t (2^t). DTL can be integrated with other removal attack vulnerable techniques such as SARLock and Anti-SAT [36].

Mirrored-CAS (MCAS) [31] consists of two CAS blocks, where one has hard-coded k_{secret} acting like a perturb/corrupt unit and the other is a key-controlled restore unit. The primary difference between TTLock and MCAS is that the AND-tree utilized in the former is replaced by an AND-OR cascaded structure in the latter, thereby thwarting the SPS attack [23].

[9]On-set size of a logic cone corresponds to the number of input patterns resulting in an output '1.' For an AND-tree, the on-set size is 1.

E. Multi-flip Locking Technique (MFLT)

Strong Anti-SAT (SAS) [38] ensures that, given any incorrect (including approximate) key, the error injected by the locking circuitry will have a significant application-level impact. SAS is also provably-secure against SAT-based attacks. A set of minterms (having a higher application-level impact) are identified as critical minterms. While most PSLL techniques have an inherent trade-off between security (*i.e.,* SAT-attack complexity) and effectiveness (*i.e.,* amount of error injected by an incorrect key), SAS ensures that an increase in effectiveness does not compromise security against SAT-based attacks.

III. Valkyrie: A Security Diagnostic Tool

The purpose of *Valkyrie* is to diagnose the (hardware implementation of) PSLL design for any structural vulnerabilities that can potentially lead to a successful CR attack. For example, an attacker can recover the original design from the locked design by removing the locking circuit (for SFLT) or the perturb and restore units (for DFLT) as shown in Fig. 2. An attacker can identify the critical signal(s) and remove the logic cones driving these signal(s) in the locked design. The designer, on the other hand, can mimic the effect of logic cone removal by inserting logical fault(s)[10] on the critical signal(s).

Next, we explain the methodology of *Valkyrie* as a security diagnostic tool for a security-enforcing designer to proactively identify critical signal(s) before taking the design to silicon.

A. Important PSLL Properties

Note that the critical signal(s) may not always be connected to an X(N)OR gate (Fig. 2 is a representative example). Consider the example shown in Fig. 3(a) and Fig. 3(b), which corresponds to a design locked using Anti-SAT and its subsequently synthesized version. The critical signal connected to an XOR gate in Fig. 3(a) cannot be traced in Fig. 3(b). However, a structural vulnerability exists (red shaded gate). Through our exploratory experiments, we observed that critical signals could be connected to any logic gate (Fig. 3(c)) and hence we propose a property-driven approach for *Valkyrie*.

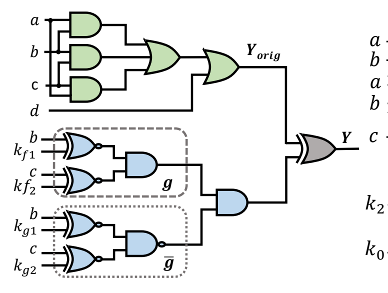
To that end, we first extract properties that are common to the considered PSLL techniques to enable us to develop a generic security diagnostic tool. *If modifications are made to future PSLL constructions, relevant properties must be updated in the Valkyrie security diagnostic tool.*

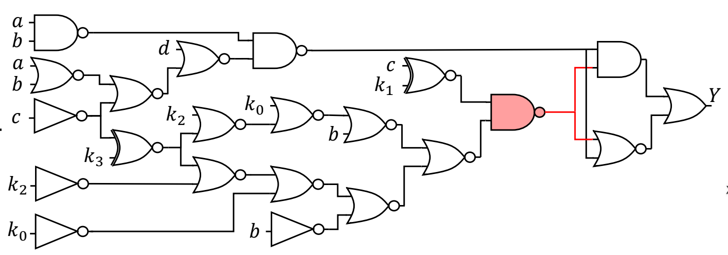
The assets available to the designer are (i) the original design and (ii) the locked design. Let the original design be denoted by D_{orig} and the locked design by D_{locked}. Also, let $D_{locked(cs,i)}$ be the locked design with a stuck-at-i fault inserted on net cs, where i can be 0 or 1; effectively, the net cs in D_{locked} is tied to logic value i.

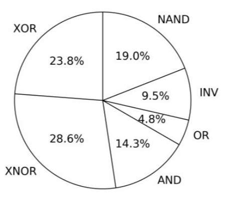
In SFLT, there is only one critical signal cs, which must be tied to the correct value to recover the original design:

Property 1: $D_{orig} = D_{locked(cs,i)}$, $\exists(cs,i)$: $cs \in Nets$, $i \in \{0,1\}$

[10]Physical fault denotes fault-injection on a chip using laser and micro-probing. Logical fault denotes tieing a net to a logical value in a gate-level netlist for simulation.

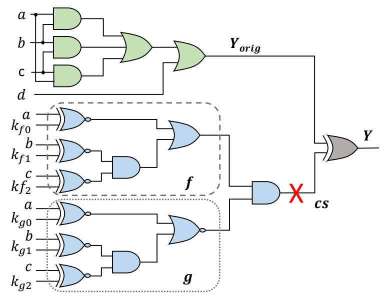

(a)

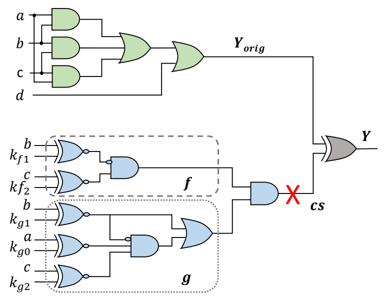

(b)



(c)

Fig. 3. (a) Locked design using Anti-SAT. (b) Locked design (post-synthesis). The XOR locking gate (dark grey) is disintegrated and hence no candidate critical signal is identified if one were to exclusively test the fan-ins of the XOR gates. Following such an approach will result in false negative, *i.e.,* a vulnerable circuit would be termed secure. (c) Pie-chart showing distribution of logic gates connected to the critical signal.


(a)


(b)

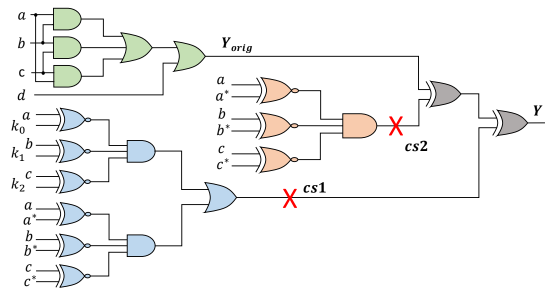

(c)

Fig. 4. (a) Gen-Anti-SAT (complementary) (b) Gen-Anti-SAT (Non-complementary) (c) Corrupt-and-Correct (CAC). The gates in the original design are denoted in green and the key-controlled locking unit is shown in blue; the output of the locking unit forms the critical signal cs. For CAC, the hard-coded structure is denoted in orange; the output forms the critical signal $cs2$, while the output of the locking unit forms the critical signal $cs1$. The original design is appended with the hard-coded structure and the locking unit at these critical signals using XOR gates (shown in gray) to form output Y.

Property 2: The critical signal in SFLT must be connected to only key inputs and the primary inputs that drive the locking circuit.

Property 3: The critical signal in SFLT must affect only the protected output ports.

Figures 4(a-b) illustrate the complementary and non-complementary versions of Gen-Anti-SAT. The critical signal $cs1$ satisfies Property 2 and 3 as it is connected to only key inputs (k_f, k_g bits) and the primary inputs relevant to locking (a, b, c), and affects only the protected output port Y.[11]

In DFLT, there are two critical signals, $cs1$ and $cs2$, which must be tied to the correct value to recover the original design.

Property 4: $D_{orig} = D_{locked(cs1,cs2,i,j)}$, $\exists(cs1, cs2, i, j)$: $cs1, cs2 \in Nets$, $i, j \in \{0, 1\}$

Property 5: One of the critical signals $cs1$ must be connected to only key inputs and the primary inputs that drive the restore unit.

Property 6: The other critical signal $cs2$ must be connected to only the primary inputs that drive the perturb unit.

Property 7: Both critical signals $cs1$ and $cs2$ must affect only the protected output ports.

Property 8: Critical signal(s) must not remove any primary inputs when fault(s) is(are) inserted.

Figure 4(c) illustrates the CAC technique. Nets $cs1$ and $cs2$ satisfy all the properties, viz., $cs1$ is connected to only the key inputs (k_0, k_1, k_2) and the primary inputs involved in locking (a, b, c), and affects only the protected output port (Y), while $cs2$ is connected to only the primary inputs involved in locking (a, b, c) and affects only the protected output port (Y).

[11]In case of non-complementary Gen-Anti-SAT technique, reduced key-size is observed for f block adhering to the algorithm proposed in [37].

Generalization of PSLL properties: The properties outlined above stem from the inherent construction of the considered PSLL techniques and can be seamlessly extended to MFLT. Recall that the SAT-based attacks are thwarted by forcing the attack to rule out exactly one incorrect key per attack iteration. Since point-functions (e.g., AND-tree) help satisfy the aforementioned condition, most PSLL techniques use point-functions in their construction. The point-functions modify the functionality for selected input patterns (perturb unit), thereby restricting their dissolution (during synthesis) with the original function. Thus, structural hints are left behind, which can be identified using these properties.

B. Methodology

Problem formulation: Given an original design D_{orig} and a locked design D_{locked}, find a critical signal cs (or pair of critical signals) and its (their) logical value(s) csv = {0, 1}, which facilitate the functional recovery of the original design as per Property 1 and Property 4.

Approach: We insert faults (stuck-at-0 or stuck-at-1) at a chosen net in the locked design and then compare this modified functionality with the original design. As per Property 1 and Property 4, if the correct net(s) is(are) identified as the critical signal(s) and the correct value(s) is(are) identified for the net(s), then the locked design implements the same functionality as the original design.

A naïve formulation would require a designer to insert a stuck-at-0 and a stuck-at-1 fault at each location in the locked design and compare the functionality with the original design. Doing so requires $2n$ calls in SFLT and $4\binom{n}{2}$ calls in DFLT to the equivalence checker, where n is the number of nets in the fanin of the protected output port. We address this challenge (reducing the algorithmic complexity) by utilizing the properties to prune the set of candidate critical signals (CCS) and then proceed with the fault insertion and equivalence checking only on the candidate signals that satisfy the properties.

Consider Fig. 2(a), where cs is the critical signal in the locked design, identified using the outlined properties for SFLT. Tieing this net to logic 0 returns the original design. Next, in Fig. 2(b), there are two critical signals $cs1$ and $cs2$ in the locked design, which is identified using the properties for DFLT. Tieing these nets to logic 0 returns the original design. However, synthesis-induced optimizations can result in netlist restructuring, bubble-pushing, etc.; e.g., an XOR gate can transform into an XNOR gate. Figure 3 demonstrates an example of one such synthesis transformation, where the critical signal is no longer connected to an XOR gate. Critical signal polarity may also change in such cases, necessitating identifying the correct value(s) to tie the critical signal(s) to. We address this challenge by performing a functional equivalence check between the fault-inserted locked design and the original design for both faults on the critical signal.

The problem of functional equivalence checking can be mapped to a *Boolean satisfiability (SAT)* problem, as explained next. We construct a miter M from D_{orig} and $D_{locked(cs,i)}$ and the output of M as: *out* = $D_{orig} \oplus D_{locked(cs,i)}$. Signal *out* evaluates to 1 iff the outputs of $D_{locked(cs,i)}$ and D_{orig} result in different logical values for *at least* one input pattern. Therefore, the problem of checking the equivalence between D_{orig} and $D_{locked(cs,i)}$ is analogous to testing the satisfiability of this miter M. The miter is converted to a conjunctive normal form (CNF) F using *Tseitin* transformations and fed to a SAT solver.[12] If the solver returns *SAT*, then D_{orig} and $D_{locked(cs,i)}$ are functionally non-equivalent. However, if the solver returns *UNSAT*, it implies that no input pattern satisfying *out* exists and that the two designs D_{orig} and $D_{locked(cs,i)}$ are functionally equivalent.

Practicality: Fault insertion and functional equivalence checks are regular operations that a security-enforcing designer can perform. These operations enable designers to check if they identified the correct critical signal(s) and the correct value(s). More importantly, the designer has the luxury of having access to both the locked design and the *original design*, which the logic locking attack threat model assumes that attackers do not have access to. While the designer can check if the locked design has structural vulnerabilities that would lead to the recovery of the original design, one may question the validity of this diagnostic tool that relies on the use of the original design, which is not available to the attackers. We revisit this point in Sec. IV, to show that this is not that different from utilizing an oracle, which attackers are assumed to have access to, *concluding that the validity analysis is justifiable and practical.*

[12]Given a CNF X, the satisfiability problem entails finding an assignment to the variables of X for which X evaluates to 1 (defined as a satisfying assignment) or proving that such an assignment does not exist.

C. Function Definitions

Here, we define the common functions used in the algorithms and subsequent sections of this work.

- *PIP*: A subset of primary inputs involved in a comparison with the key inputs to produce a critical signal in the locked design are termed as protected input ports. For e.g., in Fig. 4(a), the input ports a, b, and c are *PIPs*.
- *POP*: A subset of primary outputs protected by the locking technique are defined as protected output port(s). For e.g., in Fig. 4(a), Y is the *POP*.
- `fanout_endpoints(`net`)`: Returns a list of primary outputs which lie in the fanout of signal net. For e.g., in Fig. 4(a), it returns output Y for signal $cs1$.
- `fanin_startpoints(`net, in`)`: Returns a subset of primary inputs in which lie in the fanin of signal net For e.g., in Fig. 4(a), it returns inputs $\{a, b, c\}$ for net $cs1$ and in = $\{a, b, c, d\}$.
- `fanin_nets(`in, y`)`: Returns a topologically sorted list of signals controlled by all inputs in and affecting only and all primary outputs y.
- `fault(`net, i`)`: Inserts a fault i (stuck-at-0 or stuck-at-1) at signal net and returns this fault-inserted design.
- `ATPG(`net, i`)`: Returns a pattern which detects stuck-at fault i at signal net This pattern is denoted as a *test pattern*. Returns NULL, if ATPG cannot generate any test pattern.
- `A.append(B)`: Adds item B to list or array `A`.
- `len(A)`: Returns the number of elements in array `A`.

D. Algorithm

The tool takes two inputs: a locked design D_{locked} and an original design D_{orig}. For SFLT, `StructAnalysis()` (lines 1–5 in Alg. 1) identifies the $POPs$ and $PIPs$, using graph traversal. This information is fed to `CriticalSignal()` (lines 6–13), which, based on the properties outlined before, shortlists a set of CCS. To further prune out incorrect critical signals and fault values, we utilize `EquiCheck()` (lines 17–19). We iteratively construct a fault-inserted locked design ($D_{locked(cs_i,j)}$) using the candidate critical signal cs_i and fault value $j \in \{0, 1\}$. This design is fed along with the original design to `EquiCheck()`. It first constructs a miter from the fault-inserted locked design and the original design and then, using `ATPG()` generates test patterns to detect the stuck-at fault. If `ATPG()` returns *NULL*, then the fault-inserted locked design $D_{locked(cs_i,j)}$ and the original design D_{orig} are functionally equivalent and the corresponding critical signal cs_i and critical signal value j are returned; otherwise they are functionally non-equivalent. The algorithm is similar for DFLT; the DFLT properties are utilized instead.

Example 1: Here, we demonstrate the applicability of the tool on a **synthesized version of the complementary Gen-Anti-SAT technique.** Figure 5(a) illustrates the AND-Inverter-Graph (AIG) for Gen-Anti-SAT shown in Fig. 4(a). We shortlist the nets marked in blue as the candidate set CCS using the properties to determine the critical signal $cs1_i$. Although there are two candidate signals in CCS, one of them

Algorithm 1: *Valkyrie* Security Diagnostic Tool

```
Input: Original design ($D_{orig}$), Locked design ($D_{locked}$),
       Primary inputs ($PI$), Key-inputs ($KI$)
Output: Critical signals $CS$, Critical signal values $CSV$
1  procedure StructAnalysis(keyin,in):
2      $POP \leftarrow$ fanout_endpoints(keyin)
3      $nets \leftarrow$ fanin_nets(keyin,POP)
4      $PIP \leftarrow$ fanin_startpoints(nets,in)
5      return $POP$, $PIP$
6  procedure CriticalSignal(in,y):
7      $NETS \leftarrow$ fanin_nets(in,y)
8      for $net \in NETS$ do
9          $pi\_net \leftarrow$ fanin_startpoints(net,in)
10         $co\_nets \leftarrow$ fanout_endpoints(net)
11         if (co_nets == y) && ($pi\_nets \subseteq in$) then
12             $CCS$.append(net)
13     return $CCS$
14 procedure Miter(d1,d2,y):
15     $out \leftarrow \bigvee_{i \in y} (d1_i \oplus d2_i)$
16     return out
17 procedure EquiCheck(d1,d2,y):
18     $out \leftarrow$ Miter(d1,d2,y)
19     return $\overline{\text{ATPG}(out, 0)}$
20 function Valkyrie:
21     $\{POP, PIP\} \leftarrow$ StructAnalysis($KI$,$PI$)
       // Single-flip identification
22     $\{CCS\} \leftarrow$ CriticalSignal($KI$,$POP$)
23     for $cs_i \in CCS$ do
24         $D_{locked(cs_i,j)} \leftarrow$ fault($cs_i, j$) $\forall$ j $\in$ {0,1}
25         if EquiCheck($D_{orig}$,$D_{locked(cs_i,j)}$,$POP$) then
26             $CS$.append($cs_i$)
27             $CSV$.append($j$)
28             return $CS$, $CSV$
       // Double-flip identification
29     $\{CCS1\} \leftarrow$ CriticalSignal($KI$,$POP$)
30     $\{CCS2\} \leftarrow$ CriticalSignal($PIP$,$POP$)
31     for $cs1_i \in CCS1$ do
32         for $cs2_j \in CCS2$ do
33             $D_{locked(cs1_i,cs2_j,k_1,k_2)} \leftarrow$
                 fault($cs1_i, cs2_j, k_1, k_2$) $\forall$ $k_1$, $k_2$ $\in$ {0,1}
34             if
                 EquiCheck($D_{orig}$,$D_{locked(cs1_i,cs2_j,k_1,k_2)}$,$POP$)
                 then
35                 $CS$.append($cs1_i$,$cs2_j$)
36                 $CSV$.append($k_1$,$k_2$)
37                 return $CS$, $CSV$
```

is connected to an inverter and hence can be excluded from the list, as only the critical signal value will be different. A fault-inserted locked design is constructed for $cs1_1$ net. Tieing this net to logic 0 makes the fault-inserted locked design functionally equivalent to the original design.

Example 2: Next, we showcase the applicability of the tool on a **synthesized CAC-locked design**. Figure 5(b) illustrates the AIG for the CAC technique shown in Fig. 4(c). Although there are four candidate signals in $CCS1$ and $CCS2$, two are connected to inverters and can be excluded from the list. The remaining candidates/nodes are used to construct a fault-inserted locked design. Tieing the nets $cs1_1$ to 0 and $cs2_1$ to 1 makes the fault-inserted locked design equivalent to the original design. Note that the value of $cs2_1$ has changed from 0 to 1 due to synthesis-induced transformations.

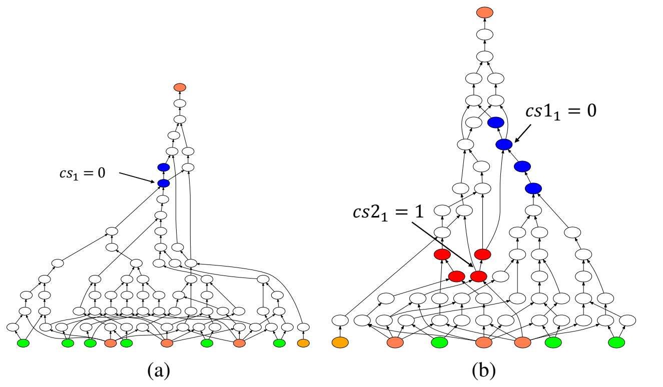


Fig. 5. $PIPs$ and non-PIP are shown in coral and orange. KIs are shown in green. (a) AND-Inverter-Graph for complementary Gen-Anti-SAT technique. Nodes shown in blue correspond to CCS set. (b) AND-Inverter-Graph for CAC technique. Nodes shown in blue correspond to $CCS1$ set and nodes shown in red correspond to $CCS2$ set.

IV. A Generic Circuit-Recovery Attack

Recall that the designer is equipped with the most precious asset, the original design, to carry out the security diagnosis. In this section, we show that the availability of the original design is not *pivotal* and that it can be efficiently replaced with an oracle to realize a practical attack. We showcase how an attacker can exploit the identified vulnerabilities by launching our circuit-recovery attack with assets that an attacker normally has, *i.e.,* the locked design and the oracle Orc.[13]

Problem formulation: Given a locked design D_{locked} and a black-box oracle Orc, an attacker aims to find a critical signal cs (or pair of critical signals) and its (their) logical value(s) $csv = \{0, 1\}$, which facilitates the recovery of the original design, D_{orig}. Although the problem formulation is the same as mentioned in Sec. III, there is one significant difference; *an oracle now replaces the original design.* Thus, an attacker can now use the oracle Orc for querying I/O pairs.

Challenges: While an attacker can utilize the principles outlined in Sec. III to obtain a set of candidate critical signals from the locked design, she faces the following challenges.

RC4 How to identify the actual critical signal (or critical signal pair) among all these candidates?

RC5 Furthermore, how to compute the value of this critical signal (or critical signal pair) to verify the correctness of this solution, all in the *absence of the original design*?

RC6 Finally, would the effectiveness of this attack be equivalent to that of the tool proposed in Sec. III *despite the replacement of the original design with the oracle?*

Theorem 1. *A correct solution (cs, csv) verified by utilizing the original design can always be verified by utilizing the oracle instead of the original design.*

Proof. Given that the correct solution (cs, csv) lies in the candidate solution list, a special set of input patterns can be computed to prune out the incorrect candidate solutions. Consider an input pattern generated to compare two candidate solutions (CS_1, CS_2) such that $CS_1 \neq CS_2$. Now, querying the oracle for this input pattern will definitely eliminate one of

[13]Circuit recovery is analogous to function recovery (FR) of the design.

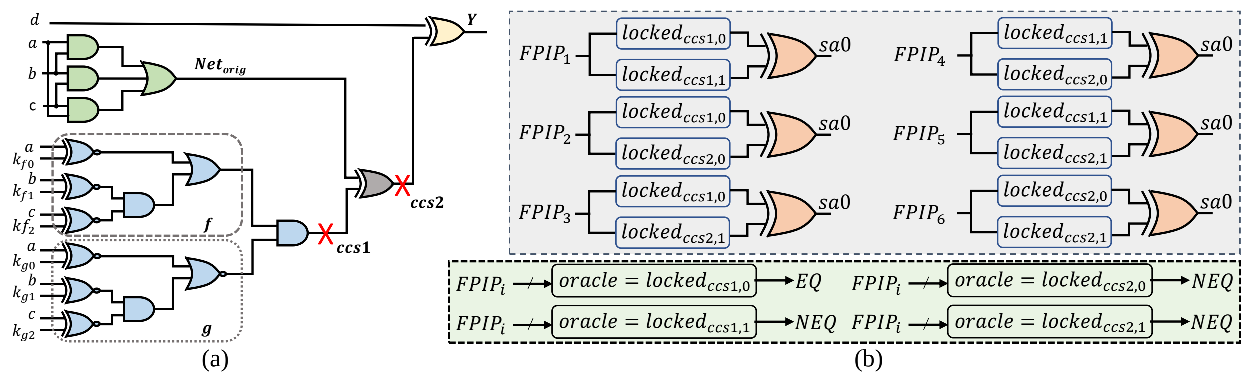


Fig. 6. High-level approach for the circuit-recovery attack. (a) Internal net Net_{orig} is locked using Gen-Anti-SAT. Two candidate critical signals $ccs1$ and $ccs2$ are identified using structural properties. (b) Gray box shows the miter circuits for all combination of faults. Test patterns detecting stuck-at-0 ($sa0$) are collected—they are called fault-pruning input patterns ($FPIPs$). Green box shows the oracle step. EQ denotes that the fault-inserted locked design returns the same output as the oracle for all the $FPIPs$ and NEQ denotes non-equivalence.

TABLE IV
TIME COMPLEXITY OF THE FUNCTIONS USED IN THE *Valkyrie* SECURITY DIAGNOSTIC TOOL AND THE CIRCUIT-RECOVERY ATTACK

Function	Time complexity
$fault()$	$O(1)$
$append()$	$O(1)$
$fanout\_startpoints()$	O(V+E)
$fanout\_endpoints()$	O(V+E)
$fanin\_nets()$	O(V+E)
$StructAnalysis()$	O(V+E)
$CriticalSignal()$	O(P)
$Valkyrie()$ $(SFLT)$	$O(N)$
$Valkyrie()$ $(DFLT)$	$O(N*M)$
$CircuitRecovery()$ $(SFLT)$	$O(N^2)$
$CircuitRecovery()$ $(DFLT)$	$O((N*M)^2)$

V: vertices (gates) E: edges (nets) P: # nets in the output logic cone
N: # of $CCS1$ M: # of $CCS2$

the two candidate solutions. This pattern generation process is repeated until all the incorrect candidate solutions are pruned out, leaving behind the correct solution (cs, csv). Thus, if a correct solution was verified using the original design, it can also be verified using an oracle for a few input patterns. □

Approach: Figure 6 illustrates the high-level approach of the proposed CR attack. There are two candidate critical signals ($ccs1$ and $ccs2$), so there are a total of four fault-inserted locked designs (candidate solutions). A total of $\binom{4}{2} = 6$ miter-circuits are generated for comparing all fault-inserted locked designs with each other; test patterns detecting stuck-at-0 fault at the output of miter circuits is computed. We call these patterns as fault-pruning input patterns ($FPIPs$), since they will be used to prune out incorrect solutions. These $FPIPs$ are applied to the four fault-inserted locked designs and corresponding output responses are collected. Finally, using the oracle responses, all the incorrect solutions are pruned out and the correct fault-inserted locked design which passes equivalence (EQ) for all the six $FPIPs$ is identified.

Algorithm: Similar to the security diagnostic tool, an attacker obtains a set of candidate critical signals using `StructAnalysis()` and `CriticalSignal()`. Then using `fault()`, the attacker creates fault-inserted locked designs. Subsequently, the attacker uses the `Miter()` and `ATPG()` to obtain $FPIPs$. Finally, the oracle is queried with these $FPIPs$ to obtain correct responses; $FPIPs$ are also applied to fault-inserted locked designs to collect their responses. The fault-inserted locked design whose responses match the oracle for all the $FPIPs$ is returned as the recovered original design D_{orig}.

Generality & Feasibility: The proposed CR attack, just like the diagnostic tool, is based on the generic properties of the considered PSLL techniques and can be contrasted from the technique-specific attacks such as the SPS attack [23] that relies on signal probabilities to break Anti-SAT. The proposed CR attack makes minimal modifications to the GDSII—at most multiple signals have to be tied to the appropriate logic values. This is in contrast to the bypass attack [24] where additional logic cells have to be inserted to bypass the locked design, thereby making it a costly attack.

Time complexity: Table IV depicts the time complexity for functions used in *Valkyrie* security diagnostic tool and CR attack. The time complexity depends on the size of the design and, thus, on the number of signals (nets) in the design.

V. EXPERIMENTAL SETUP AND RESULTS

Initially, we demonstrate the efficacy of the security diagnostic tool, *Valkyrie*, in identifying structural vulnerabilities for 15 PSLL techniques. Next, we show that attackers can exploit the vulnerabilities to retrieve the original design (from a locked design) using the proposed CR attack.

A. Experimental Setup

Locking techniques: We implement 15 PSLL techniques which include SARLock [14], Anti-SAT [13], TTLock, SFLL-HD, SFLL-flex [25], SFLL-rem [28], CAC, SARLock-DTL, Anti-SAT-DTL, CAC-DTL [36], CASLock, MCAS [31], SAS [38], and complementary and non-complementary versions of Gen-Anti-SAT [37]. We implement these techniques using different programming languages such as C++ and *Python*. We implement the locking techniques on different abstraction levels (*BENCH*, RTL, and synthesized *Verilog* design). The locked designs have been checked for security guarantees, especially against the SAT-based attack [10], as highlighted in the respective publications.

Algorithm 2: Circuit-recovery (CR) Attack

```
Input: Oracle ($Orc$), Locked design ($D_{locked}$), Primary inputs ($PI$), Key-inputs ($KI$)
Output: $D_{orig}$
1  function CircuitRecovery:
2      $\{POP, PIP\} \leftarrow$ StructAnalysis($KI$,$PI$)
       // Single-flip technique
3      $\{CCS\} \leftarrow$ CriticalSignal($KI$,$POP$)
4      $cnt = 0$
5      for $cs_i \in CCS$ do
6          $cnt = cnt + 1$
7          $D_{locked(cnt)} \leftarrow$ fault($cs_i, 0$)
8          $cnt = cnt + 1$
9          $D_{locked(cnt)} \leftarrow$ fault($cs_i, 1$)
10     for $i \in \{1, cnt-1\}$ do
11         for $j \in \{i+1, cnt\}$ do
12             $out \leftarrow$ Miter($D_{locked(i)}$,$D_{locked(j)}$,$POP$)
13             $FPIP$.append (ATPG($out$, 0))
14     for $i \in \{1, cnt\}$ do
15         $comp = 0$
16         for $ip \in FPIP$ do
17             $Orc_{out} \leftarrow Orc(ip)$
18             $Locked_{out} \leftarrow D_{locked(i)}(ip)$
19             if $Orc_{out}$ == $Locked_{out}$ then
20                 $comp$++
21                 if $comp$ == len($FPIP$) then
22                     $D_{orig}$ = $D_{locked(i)}$
23                     return $D_{orig}$
24             else
25                 break
       // Double-flip technique
26     $\{CCS1\} \leftarrow$ CriticalSignal($KI$,$POP$)
27     $\{CCS2\} \leftarrow$ CriticalSignal($PIP$,$POP$)
28     $cnt = 0$
29     for $cs1_i \in CCS1$ do
30         for $cs2_i \in CCS2$ do
31             for $f1 \in \{0, 1\}$ do
32                 for $f2 \in \{0, 1\}$ do
33                     $cnt = cnt + 1$
34                     $D_{locked(cnt)} \leftarrow$ fault($cs1_i, cs2_i, f1, f2$)
35     for $i \in \{1, cnt-1\}$ do
36         for $j \in \{i+1, cnt\}$ do
37             $out \leftarrow$ Miter($D_{locked(i)}$,$D_{locked(j)}$,$POP$)
38             $FPIP$.append (ATPG($out$, 0))
39     for $i \in \{1, cnt\}$ do
40         $comp = 0$
41         for $ip \in FPIP$ do
42             $Orc_{out} \leftarrow Orc(ip)$
43             $Locked_{out} \leftarrow D_{locked(i)}(ip)$
44             if $Orc_{out}$ == $Locked_{out}$ then
45                 $comp$++
46                 if $comp$ == len($FPIP$) then
47                     $D_{orig}$ = $D_{locked(i)}$
48                     return $D_{orig}$
49             else
50                 break
```

Test cases: We study the effectiveness of our security diagnostic tool and proposed CR attack on eight combinational benchmarks from the ITC-99 suite. Each benchmark is locked 20 times for different key-sizes to account for variations in POs, PIs, and internal nets. We create a dataset of more than $20,000$ locked designs. Besides in-house locking, we also lock benchmarks using PSLL techniques from NEOS [39].

Security diagnostic tool setup: We develop two versions of the tool, one using academic tools and the other using industry-standard tools. We implement the academic version using a combination of *C++* and *Python* integrated with *ABC* [40] and *MiniSAT* [41]. We implement the industrial version using *TCL* and *Bash* integrated with *Synopsys Design Compiler* and *Cadence Conformal LEC*. Note that the academic version of the tool necessitates the files in a non-industry-standard format like *BENCH* since open-source tools like *ABC* and *MiniSAT* can process *BENCH* format. However, our industrial version works on synthesized *Verilog* designs.

Attack setup: We implement the CR attack using *TCL* scripts integrated with *Synopsys Design Compiler* and *Synopsys Tetramax*. Our attack can process *Verilog* files synthesized with all the gates available in the technology library. We perform our experiments on a 128-core Intel Xeon processor running at 2.4 GHz with 512 GB of RAM.

B. Results of Valkyrie Security Diagnostic Tool

Since *Valkyrie* aims to identify structural vulnerabilities, we evaluate the efficacy of the security diagnostic tool on locked benchmarks with varying structures. The underlying structure is governed by choice of synthesis tools, synthesis commands, technology libraries, types of logic gates used to synthesize the design, and the abstraction level. Next, we discuss the impact of different parameters on the efficacy of the security diagnostic tool.

Effect of technology library: We synthesize the locked designs using two technology libraries, one academic (Nangate 45nm) and one foundry-compatible *GlobalFoundries* 65nm process. The rationale behind using different libraries is as follows: different libraries contain varied types of logic gates, which influence the decisions made by the synthesis tool to generate designs having different structural representations.

Effect of synthesis tool: The choice of the synthesis tool results in different structural representations of a locked design due to the different types of optimization algorithms. We synthesize the locked designs using two commercial synthesis tools (i) *Synopsys Design Compiler* (`M-2016.12-SP2`), and (ii) *Cadence Genus* (`17.13-s033_1`).

Execution time: Table V illustrates the average execution time (over 20 trials) for our tool to identify structural vulnerabilities in 15 PSLL techniques. The benchmarks are locked with a key-size of 128 for SARLock, TTLock, SFLL-HD, SFLL-flex, and CAC, while a key-size of 256 is chosen for Anti-SAT, CASLock, MCAS, and Gen-Anti-SAT. For SFLL-HD, h is 16, for SFLL-flex, the number of PP is 16, and for SFLL-rem, a key-size of 80 is chosen [28]. For the DTL techniques, we diversify the AND-tree by replacing 16 gates in the AND-tree with OR gates. For the SAS experiments, we consider 4 SAS blocks and key-size of 512. We lock the ITC-99 benchmarks at RTL and synthesize them using two-input gates from the Nangate 45nm library [42]. The tool identifies vulnerabilities in all the benchmarks for all the considered PSLL techniques in $\sim$20 minutes for the

TABLE V

AVERAGE EXECUTION TIME (IN SECONDS) IN IDENTIFYING VULNERABILITIES FOR *Valkyrie* SECURITY DIAGNOSTIC TOOL FOR 20 RANDOM TRIALS ON ITC-99 BENCHMARKS USING 15 PSLL TECHNIQUES SYNTHESIZED USING TWO-INPUT GATES FROM NANGATE 45NM TECHNOLOGY LIBRARY. THE SUCCESS RATE IS 100% FOR ALL BENCHMARKS FOR ALL THE CONSIDERED PSLL TECHNIQUES

Defense / Circuit	SARLock	Anti-SAT	TTLock	SFLL-HD	SFLL-flex	SFLL-rem	CASLock	MCAS	SAS	Gen-Anti-SAT		CAC	DTL		
										Comp.	Non-Comp.		SARLock	Anti-SAT	CAC
b14_C	144	166	187	248	204	172	190	209	251	147	161	153	165	175	182
b15_C	169	165	217	457	290	198	264	234	299	148	160	162	181	171	168
b20_C	163	160	192	536	305	208	168	278	202	151	184	173	172	182	199
b21_C	163	164	204	365	292	295	284	307	337	240	137	173	173	190	185
b22_C	166	177	208	335	298	194	183	236	318	153	160	167	167	183	237
b17_C	182	182	243	458	364	410	256	420	489	288	294	247	205	371	367
b18_C	383	378	401	864	805	379	284	304	702	294	318	693	429	396	768
b19_C	923	381	615	1,196	911	475	307	512	631	428	354	803	1,006	389	865

TABLE VI

AVERAGE EXECUTION TIME (IN SECONDS) IN IDENTIFYING VULNERABILITIES FOR *Valkyrie* DIAGNOSTIC TOOL FOR 20 RANDOM TRIALS ON ITC-99 BENCHMARKS USING 15 PSLL TECHNIQUES SYNTHESIZED USING ALL AVAILABLE GATES FROM GLOBALFOUNDRIES 65NM TECHNOLOGY LIBRARY

Defense / Circuit	SARLock	Anti-SAT	TTLock	SFLL-HD	SFLL-flex	SFLL-rem	CASLock	MCAS	SAS	Gen-Anti-SAT		CAC	DTL		
										Comp.	Non-Comp.		SARLock	Anti-SAT	CAC
b14_C	725	730	803	1,210	1,118	1,053	760	940	1,134	823	846	822	823	798	844
b15_C	947	912	1,145	1,999	1,429	1,071	963	1,125	983	869	918	911	978	1,012	914
b20_C	801	823	926	1,900	1,509	1,019	901	1,411	1,027	909	810	799	826	853	800
b21_C	785	949	1,013	1,503	1,420	1,024	920	1,100	960	808	825	816	860	979	821
b22_C	900	974	1,011	1,603	1,505	1,009	900	1,049	1,062	960	991	907	917	994	1,243
b17_C	1,115	1,105	1,402	2,193	1,677	1,128	1,158	1,511	1,094	875	1,440	1,460	1,508	1,140	1,412
b18_C	997	1,034	1,453	3,203	2,816	1,419	960	1,410	1,198	924	1,040	2,505	1,069	1,418	2,823
b19_C	3,925	1,405	2,023	4,400	3,199	1,516	1,033	2,478	1,171	1,451	1,492	3,171	4,071	1,441	4,192

largest ITC-99 benchmarks (b18_C with 117,941 gates and b19_C with 237,962 gates). Table VI shows the execution time of our tool to identify vulnerabilities in locked designs synthesized with all the available gates for *GlobalFoundries* 65nm technology library. We observe a higher execution time (a little over 70 minutes for b19_C) for the 65nm technology library due to the different structural representations of the locked design (owing to the synthesis-induced changes) and the extra time required for loading technology libraries. **Our results highlight that there is no dependence on the choice of the library and type of gates available for synthesis on the execution time.**[14]

Effect of synthesis commands: Next, we evaluate the effect of different synthesis commands. For Synopsys DC, we used "`compile_ultra`," "`compile_ultra -incremental`," and "`optimize_netlist -area`." For Cadence Genus, we used the command "`synthesize -to_mapped`" with different efforts (medium and high). We also used *ABC*, an academic synthesis tool, and evaluated the effect of commands like `strash`, `logic`, and `refactor`.

Effect of abstraction level: We implement locking techniques at different abstraction levels viz., (i) *BENCH*, (ii) RTL, and (iii) synthesized Verilog. For (i), we verify the effectiveness of the diagnostic tool by invoking the academic version, while for (ii) and (iii), we leverage the industrial version.

Effect of key-size: Increasing the key-size for PSLL techniques increases the security level of the locked design [25]. This increase in key-size has no bearing on the effectiveness of *Valkyrie* in identifying structural vulnerabilities, apart from the negligible impact on the execution time.

[14]Due to lack of space, we do not show execution time for all the considered parameters—*Valkyrie* achieves 100% success in all considered test cases.

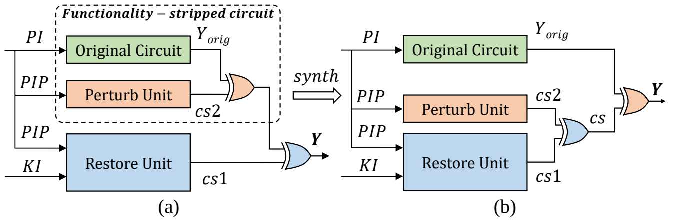


Fig. 7. Conversion from DFLT to SFLT post-synthesis (synth).

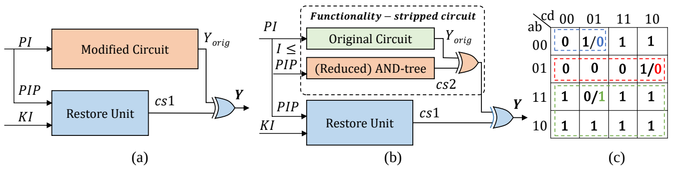


Fig. 8. (a) Intended implementation of DFLT (b) Speculated implementation of DFLT (CAC-DTL) (c) Karnaugh map showing cases for reduction in an AND-tree. $0/1$ denotes Y_{orig} is 0 and Y is 1. Choosing particular input pattern $ABCD$ reduces the AND-tree as shown in dashed rectangles.

Takeaway Message: Our experiments reinforce that *Valkyrie* identifies structural vulnerabilities in all the considered PSLL techniques (100% success) for all test cases generated with different variations (synthesis tools, synthesis commands, different technology libraries, various abstraction levels, and key-sizes).

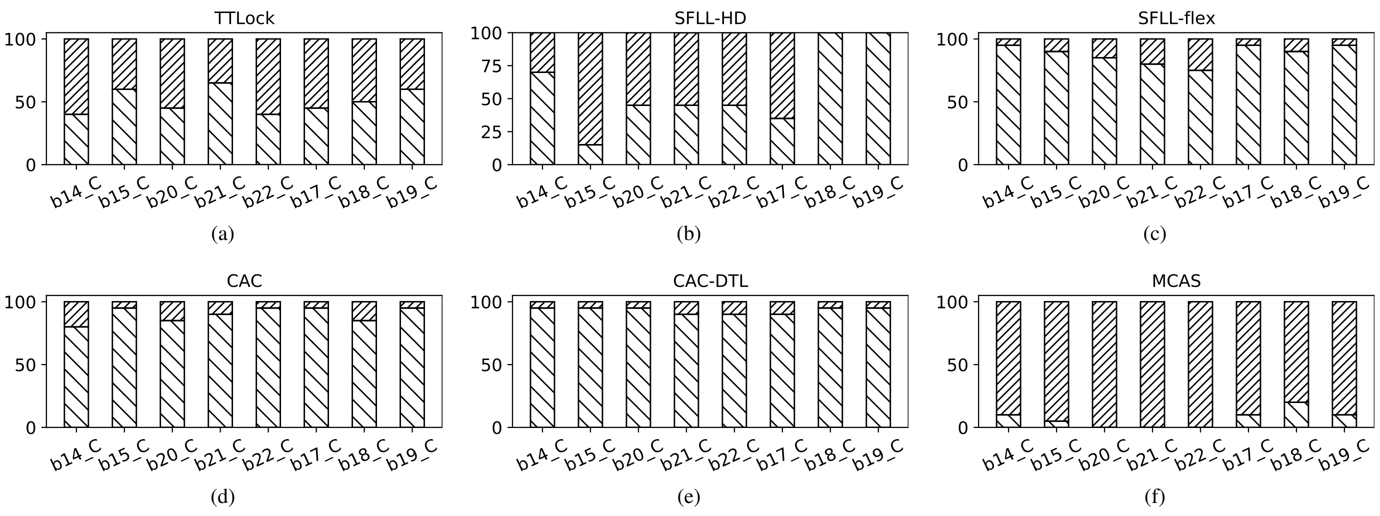


Fig. 9. Conversion from DFLT to SFLT for designs synthesized using all available gates in Nangate 45nm technology library. Synthesis-induced optimizations convert two critical signals ($cs1$, $cs2$) to a single flip signal (cs) (see Figure 7). Back-slash and forward-slash correspond to one and two critical signal(s).

TABLE VII
AVERAGE EXECUTION TIME (IN SECONDS) FOR THE **CIRCUIT-RECOVERY ATTACK** FOR ITC-99 BENCHMARKS. THE LOCKED DESIGNS ARE SYNTHESIZED USING ALL THE AVAILABLE GATES FROM THE NANGATE 45NM TECHNOLOGY LIBRARY

Circuit \ Defense	SARLock	Anti-SAT	TTLock	SFLL-HD	SFLL-flex	SFLL-rem	CASLock	MCAS	SAS	Gen-Anti-SAT		CAC	DTL		
										Comp.	Non-Comp.		SARLock	Anti-SAT	CAC
b14_C	414	435	473	800	693	450	451	439	470	444	454	466	449	470	497
b15_C	498	510	521	1,243	868	582	533	547	596	528	532	590	513	626	617
b20_C	598	595	592	1,485	1,494	702	598	612	611	598	622	618	600	611	629
b21_C	628	615	628	1,592	1,452	643	665	628	620	632	612	662	637	632	648
b22_C	650	685	621	1,940	1,706	667	693	696	961	680	679	690	678	685	714
b17_C	782	795	745	2,699	2,691	740	790	813	827	768	767	720	795	1,014	740
b18_C	1,003	994	1,032	3,598	3,502	990	984	903	1,771	805	915	910	1,023	1,010	1,022
b19_C	1,923	1,948	1,972	4,616	4,590	1,990	1,905	1,920	2,423	1,969	1,992	1,945	1,959	1,973	2,040

TABLE VIII
AVERAGE EXECUTION TIME (IN SECONDS) FOR THE **CIRCUIT-RECOVERY ATTACK** FOR ITC-99 BENCHMARKS. THE LOCKED DESIGNS ARE SYNTHESIZED USING ALL THE AVAILABLE GATES FROM GLOBALFOUNDRIES 65NM TECHNOLOGY LIBRARY

Circuit \ Defense	SARLock	Anti-SAT	TTLock	SFLL-HD	SFLL-flex	SFLL-rem	CASLock	MCAS	SAS	Gen-Anti-SAT		CAC	DTL		
										Comp.	Non-Comp.		SARLock	Anti-SAT	CAC
b14_C	799	805	819	1,995	1,800	1,049	825	798	825	788	810	890	870	929	898
b15_C	975	1,010	1,003	2,470	2,198	1,189	980	1,020	902	1,020	1,023	1,011	1,016	1,026	1,037
b20_C	978	920	1,390	3,376	2,516	1,124	1,019	1,014	978	1,060	1,025	1,010	980	997	1,029
b21_C	910	1,019	1,016	3,329	2,507	1,121	991	1,108	902	1,027	1,007	1,019	1,098	1,018	1,028
b22_C	1,208	1,229	1,219	3,700	3,149	1,200	1,134	1,207	1,115	1,217	1,197	1,159	1,280	1,250	1,210
b17_C	1,304	1,490	1,498	4,392	4,410	1,379	1,488	1,506	1,503	1,420	1,410	1,404	1,443	1,495	1,490
b18_C	2,095	2,150	2,194	5,198	4,704	2,433	2,498	2,490	4,760	2,398	2,425	2,439	2,499	2,401	2,495
b19_C	5,920	5,939	5,940	8,067	8,197	5,409	5,327	5,989	6,890	5,320	5,319	5,173	5,980	5,996	5,895

C. Important Findings of Valkyrie Security Diagnostic Tool

1) The general construction of all DFLT techniques (see Fig. 7 (a)) mandates the presence of two critical signals ($cs1$ and $cs2$) in the locked design. Our experimental analysis reveals the presence of a new critical signal, cs, resulting from an XOR operation between $cs1$ and $cs2$. In a nutshell, synthesis-induced optimizations transform the DFLT techniques to SFLT techniques, as shown in Fig. 7 (b). *Valkyrie* flags this new critical signal cs as a vulnerability, informing the designer to discard this vulnerable defense. We illustrate the frequency of the conversion from DFLT to SFLT as stacked bar graphs in Fig. 9 for ITC-99 benchmarks for 20 trials each, for six PSLL techniques synthesized using the full library. A majority number of trials in SFLL-flex and CAC results in SFLT techniques compared to MCAS.
2) The intended implementation for DFLT techniques consists of a modified (functionality stripped) design and a key-controlled restore unit (Fig. 8 (a)). The modified design is obtained by inserting point-functions (TTLock, CAC) and diversifying point-functions (replacing some gates in AND-tree with OR/NAND gates, which include CAC-DTL). *However, Valkyrie diagnostic tool conjectures that all the techniques mentioned above include*

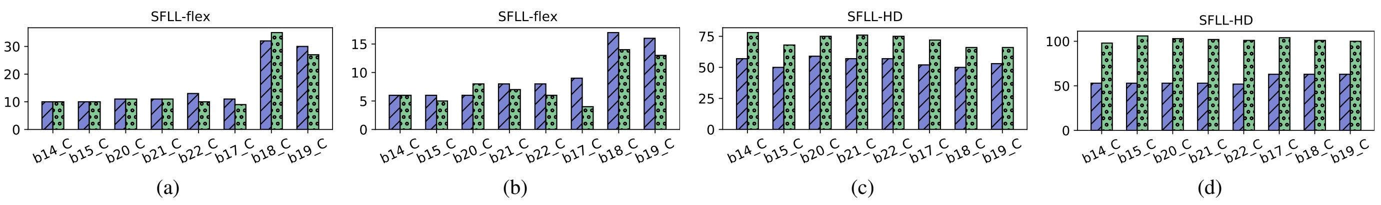


Fig. 10. Number of candidate critical signals ($CCS1$, $CCS2$) for ITC-99 benchmarks. All the locked designs have been synthesized with Nangate 45nm library. (a) SFLL-flex locked designs synthesized with 2-input gates. (b) SFLL-flex locked designs synthesized with full-library. (c) SFLL-HD locked designs synthesized with 2-input gates. (d) SFLL-HD locked designs synthesized with full-library. The blue color (forward slash) denotes the number of signals in $CCS1$ and the green color (circles) denotes the number of signals in $CCS2$.

some point-function-related circuitry that appears standalone in the locked design (Fig. 8 (b)). We note that *reduced-order point-functions*[15] are present for all the aforementioned techniques. For instance, consider TT-Lock and CAC, where a point-function is added to the original design; this point-function appears standalone for most cases (even post-re-synthesis). However, there are a handful of cases where we observed a reduction in the point-function structure (rather than a perturb unit comprising of 64 primary inputs, we observe a perturb unit of 60–61 primary inputs (Fig. 8 (c)). Such special cases are addressed by Property 6 (see Sec. III-A), where the critical signal $cs2$ is identified by being connected to *either all or a subset* of only protected input ports.

3) Recall that SFLL-rem is architected using principles of logic removal (for stripping functionality) as opposed to other DFLT techniques [28]. However, our analysis reveals the existence of a reduced-order point-function which highlights the deficiency of existing CAD tools.
4) Finally, DTL [36] and Gen-Anti-SAT [37] were proposed specifically to thwart the SPS and other removal attacks by skewing the signal probabilities of the critical signal toward 0.5. These techniques were thus thought (and proclaimed) to be devoid of structural vulnerabilities. However, *Valkyrie identifies structural vulnerabilities in these techniques.* This finding is especially important and signifies the importance of the proposed tool in enabling researchers to test the hardware implementation of their PSLL techniques for structural vulnerabilities.

D. Results for the Circuit-Recovery (CR) Attack

In this section, we present our results for the CR attack on 15 (eight unbroken) PSLL techniques. Note that the time taken by the attack is higher than the security diagnostic tool due to the querying of the oracle (Table VII and Table VIII). *We observe 100% success in breaking all the considered PSLL techniques.* We combined the properties identified from the considered PSLL techniques in our CR attack, making it generic and applicable in handling all considered techniques. We further test the generality of our CR attack against different synthesis tools, synthesis commands, technology libraries, and abstraction levels, similar to how we tested the generality of the diagnostic tool as discussed in Sec. V-B.

[15]Refers to a point-function that is connected to a subset of the PIPs.

Takeaway Message: Our CR attack can always recover the original design in all the considered PSLL techniques irrespective of the synthesis tool, synthesis commands, technology libraries, or abstraction levels. Our proposed attack is *agnostic* to the choice of the synthesis tool.

Number of candidate critical signals: Figure 10 illustrates the maximum number of candidate critical signals for ITC-99 benchmarks locked using SFLL-flex and SFLL-HD, synthesized using two-input gates and full-library, respectively. We observe at most 106 candidate critical signals for SFLL-HD, the highest amongst all considered PSLL techniques. Candidate critical signals directly correlate with the runtime of the CR attack, as highlighted in Table VII and Table VIII.

Critical signal values: As explained in Sec. III, synthesis-induced optimizations facilitate netlist restructuring, bubble pushing, etc., which sometimes invert the values (csv) of the critical signals. Recall that, while the identification of the critical signals (cs) is possible using the reverse-engineered locked netlist (white-box structural analysis), assigning correct values to those critical signals (which facilitates the recovery of the original design from the locked design) is a non-trivial task, and requires the usage of an oracle. We show that on average, the percentage of critical signal value(s) as logic 0 and as logic 1 in Figs. 11 and 12 for ITC-99 benchmarks when synthesized using all available gates (full-library) for the Nangate 45nm library. We observe that across all the considered benchmarks, the inversion for the critical signal value occurs 58.13% for SARLock, 42.5% for Anti-SAT, 54.38% for CASLock, 19.38% for Gen-Anti-SAT (non-complementary), 46.25% for SARLock-DTL, and 30.63% for Anti-SAT-DTL (see Fig. 11). We observe a similar trend for DFLT, where both the values of the critical signals $cs1$ and $cs2$ invert due to synthesis-induced optimizations (see Fig. 12). This analysis highlights the importance of ascertaining the correct critical signal values to launch a successful CR attack for PSLL techniques.

E. Comparison with State-of-the-Art Circuit-Recovery Attacks

The research community has proposed many CR attacks to challenge the security guarantees of the PSLL techniques. However, almost all of these attacks are locking-technique specific, as highlighted in Table II. *Our generalized CR attack is successful in retrieving the original design from the locked design for all the considered PSLL techniques* in contrast to

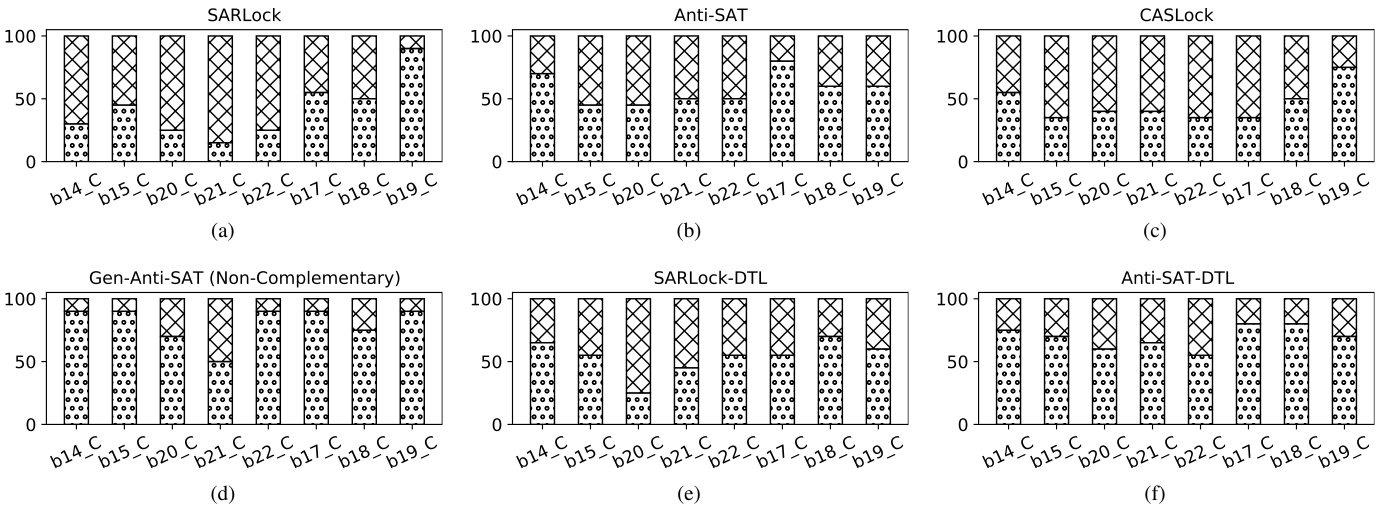


Fig. 11. Distribution of critical signal values (csv) for SFLT into logic 0 (circles) and logic 1 (mesh) for ITC-99 benchmarks.

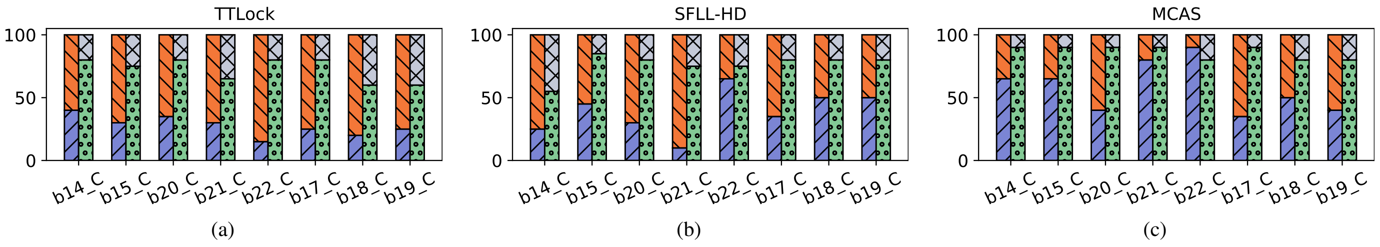


Fig. 12. Distribution of the pair of critical signals ($csv1$ and $csv2$) for DFLT. The blue color (forward slash) denotes logic 0 and orange (back slash) denotes logic 1, for the critical signal $cs1$. Similarly, the green color (circles) denotes logic 0 and the gray color (mesh) denotes logic 1, for the critical signal $cs2$.

the prior attacks such as the Bypass [24], SPS [23], GNNUnlock [33]. The SPS and bypass attacks are locking-technique specific (SARLock and Anti-SAT). Furthermore, they do not account for the recently developed techniques. GNNUnlock learns the structural features of the PSLL techniques like Anti-SAT, TTLock, and SFLL-HD, where point-functions are added explicitly. However, GNNUnlock identifies the critical signals (cs) but not their corresponding logical values (csv). Finally, the attacks proposed in [32] work on CASLock, MCAS, and Anti-SAT but may not apply to other PSLL techniques.

Takeaway Message: The proposed CR attack is generic, applicable to a plethora of techniques, and can be extended to future PSLL techniques. Unlike the prior attacks in the literature, our attack is not locking-technique specific. Furthermore, the attack identifies the value of the critical signal(s), which is imperative for the complete functional recovery of the original design from the locked design.

VI. Discussion

A. Ramifications of Not Using Valkyrie Diagnostic Tool

Recall that the design IP is the designer's secret due to extensive research and development (R&D) ranging in billions of US dollars [43]. Consequently, a security-enforcing designer uses a PSLL technique to protect the design IP. Using the CR attack, the attacker can identify, isolate, and remove the locking circuitry, thereby obtaining the original design IP. However, access to the design IP leads to an economic loss to the designer, as explained next. For instance, consider companies A and B, where A invests in R&D to develop a state-of-the-art design IP. B is a direct competitor of A and colludes with untrusted entities in the supply chain to steal the design IP for advancing their state-of-the-art *without* investing in R&D. Thus, company A has now lost its competitive advantage in the market and targeted consumers. Hence, it becomes imperative that the designers systematically analyze the hardware implementation of the PSLL technique for vulnerabilities before sending the design for fabrication.

B. Towards a Security Sign-off Tool for PSLL Techniques

Recall that our primary goal was to formulate a security diagnostic tool to identify structural vulnerabilities in the hardware implementation of PSLL techniques. Our results demonstrate that the considered PSLL techniques have structural vulnerabilities (existence of critical signal(s)) when implemented using CAD tools.[16] This gives rise to two pertinent questions: (i) why do these structural vulnerabilities exist?

[16]The hardware implementation of a PSLL technique can be architected through security-aware synthesis to dissolve the critical signal(s). In such a scenario, *Valkyrie* informs the designer that no such critical signal(s) could be found. However, developing security-aware synthesis tools that leave no structural traces remains an open problem for the research community.

(ii) what can a designer do after identifying these vulnerabilities? Addressing (i), we state that the vulnerabilities in the considered PSLL techniques arise due to the reliance on (security-agnostic) CAD tools. Designers use CAD tools to blend the protection logic with the original design to ensure structural security. However, these tools are not designed for security but for reducing power, performance, and area (PPA). This non-cognizance to security leaves structural traces/hints, which *Valkyrie* identifies. Addressing (ii), the designer can discard the locked design at hand and move to another PSLL technique. Nevertheless, suppose the architecture of the new PSLL technique is similar to the 15 PSLL techniques considered in this work. In that case, *Valkyrie* will be able to find the vulnerability in the new PSLL technique. Moreover, designers can also adopt a non-PSLL technique as suggested in Sec. VI-C. To that end, as a part of future work, we aim to extend the security diagnostic tool to offer "guidance" in terms of locking alternatives to the security-enforcing designer, thereby converting the current diagnostic tool into a security sign-off tool for PSLL techniques.

C. Other Logic Locking Solutions

As mentioned in Sec. I, researchers have integrated SAT-hard structures for increasing the complexity of the SAT-based attack [10]. Recently, scan locking techniques have been developed to thwart SAT-based attacks. A complete scan access is required by the attack to access the internal nodes of the design-under-attack. Obfuscating the scan chains ensures obfuscation of the test stimuli and the corresponding test responses [44]. However, linear obfuscation of scan stimuli and responses makes it vulnerable to modeling attacks [45]. Another approach to thwart existing structural attacks could be to leverage universal circuits to further obfuscate the locked circuit and resolve the dilemma between locking robustness and structural security [46]. Further, a new approach was recently presented, where the secret key is cleared from the key-registers when scan access is detected [47]. This way, the oracle is now non-functional and a logic locking technique that could generally be broken with a functional oracle (e.g., a non-PSLL technique) can now be employed.

VII. Conclusion

The accelerated pace of the cat-and-mouse game between attackers and defenders in provably-secure logic locking (PSLL), with all the recent defense techniques relatively short-lived, points to the lack of a security diagnostic tool.

In this work, we extract generic properties of PSLL techniques to build a security diagnostic tool (*Valkyrie*) that a security-enforcing designer can use to assess the structural vulnerabilities of PSLL techniques before taking the design to silicon. Researchers can use our tool to test the resilience of a new PSLL technique before deployment. *Such a security diagnostic tool for PSLL techniques is the first-of-its-kind.*

The proposed *Valkyrie* security diagnostic tool searches for the critical signals in a locked design using fault insertion and equivalence checking techniques. We also showcase that any vulnerability identified by the diagnostic tool has a corresponding attack vector that attackers can exploit.

To that end, we propose a circuit-recovery attack that advances the state-of-the-art by being generic as opposed to other attacks which have been predominantly locking-technique specific. Furthermore, we showcase that these attacks are equivalent in strength to the security diagnostic tool, assuring that if *Valkyrie* identifies a vulnerability, it can always be exploited by the attackers. The proposed attack retrieves the original functionality from the locked design, thus acting as a cautionary tale to the designer.

We validate our claims on 15 PSLL techniques (**including eight unbroken**) for different synthesis tools, technology libraries, and abstraction levels across a dataset of more than $20,000$ designs. *Our security diagnostic tool is always successful in identifying structural vulnerabilities in locked designs.* Further, our circuit-recovery attack successfully validates the results from *Valkyrie* by circumventing the security guarantees of all the considered PSLL techniques. We envision that our tool serves as a vehicle to test the resilience of a newly developed PSLL technique. Finally, our experimental analysis highlights the security-obliviousness of the current CAD tools, especially for PSLL techniques, and calls for either developing security-centric CAD tools or adopting other logic locking solutions.

Acknowledgment

The authors would like to acknowledge Nikhil Limaye for suggesting the acronym "*Valkyrie*."

References

[1] M. Hachman, "Chip shortages will continue until 2023," https://www.pcworld.idg.com.au/article/687673/chip-shortages-will-continue-until-2023-superfoundry-tsmc-says/, 2021, [Online; accessed 14-July-2021].

[2] R. Zafar, "TSMC's Total 3nm Investment Will Equal At Least $ 23 Billion," https://wccftech.com/tsmc-3nm-investment-23-billion-project-end/, 2021, [Online; accessed 14-July-2021].

[3] "Intel: Sorry, But Our 7nm Chips Will Be Delayed to 2022, 2023," https://www.pcmag.com/news/intel-sorry-but-our-7nm-chips-will-be-delayed-to-2022-2023/, 2020, [Online; accessed 22-July-2021].

[4] "Intel to Decide on Tapping Third-Party Foundry for 7nm Chips By Early 2021," https://www.pcmag.com/news/intel-to-decide-on-tapping-third-party-foundry-for-7nm-chips-by-early-2021, 2021, [Online; accessed 22-July-2021].

[5] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proc. IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[6] J. P. Skudlarek, T. Katsioulas, and M. Chen, "A platform solution for secure supply-chain and chip life-cycle management," *Computer*, vol. 49, no. 8, pp. 28–34, 2016.

[7] D. P. Affairs, "DARPA Selects Teams to Increase Security of Semiconductor Supply Chain," accessed July 28th, 2021. [Online]. Available: https://www.darpa.mil/news-events/2020-05-27

[8] A. Chakraborty, N. G. Jayasankaran, Y. Liu, J. Rajendran, O. Sinanoglu, A. Srivastava *et al.*, "Keynote: A disquisition on logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 1952–1972, 2019.

[9] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending piracy of integrated circuits," *Computer*, vol. 43, no. 10, pp. 30–38, 2010.

[10] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *IEEE International Symposium on Hardware Oriented Security and Trust*, 2015, pp. 137–143.

[11] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," in *IEEE International Symposium on Hardware Oriented Security and Trust*, 2017, pp. 95–100.

[12] Y. Shen and H. Zhou, "Double DIP: Re-evaluating security of logic encryption algorithms," in *Proceedings of the Great Lakes Symposium on VLSI*, 2017, pp. 179–184.
[13] Y. Xie and A. Srivastava, "Mitigating SAT attack on logic locking," in *Proc. Cryptogr. Hardw. Embed. Sys.*, 2016.
[14] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," in *IEEE International Symposium on Hardware Oriented Security and Trust*, 2016, pp. 236–241.
[15] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, "Cross-lock: Dense layout-level interconnect locking using cross-bar architectures," in *Proceedings of the Great Lakes Symposium on VLSI*, 2018, pp. 147–152.
[16] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Full-lock: Hard distributions of SAT instances for obfuscating circuits using fully configurable logic and routing blocks," in *Proceedings of the 56th Design Automation Conference*, 2019, pp. 1–6.
[17] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "Cyclic obfuscation for creating SAT-unresolvable circuits," in *Proceedings of the on Great Lakes Symposium on VLSI*, 2017, pp. 173–178.
[18] H. Zhou, R. Jiang, and S. Kong, "CycSAT: SAT-based attack on cyclic logic encryptions," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 49–56.
[19] K. Shamsi, D. Z. Pan, and Y. Jin, "IcySAT: Improved SAT-based attacks on cyclic locked circuits," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–7.
[20] Y. Shen, Y. Li, A. Rezaei, S. Kong, D. Dlott, and H. Zhou, "BeSAT: Behavioral SAT-based attack on cyclic logic encryption," in *24th Asia and South Pacific Design Automation Conference*, 2019, pp. 657–662.
[21] J. Sweeney, M. J. Heule, and L. Pileggi, "Modeling techniques for logic locking," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.
[22] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "NNgSAT: Neural network guided SAT attack on logic locked complex structures," in *IEEE/ACM International Conference On Computer Aided Design*, 2020, pp. 1–9.
[23] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal attacks on logic locking and camouflaging techniques," *IEEE Transactions on Emerging Topics in Computing*, vol. 8, no. 2, pp. 517–532, 2017.
[24] X. Xu, B. Shakya, M. M. Tehranipoor, and D. Forte, "Novel bypass attack and BDD-based tradeoff analysis against all known logic locking attacks," in *International conference on cryptographic hardware and embedded systems*. Springer, 2017, pp. 189–210.
[25] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1601–1618.
[26] D. Sirone and P. Subramanyan, "Functional analysis attacks on logic locking," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2514–2527, 2020.
[27] F. Yang, M. Tang, and O. Sinanoglu, "Stripped functionality logic locking with hamming distance-based restore unit (SFLL-hd)–Unlocked," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 10, pp. 2778–2786, 2019.
[28] A. Sengupta, M. Nabeel, N. Limaye, M. Ashraf, and O. Sinanoglu, "Truly stripping functionality for logic locking: A fault-based perspective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4439–4452, 2020.
[29] "CSAW'19 Logic Locking Conquest," 2019. [Online]. Available: https://sites.google.com/nyu.edu/logiclocking19/results?authuser=0
[30] Z. Han, M. Yasin, and J. J. Rajendran, "Does logic locking work with EDA tools?" in *30th USENIX Security Symposium*, 2021.
[31] B. Shakya, X. Xu, M. Tehranipoor, and D. Forte, "CAS-Lock: A security-corruptibility trade-off resilient logic locking scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 175–202, 2020.
[32] A. Sengupta, N. Limaye, and O. Sinanoglu, "Breaking CAS-lock and its variants by exploiting structural traces," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, no. 3, pp. 418–440, 2021.
[33] L. Alrahis, S. Patnaik, F. Khalid, M. A. Hanif, H. Saleh, M. Shafique *et al.*, "GNNUnlock: Graph Neural Networks-based Oracle-less Unlocking scheme for Provably Secure Logic Locking," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 780–785.
[34] F. Guo, W. Susilo, and Y. Mu, *Introduction to Security Reduction*. Springer, 2018.
[35] J. Buchmann, *Introduction to Cryptography*. Springer, 2004, vol. 335.
[36] K. Shamsi, T. Meade, M. Li, D. Z. Pan, and Y. Jin, "On the Approximation Resiliency of Logic Locking and IC Camouflaging Schemes," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 2, pp. 347–359, 2018.
[37] J. Zhou and X. Zhang, "Generalized SAT-Attack-Resistant Logic Locking," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2581–2592, 2021.
[38] Y. Liu, M. Zuzak, Y. Xie, A. Chakraborty, and A. Srivastava, "Strong Anti-SAT: Secure and effective logic locking," in *21st International Symposium on Quality Electronic Design (ISQED)*, 2020, pp. 199–205.
[39] K. Shamsi, "Attack tool and benchmarks," 2020. [Online]. Available: https://bitbucket.org/kavehshm/neos/src/master/
[40] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial-strength Verification Tool," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 24–40.
[41] N. EEN, "MiniSat : A SAT solver with conflict-clause minimization," *Proc. SAT-05 : 8th Int. Conf. on Theory and Applications of Satisfiability Testing*, pp. 502–518, 2005. [Online]. Available: https://ci.nii.ac.jp/naid/10026031126/en/
[42] (2011) NanGate FreePDK45 Open Cell Library. Nangate Inc. [Online]. Available: http://www.nangate.com/?page_id=2325
[43] "Semiconductor R&D Spending: Intel accounted for 36%," https://anysilicon.com/semiconductor-rd-spending-intel-accounted-36/, 2016, [Online; accessed 23-July-2021].
[44] X. Wang, D. Zhang, M. He, D. Su, and M. Tehranipoor, "Secure scan and test using obfuscation throughout supply chain," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 9, pp. 1867–1880, 2017.
[45] L. Alrahis, M. Yasin, N. Limaye, H. Saleh, B. Mohammad, M. Alqutayri *et al.*, "ScanSAT: Unlocking static and dynamic scan obfuscation," *IEEE Transactions on Emerging Topics in Computing*, 2019.
[46] H. Zhou, A. Rezaei, and Y. Shen, "Resolving the trilemma in logic encryption," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.
[47] N. Limaye, E. Kalligeros, N. Karousos, I. G. Karybali, and O. Sinanoglu, "Thwarting All Logic Locking Attacks: Dishonest Oracle With Truly Random Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 9, pp. 1740–1753, 2021.

Nimisha Limaye is a Ph.D. candidate at the Department of Electrical and Computer Engineering at New York University, USA and also a Global Ph.D. Fellow with New York University Abu Dhabi, UAE. Her research interests include hardware security and in particular logic locking, scan locking, and hardware Trojans.

Satwik Patnaik received his Ph.D. degree in Electrical engineering from Tandon School of Engineering, New York University, Brooklyn, NY, USA in September 2020. He is currently a Postdoctoral researcher with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX, USA. His research delves into IP protection techniques, CAD frameworks for incorporating security, leveraging 3D paradigm for security, exploiting security properties of emerging devices, and applied machine learning for hardware security.

Ozgur Sinanoglu is a professor of electrical and computer engineering at New York University Abu Dhabi. He obtained his Ph.D. in Computer Science and Engineering from University of California San Diego. Prof. Sinanoglu's research interests include design-for-test, design-for-security and design-for-trust for VLSI circuits, where he has more than 200 conference and journal papers, and 20 issued and pending US Patents. His recent research in hardware security and trust is being funded by US National Science Foundation, US Department of Defense, Semiconductor Research Corporation, Intel Corp, and Mubadala Technology.