Advances in Logic Locking: Past, Present, and Prospects

Hadi Mardani Kamali, Kimia Zamiri Azar, Farimah Farahmandi, and Mark Tehranipoor

Abstract—Logic locking is a design concealment mechanism for protecting the IPs integrated into modern System-on-Chip (SoC) architectures from a wide range of hardware security threats at the IC manufacturing supply chain. Logic locking primarily helps the designer to protect the IPs against reverse engineering, IP piracy, overproduction, and unauthorized activation. For more than a decade, the research studies that carried out on this paradigm has been immense, in which the applicability, feasibility, and efficacy of the logic locking have been investigated, including metrics to assess the efficacy, impact of locking in different levels of abstraction, threat model definition, resiliency against physical attacks, tampering, and the application of machine learning. However, the security and strength of existing logic locking techniques have been constantly questioned by sophisticated logical and physical attacks that evolve in sophistication at the same rate as logic locking countermeasure approaches. By providing a comprehensive definition regarding the metrics, assumptions, and principles of logic locking, in this survey paper, we categorize the existing defenses and attacks to capture the most benefit from the logic locking techniques for IP protection, and illuminating the need for and giving direction to future research studies in this topic. This survey paper serves as a guide to quickly navigate and identify the state-of-the-art that should be considered and investigated for further studies on logic locking techniques, helping IP vendors, SoC designers, and researchers to be informed of the principles, fundamentals, and properties of logic locking.

Index Terms—IP Piracy, IC Supply Chain, IP Protection, Logic Locking.

1 Introduction

Modern integrated circuit (IC) supply chain has evolved dramatically in the last two decades for multiple reasons, such as ever-increasing cost/complexity of IC manufacturing, huge recurring cost of IC fab maintenance and troubleshooting, aggressive time-to-market, the expedition of the IC supply chain flow, involvement of multiple third-party Intellectual Property (IP) vendors, engagement of cutting-edge technologies, being a primary forerunner in the semiconductor market, etc. [1]. Over time, the economy of scale has pushed for ever-increasing adoption of the IC supply chain's horizontal model. In the IC supply chain's horizontal model, separate entities fulfill various stages of design, fabrication, testing, packaging, and integration of ICs, forming a globally distributed chain.

Plunged in the globalization ocean of the IC supply chain, given the complexity of originally implementing major components of a chip, the design team will bring and acquire multiple 3^{rd} party IPs from numerous IP owners to reduce time-to-market. Additionally, given the overall cost of fabrication, wafer sort, dicing, packaging, package test, and getting access to state-of-the-art technologies, performing most, if not all, these steps in the offshore facility may be the preferred approach for design houses. Outsourcing and the involvement of numerous stakeholders in various stages of the supply chain dramatically reduce the cost and time-to-market of the chip.

Retaining of being competitive is even getting worse especially in the current post-pandemic market, where chip demand is exceeding foundry fab capacity, causing tremendous shortages in the market, which even results in increasing the share prices of major contract chipmakers, including TSMC, UMC, and SMIC [2]. In such a market with this unprecedented demand, we will face a more panic IC design, implementation, manufacturing, and testing by original equipment manufacturers (OEM), which is done precariously to steal the market/contracts. So, with fewer precautions taken by the OEMs to meet the market demand, and by getting more globalization, OEMs and/or IP vendors face a drastic reduction of the control and monitoring capability over the supply chain.

Despite the benefit achieved by the globalization of the IC supply chain, after the involvement of multiple entities within the IC supply chain with no reciprocal trust and lack of reliable monitoring, the control of original manufacturers and IP owners/vendors over the supply chain will be reduced drastically, resulting in numerous hardware security threats, including but not limited to IP piracy, IC overproduction, and counterfeiting [3], [4].

To address threats associated with the horizontal IC supply chains, a variety of design-for-trust countermeasures have been investigated in the literature. Watermarking, IC metering, IC camouflaging, and hardware obfuscation are examples of passive to active design-for-trust countermeasures [5], [6], [7], [8]. In comparison to other Design-for-trust countermeasures, logic locking as a proactive technique for IP protection has garnered remarkable attention in recent years, culminating in a plethora of research over the last two decades on designing a variety of robust solutions at different levels of abstraction.

Logic locking enables the IP/IC designers to provide limited post-fabrication programmability to the fabricated designs, thereby concealing the underlying functionality

- *K.Z. Azar, H.M. Kamali, F. Farahmandi, M. Tehranipoor are with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, 32611.*
E-mail Addresses: h.mardanikamali@ufl.edu, k.zamiriazar@ufl.edu, farimah@ece.ufl.edu, tehranipoor@ufl.edu.

behind an ocean of options. The resultant locked circuit functionality is governed by the logic locking secret, known as the key, which is known only to authorized/trusted entities, such as IP owners or original component manufacturers (OCM), and by loading the correct key value, the design house can unlock the circuit. The concept of locking a circuit against malicious or unauthorized activities was first introduced in *Lock and Key* technique [8], [9], [10]. In this technique, a key-based mechanism has been introduced that add programmability (randomness) to the design-for-testability (DFT) sub-circuits (subchains) when being accessed by an unauthorized user. Over time, this programmability has been extended to other parts of the design, like combinational parts [11], [12], sequential (non-DFT) parts [13], [14], and even parametric/behavioral (non-Boolean) aspects of the design [15]. With high applicability and adaptability of logic locking, a wide range of logic locking techniques currently being explored in both the academic context and semiconductor industries, such as Mentor Graphic's TrustChain platform enabled by logic locking and the newest Defense Advanced Research Projects Agency (DARPA) project on Automatic Implementation of Secure Silicon (AISS) [16], [17].

Logic locking countermeasures appeared as a promising protection mechanism against IP piracy and IC overproduction. However, the development of attacks on logic locking by white hatter researchers, known as logic de-obfuscation attacks, is also essential for the evolution of the logic locking paradigm. This helps the researchers to distinguish between weak and robust countermeasures, highlighting the weaknesses of existing countermeasures and illuminating the need for and giving direction to future research in this area. Hence, for more than a decade, numerous studies investigated logic locking in both defense and attack perspectives. It results in the evolution of both the attacks and countermeasures to become increasingly sophisticated. Furthermore, over time, the emergence of cutting-edge technologies, e.g. failure analysis (FA) equipment [18], [19], [20], application of state-of-the-art approaches like the usage of machine learning [21], [22], and deeper infiltration by the adversaries even into trusted facilities [23], show that the logic locking countermeasures are not yet as mature as what promised from the theoretical point of view.

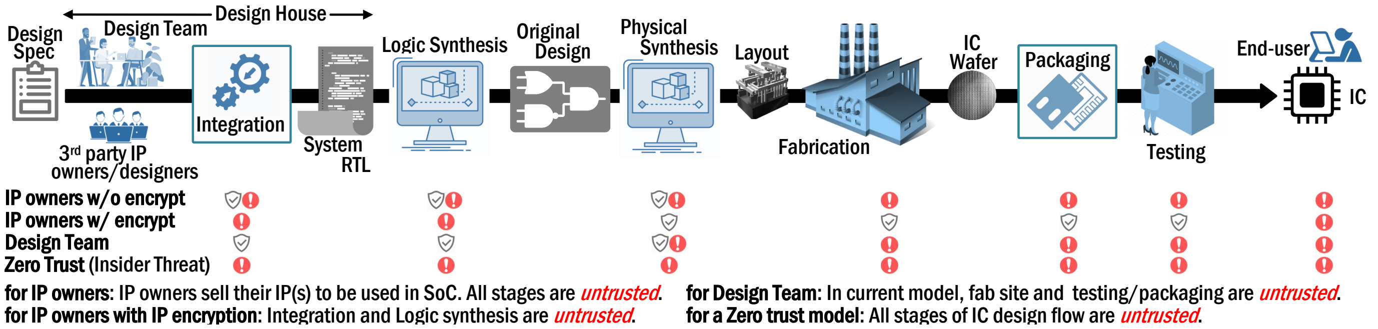
In this survey paper, we first holistically review the direction of logic locking through the last decade, in both the attack and defense sides. We will define the parameters, characteristics, and assumptions, either directly or indirectly affect the outcome of logic locking or the de-obfuscation attacks. Then, we will categorize all defenses and attacks and will evaluate each category based on all pre-defined characteristics and parameters, separately. By providing a very comprehensive comparison between different categories, in both the attack and defense sides, this paper helps to delineate the future of logic locking and new possible directions.

The paper is organized as follows. In Section 2, we first provide all background knowledge, including pivotal parameters, characterization, models, and assumptions on logic locking helping us to have a better understanding of the categorization provided through the paper. Then Section 3 provides the details of state-of-the-art logic locking techniques, different breeds, characteristics, etc. Section 4 will also cover all notable de-obfuscation attacks introduced so far on logic locking. Then in Section 5, based on all observations and details covered in this survey, we will discuss the possible futuristic trends in this domain, and finally, Section 6 concludes the paper.

2 Background: Parameters, Models, and Assumptions in Logic Locking

Fig. 1 demonstrates the main steps of modern IC design flow, in which by starting from the design specification, multiple parties will be involved horizontally in the process of IC manufacturing [24], and by outsourcing that can be engaged at different stages, the OEMs have the least reliable form of control to their belongings (the circuitry or the IP), which results in introducing these contracted and mainly offshore entities as the untrusted parties. As for malicious/untrusted entities, from each IC manufacturing stage, per Fig. 2, given any representation of the IC, including integrated design, physically-synthesized netlist, layout, packaged IC, or IC under test, the functionality can be reverse-engineered. The toughest yet possible form of circuit reconstruction is usually referred to as physical reverse-engineering, which can be done by malicious end-users. Successful reverse-engineering, in any form, allows the malicious (untrusted) entities to steal the IP/design and/or illegally over-produce or re-use it.

Given such circumstances, in which the IP vendors or design teams have to protect their belongings (IP and the design (integrated)) against any form of reverse-engineering demonstrated in Fig. 2, we are witnessing different broad categories of design-for-trust techniques, which mainly are



	Integration	Logic Synthesis	Physical Synthesis	Fabrication	Packaging	Testing	IC
IP owners w/o encrypt	✓❗	✓❗	✓❗	❗	❗	❗	❗
IP owners w/ encrypt	❗	❗	✓	✓	✓	✓	❗
Design Team	✓	✓	✓❗	❗	❗	❗	❗
Zero Trust (Insider Threat)	❗	❗	❗	❗	❗	❗	❗

for IP owners: IP owners sell their IP(s) to be used in SoC. All stages are *untrusted*.
for IP owners with IP encryption: Integration and Logic synthesis are *untrusted*.
for Design Team: In current model, fab site and testing/packaging are *untrusted*.
for a Zero trust model: All stages of IC design flow are *untrusted*.

Fig. 1: Conventional (Globalized) IC Design Flow and its Trustworthiness Issue.

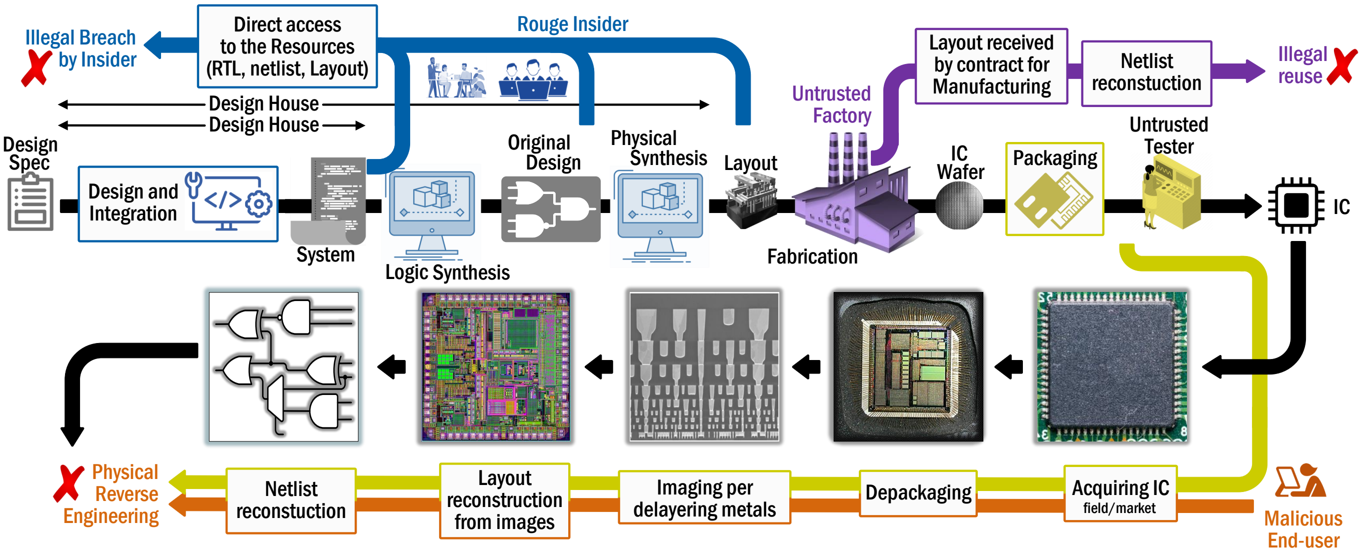
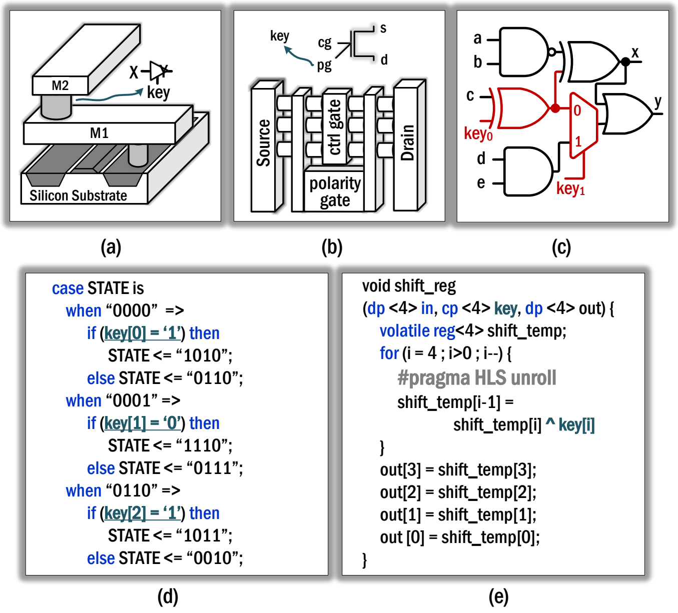


Fig. 2: Different Threat Models at Different Stages of IC Design Flow.

(i) IC camouflaging [7], [25], [26], [27], [28], (ii) split manufacturing [29], [30], [31], [32], and (iii) logic locking [10], [11], [12], [33], [34]. Considering the possibility of having reverse-engineering at different layers (as demonstrated in Fig. 2), both IC camouflaging and split manufacturing are applicable to a subset of these threats. For instance, IC camouflaging use high-structural similar logic gates (or other physical structures such as dummy vias) with different functions. However, it is only effective against post-manufacturing attempt(s) of reverse engineering (by the malicious end-user), and it provides no limitations against a foundry's attempt at reverse engineering, as a foundry has access to all masking layers and is not trapped by structural ambiguity for being able to logically extract a netlist. Split manufacturing, on the other hand, is the integration of the transistors and lower metal layers (a.k.a. Front End Of Line (FEOL) layers) fabricated in cutting-edge technology nodes by an untrusted (and mostly offshore) high-end foundry, with higher metal layers (a.k.a. Back End Of Line (BEOL) layers) fabricated at the design house's trusted low-end foundry. This countermeasure alleviates the security risks at the untrusted foundry. However, this methodology still cannot protect the design against malicious end-users and malicious insiders that threaten the confidentiality of the proprietary technology. On the contrary, logic locking -if implemented meticulously- can be a proactive hardware-for-trust technique that can protect against all previously mentioned threats through the IC supply chain.

2.1 Basic Definitions of Logic Locking

Logic locking[1] is the capability of adding post-fabrication programmability that could be added using some extra gates, known as key-programmable gates (key gates) that are driven from the secret of logic locking, i.e. the key. Logic locking techniques could be implemented at different levels of abstraction. Fig. 3 demonstrates a simple example of logic locking in different levels of abstraction. For instance, at layout-level as shown in Fig. 3(a), the metal-insulator-metal (MIM) structure, which connects two adjacent metal layers, has been engaged as key-based programmable unit for routing-based locking [35]. Table 1 shows general specification of logic locking at different layers of abstraction. In general, moving from layout-level to RTL- or HLS-level mitigates the implementation effort. However, at a lower level of abstraction, finding a logic locking countermeasure at lower overhead is easier to be achieved. Furthermore, moving to a higher level of abstraction (like RTL and HLS) can also provide some form of protection against a sub-



```
case STATE is
  when "0000" =>
    if (key[0] = '1') then
      STATE <= "1010";
    else STATE <= "0110";
  when "0001" =>
    if (key[1] = '0') then
      STATE <= "1110";
    else STATE <= "0111";
  when "0110" =>
    if (key[2] = '1') then
      STATE <= "1011";
    else STATE <= "0010";
```

```
void shift_reg
(dp <4> in, cp <4> key, dp <4> out) {
  volatile reg<4> shift_temp;
  for (i = 4 ; i>0 ; i--) {
    #pragma HLS unroll
    shift_temp[i-1] =
          shift_temp[i] ^ key[i]
  }
  out[3] = shift_temp[3];
  out[2] = shift_temp[2];
  out[1] = shift_temp[1];
  out [0] = shift_temp[0];
}
```

Fig. 3: Logic Locking Examples at Different Levels of Abstraction: (a) Layout-level Key-based Routing, (b) Transistor-level Key-based Basic Gates, (c) Key-based Logic/Routing Gate-level, (d) RTL-level Key-based FSM, (e) HLS-level Key-based Shift Register.

[1] Although the term *logic locking* indicates a specific (gate-level) variation of *hardware obfuscation*, both terms have been widely used interchangeably in the literature. In this paper, we also used *logic locking* as the synonymous word for *hardware obfuscation*.

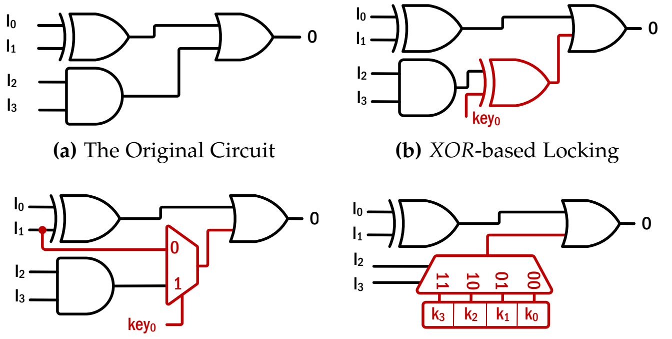
TABLE 1: Logic Locking Specification at Different Abstraction Layers.

Circuit	Granularity	Overhead	Implementation Effort
Layout-level	bitwise, wiring	Near to zero	High
Transistor-level	bitwise, switching, wiring	low	High
Gate-level	bitwise, logical	variant	Medium
RTL-level	bitwise, operational, behavioral	mid-high	low
High-level (HLS)	bitwise, operational, behavioral	mid-high	low

set of insider threats. Although at the moment, more than 90% of existing logic locking techniques are introduced and implemented at the gate-level, mostly done as a post-synthesis stage on the synthesized gate-level netlist in the supply chain, we will demonstrate RTL/HLS-based logic locking is one of the pivotal and widely-used current trends in logic locking.

As *bitwise* form of logic locking is achievable at any abstraction layer, based on the type of key gates, either used for logic locking or obtained after synthesis/compilation, we also can categorize logic locking into three main groups: (1) *XOR*-based, (2) *MUX*-based, and (3) *LUT*-based. As their names imply, they are using **eXclusive-OR**[2], **MUltipleXer**s, **Look-Up-Table**s for obfuscation, respectively. Fig. 4 depicts a simple example of each of these models, demonstrated at gate-level abstraction. It is worth mentioning that except layout-level and transistor-level locking that can be applied for concealment of wiring or switching, for both RTL-level and HLS-level that can target operational or behavioral, the locking part in the resultant netlist generated after synthesis has been implemented using one (or a combination) of these three key-gates. During the last decade, different logic locking techniques commonly have engaged these gates with different structures/functions for locking purposes. Based on some properties of these key gates, such as location, structure, count, intercorrelation, etc., the countermeasures provide various levels of robustness against the existing de-obfuscation attacks.

A crucial property of logic locking techniques is the **output corruptibility**. Output corruptibility is a very efficient measure of hiding the design's functionality while the logic locking is in place. Corruptibility means that when an incorrect key is applied to the locked circuit, (1) for how many output pins, and more importantly (2) for how many of the input patterns, the primary output (PO) will be corrupted once the key is incorrect. Usually, the output corruption is measured in terms of the hamming distance (HD) between the correct and the wrong output. Ideally, a 50% hamming distance is considered the highest deviation. Based on the location, structure, count, intercorrelation, etc., of the key-based *XOR*s/*MUX*s/*LUT*s that are engaged for locking purposes, the corruptibility will change. Corruptibility directly affects the resiliency of the countermeasures against the existing attacks. For instance, if the corruptibility is low, it allows the adversary to look for a specific way for only those POs affected or those specific input patterns that produce output corruption. For a well-designed logic locking countermeasure, the corruptibility must be high to avoid such vulnerabilities.



(a) The Original Circuit **(b)** *XOR*-based Locking **(c)** *MUX*-based Locking **(d)** *LUT*-based Locking

Fig. 4: Basic Gates used for Logic Locking, (a) Original, (b) correct key ($k_0 = 0$), (c) correct key ($k_0 = 1$), (d) correct key ($k_{0:3} = 0001$).

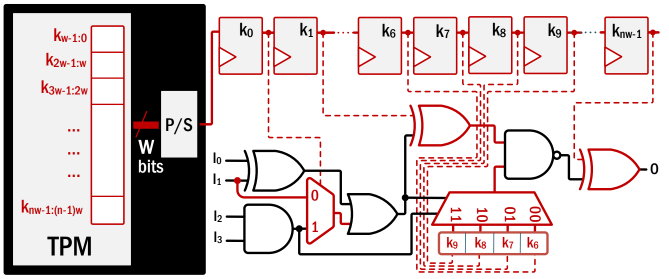


Fig. 5: Logic Locking Key Initialization from TPM.

As demonstrated in Fig. 4, the secret of logic locking, referred to as the *key*, must be provided to recover the correct functionality of the locked circuit. The **key initialization** of logic locking must be accomplished at a trusted facility and will be stored in TPM after the fabrication. Hence, the key management infrastructure around logic locking determines how the key will be initiated [36], [37], [38]. At power UP of a locked IC, as a part of the boot process, the content of TPM must be read and loaded into temporary registers connected to the locked circuit[3]. Fig. 5 shows a simple example of key initialization structure when logic locking is in place. Building the competent and secure infrastructure for key initiation[4] has been studied tremendously in the literature [36], [37], [38], [39], [40], [41], [42], whose details can be found in [43]. This part of the design consists of (1) TPM that consists of the logic locking key, (2) TPM wrapper which serializes the logic locking key via parallel-in to serial-out (P/S) module, and (3) temporary registers that stores the logic locking key while the IC is power ON, and (4) all infrastructure or customized cell required for securing the logic around key registers.

[2] For simplicity, we describe it as XOR-based. It can be trivially extended to be XNOR-based as well.

[3] This is when the logic locking key size is large enough (In the range of 1k-2k key bits). For smaller key sizes, range of <200, although not recommended for incurred routing and overhead challenges, the key could be simultaneously loaded from the TPM with no shift process, but still needs to be registered for testability purposes.

[4] In these infrastructures, regardless of the logic locking technique, different factors must be considered, e.g., the mechanism of key loading, integrating of key registers with DFT structure, removing the key leakage possibility through the scan chain, retaining the test coverage at high rates, etc.

2.2 Models/Assumptions in Attacks on Logic Locking

Based on the threat models and assumptions evaluated in the de-obfuscation attacks on logic locking, the attacks could be categorized into different sub-groups. From a malicious end-user point-of-view, some of the attacks require access to one *additional* activated version of the fabricated circuit (at-least two in total: one for reverse-engineering and obtaining the locked netlist, and one as the reference known as oracle). This group of attacks could be referred to as **oracle-guided** attacks. On the other hand, those attacks with no need for having access to the oracle are called **oracle-less** attacks. During the last decade, most of the attacks are members of oracle-guided attacks. However, in many real cases, the adversary cannot obtain one additional activated chip, and the fulfillment of this requirement is hard to be achieved. So, the adversary has to rely on only oracle-less attack models.

Many of the de-obfuscation attacks are **invasive**. They require access to the netlist of the chip. Acquiring the netlist of the chip could be accomplished differently at various stages of the IC supply chain, as demonstrated in Fig. 2. For instance, the malicious end-user as the adversary can obtain the fabricated IC from the field/market, and then reconstructs the netlist through physical reverse engineering (orange path), in which the main steps of physical reverse engineering are de-packaging, delayering, imaging, image (of metal layers) processing, and re-constructing the netlist. In this case, during the physical reverse engineering, since the key is stored in TPM, it will be wiped out in the de-packaging stage, and the obtained netlist would be locked.

Regarding the obtained (reverse-engineered) netlist, the same happens in all other cases, and the extracted netlist would be the locked version with no (correct) key. Another example is when the adversary might be at the foundry, and once they receive the GDSII of the chip from the design house to be fabricated, the GDSII is provided without the correct key (locked). In this case, although no delayering or physical infiltration is required, GDSII is required to be accessed and evaluated for netlist extraction, thus we consider it a *weak* invasive model. Even while the adversary is a rogue insider in the design house, except verification engineer[5] who needs the key for verification purposes, there is no necessity of sharing the key for stages like integration, synthesis, floorplanning, etc., showing that the obtained design is available with no key.

Unlike invasive attacks, there exists a very limited number of **semi-invasive** and **non-invasive** de-obfuscation attacks. In these attacks, the adversary relies on optical probing, such as electro-optical probing (EOP) and electro-optical frequency management (EOFM). Such attacks focus on pinpointing and probing the logic gates and flip-flops of the circuits containing the secrets. So, regardless of the logic locking technique used in the circuit, this group of attacks, which will be discussed through the paper, would be able to be a real threat for revealing the security assets like logic locking key.

The availability of design-for-testability (DFT) structure, i.e. scan chain architecture, for testability/debug purposes in ICs opens a big door for the attackers to assess and break logic locking techniques. Hence, many of the attacks on logic locking assume that the **scan chain is OPEN**. Fig. 6 shows a simplified scan architecture with two chains. Assuming that the scan chain is *OPEN*, SE, SI, and SO pins would be available. So, the adversary can reach (control and observe) each combinational part, e.g. CL_1 and CL_2 in Fig. 6, whose FFs are part of the scan chain. The scan chain access allows the adversary to divide the de-obfuscation problem into a bunch of much smaller sub-problems (each CL), and assess them independently. However, it is very common for an IC to limit/restrict access to the scan chain for security purposes. But, even while **the access to the scan chain is NOT OPEN** (e.g. SO pins are burned), some other de-obfuscation attacks have studied and demonstrated the possibility of retrieving the correct key/functionality of the locked circuit via primary inputs/outputs (PI/PO).

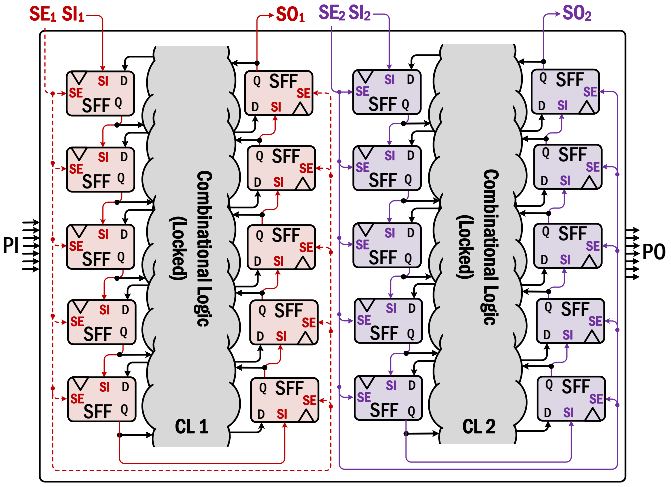


Fig. 6: Design for Testability (Scan Chain) Architecture in ICs.

Furthermore, the adversary located at the foundry might have the capability of priming and manipulating the GDSII to insert hardware Trojan for different purposes, such as key leakage through PO after the activation. The capability of inserting stealthy Trojans around the logic locking circuitry will be able to (1) disable the test access or (2) retrieve the logic locking key through PO. Hence, the capability of adding hardware Trojans when logic locking is in place, and being non-detectable by Trojan detection techniques could be a part of adversary capabilities that required meticulous consideration by the designers.

3 Logic Locking: Countermeasures

Starting 2008, numerous logic locking techniques have been introduced in the literature each trying to introduce a new countermeasure against the existing de-obfuscation attacks. Fig. 7 demonstrates a top view of existing notable logic locking techniques introduced so far. Based on the previously discussed basics and definitions of logic locking, we categorized them into different groups: (1) primitive, (2) point function, (3) cyclic, (4) LUT/routing, (5) scan-based locking/blocking, (6) sequential/FSM, (7) timing-based, (8) eFPGA-based, and (9) high-level (RTL/HLS). The main specification of any member of each countermeasure is illustrated in Fig. 7. In this section, regardless of the

[5] We assume that the verification team is always trusted.

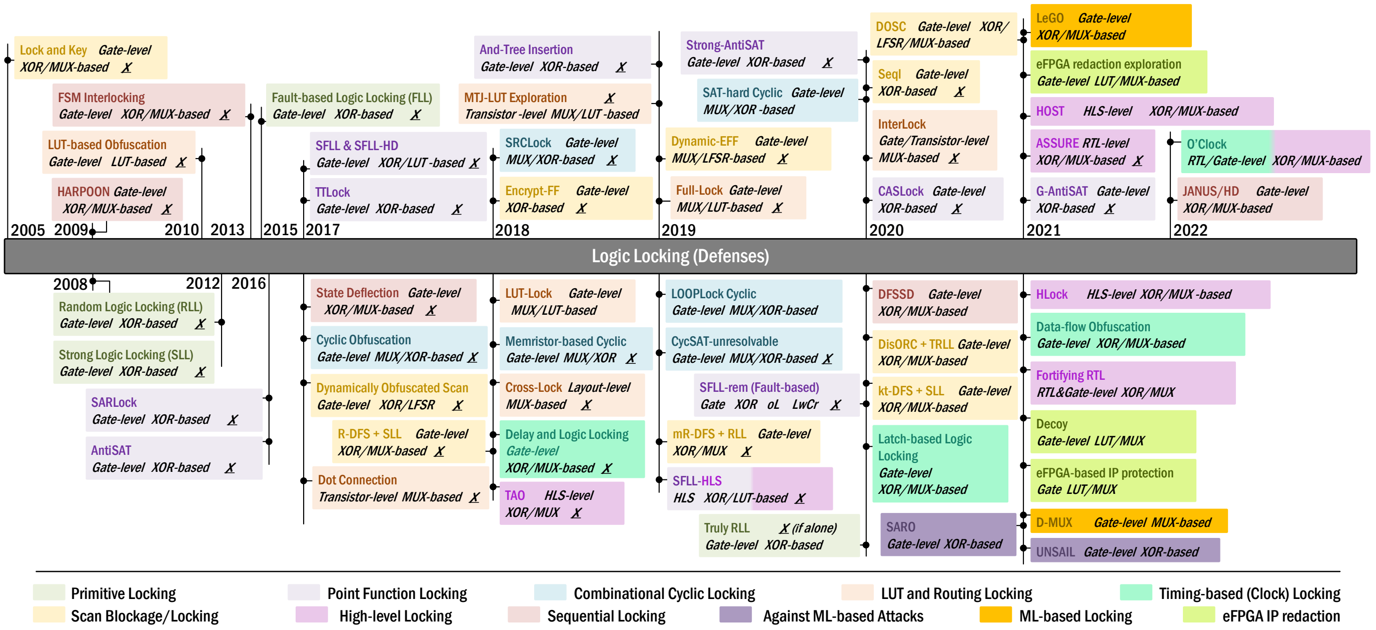

Lock and Key Gate-level XOR/MUX-based ✗
FSM Interlocking Gate-level XOR/MUX-based ✗
LUT-based Obfuscation Gate-level LUT-based ✗
HARPOON Gate-level XOR/MUX-based ✗
Fault-based Logic Locking (FLL) Gate-level XOR-based ✗
SFLL & SFLL-HD Gate-level XOR/LUT-based ✗
TTLock Gate-level XOR-based ✗
And-Tree Insertion Gate-level XOR-based ✗
MTJ-LUT Exploration ✗ Transistor-level MUX/LUT-based
SRCLock Gate-level MUX/XOR-based ✗
Encrypt-FF Gate-level XOR-based ✗
Strong-AntiSAT Gate-level XOR-based ✗
SAT-hard Cyclic Gate-level MUX/XOR-based
Dynamic-EFF Gate-level MUX/LFSR-based ✗
Full-Lock Gate-level MUX/LUT-based ✗
DOSC Gate-level XOR/LFSR/MUX-based
Seql Gate-level XOR-based ✗
InterLock Gate/Transistor-level MUX-based ✗
CASLock Gate-level XOR-based ✗
LeGO Gate-level XOR/MUX-based
eFPGA redaction exploration Gate-level LUT/MUX-based
HOST HLS-level XOR/MUX-based
ASSURE RTL-level XOR/MUX-based ✗
O'Clock RTL/Gate-level XOR/MUX-based
G-AntiSAT Gate-level XOR-based ✗
JANUS/HD Gate-level XOR/MUX-based
2005 2009 2010 2013 2015 2017 2018 2019 2020 2021 2022
Logic Locking (Defenses)
2008 2012 2016
Random Logic Locking (RLL) Gate-level XOR-based ✗
Strong Logic Locking (SLL) Gate-level XOR-based ✗
SARLock Gate-level XOR-based ✗
AntiSAT Gate-level XOR-based ✗
State Deflection Gate-level XOR/MUX-based ✗
Cyclic Obfuscation Gate-level MUX/XOR-based ✗
Dynamically Obfuscated Scan Gate-level XOR/LFSR ✗
R-DFS + SLL Gate-level XOR/MUX-based ✗
Dot Connection Transistor-level MUX-based ✗
LUT-Lock Gate-level MUX/LUT-based
Memristor-based Cyclic Gate-level MUX/XOR ✗
Cross-Lock Layout-level MUX-based ✗
Delay and Logic Locking Gate-level XOR/MUX-based ✗
TAO HLS-level XOR/MUX ✗
LOOPLock Cyclic Gate-level MUX/XOR-based
CycSAT-unresolvable Gate-level MUX/XOR-based ✗
SFLL-rem (Fault-based) Gate XOR oL LwCr ✗
mR-DFS + RLL Gate-level XOR/MUX ✗
SFLL-HLS HLS XOR/LUT-based ✗
Truly RLL ✗ (if alone) Gate-level XOR-based
DFSSD Gate-level XOR/MUX-based
DisORC + TRLL Gate-level XOR/MUX-based
kt-DFS + SLL Gate-level XOR/MUX-based
Latch-based Logic Locking Gate-level XOR/MUX-based
SARO Gate-level XOR-based
HLock HLS-level XOR/MUX-based
Data-flow Obfuscation Gate-level XOR/MUX-based
Fortifying RTL RTL&Gate-level XOR/MUX
Decoy Gate-level LUT/MUX
eFPGA-based IP protection Gate LUT/MUX
D-MUX Gate-level MUX-based
UNSAIL Gate-level XOR-based
Primitive Locking
Point Function Locking
Combinational Cyclic Locking
LUT and Routing Locking
Timing-based (Clock) Locking
Scan Blockage/Locking
High-level Locking
Sequential Locking
Against ML-based Attacks
ML-based Locking
eFPGA IP redaction

Fig. 7: Logic Locking Techniques.

existing successful attacks, we will briefly review the main specification of each logic locking category.

3.1 Primitive Logic Locking

The first group of countermeasures on combinational circuits is primitive techniques, including EPIC *a.k.a.* random logic locking (RLL) [11], strong (interference-based) logic locking (SLL) [12], and fault-based logic locking (FLL) [44]. For example, in EPIC (RLL), as its name implies, XOR-based key gates will be inserted at some arbitrarily chosen points in the circuit. All primitive techniques are XOR-based and implemented at the gate level. Since a locked circuit initiated with an incorrect key corrupts the PO by propagating errors at POs, in SLL and FLL, some features of automatic test pattern generation (ATPG) tools and testability specification, such as controllability/observability, and faults propagation/masking have been used for selecting the location of XOR-based key gates. For instance, in SLL [12], specifications like key-gates exclusion, isolation, cascading (running), mutability, and convergence have been examined, thereby by forming an interference graph of key gates, the best candidates are selected for key gate insertion, helping to enhance the strength of the logic locking against testing-based attacks (in comparison with RLL). However, using these features results in a notable reduction of corruptibility in SLL and FLL compared to that of RLL. In addition, in [42], a truly RLL has been defined that relies on XOR/XNOR insertion around inverters or buffers that is for randomly locking of signal polarities. Table 2 summarizes the specification of this breed of logic locking, showing that all are attacked by the *Boolean satisfiability* (SAT) attack, discussed in §4.1.2.

3.2 Point Function Logic Locking

The main aim of point function techniques is to minimize the number of available input patterns showing that a specific key is incorrect[6]. This breed was the first attempt against the Boolean satisfiability (SAT) attack, which can prune the keyspace by ruling out the incorrect keys in a fast-convergence approach [45], [46]. In the literature, this group of logic locking techniques is known as provably logic locking techniques. A logic locking technique is *provably* secure once they are algorithmically resilient (cannot be broken) against any type of I/O query-based attacks. SARLock and Anti-SAT are the very first logic locking techniques in this category [47], [48]. As demonstrated in Fig. 8(a), the main structure of point function techniques relies on a *flipping* (corrupting) circuitry that flips (corrupts) the limited PO(s) only for a very limited number of input patterns (e.g., 1) per each incorrect key. Also, a *masking/restore* (correcting) circuitry has been engaged in point function techniques to re-flip the impact of flipping circuitry, guaranteeing the correct functionality when the correct key is applied.

Point function techniques could be applied on the function-modified (stripped) version of the circuit, known as stripped function logic locking [51]. In such techniques, the original part is modified, and in at least one minterm, the (affected) POs are *ALWAYS* flipped/corrupted. This is done

TABLE 2: Specification of Primitive Logic Locking Techniques.

Logic Locking	Mechanism	Overhead	Corrupt	Attacked by
RLL [11]	Insertion of XOR key gates at *random* places	■□□□□	■■■	ATPG:§4.1.1, SAT:§4.1.2
FLL [12]	Insertion of XOR key gates at *lower* testable points	■■□□□	■■□	SAT:§4.1.2
SLL [44]	Interference-based key gate insertion (non-mutuable, non-isolated, etc.)	■□□□□	■■□	SAT:§4.1.2
TRLL^{*1} [42]	Insertion of XOR key gates at buffers and inverters	■□□□□	■■■	SAT:§4.1.2

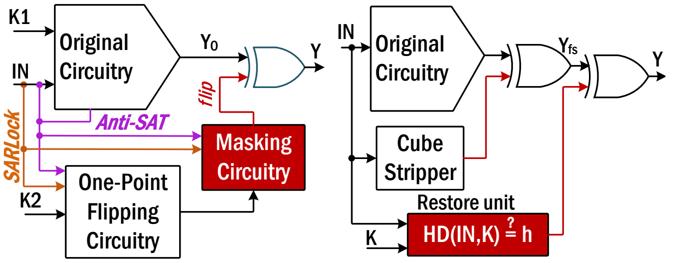
^{*1} Here TRLL is evaluated as an individual locking technique.

[6] The best case is *ONE* input pattern per each incorrect key.

TABLE 3: Specification of Point-Function Logic Locking Techniques.

Logic Locking	Mechanism	Overhead	Corrupt	Attacked[*1] by
SARLock [47]	Adding flipping circuit to corrupt only ONE input pattern per each incorrect key + masking circuit for correct key	■□□□□	■□□	Removal:§4.1.3.1, Approximate:§4.1.2.2, EDA-based:§4.1.3.5, Bypass:§4.1.2.3
AntiSAT [48]	Merging of `AND`ed two toggled functions (`g` & $\bar{\texttt{g}}$) as the flipping+masking circuitry together	■□□□□	■□□	SPS/Removal:§4.1.3.2, Approximate:§4.1.2.2, EDA-based:§4.1.3.5, Bypass:§4.1.2.3,
And-Tree [49]	hard-coded `AND` trees as flipping circuitry + a generic masking circuitry	■□□□□	■□□	CASUnlock:§4.1.3.4, EDA-based:§4.1.3.5
TTLock [50]	SARLock + Stripping original circuit for one minterm	■■□□□	■□□	EDA-based:§4.1.3.5, GNNUnlock:§4.3.3, FALL:§4.1.3.3
SFLL-HD [51]	SARLock + Stripping original circuit for $d = {}^{k}C_{h}$ minterms (h: HD and k: key size)	■■□□□	*variant* (w.r.t. h)	EDA-based:§4.1.3.5, GNNUnlock:§4.3.3, FALL:§4.1.3.3
SFLL-flex [51]	Adding the flexibility of protecting user-defined input patterns in a point-function manner + LUT-based restore circuitry	*variant* (w.r.t. user-defined list)	*variant* (w.r.t. user-defined list)	EDA-based:§4.1.3.5
SFLL-rem [52]	removing logic for creating the corruption based on fault insertion + a generic restore circuitry	■□□□□	*variant* (w.r.t. s-a fault location)	EDA-based:§4.1.3.5
G-AntiSAT [53]	Merging of `AND`ed two toggled functions (`f` & `g`) as the flipping+masking circuitry together	■■□□□	*variant* (w.r.t. specs of `f` & `g`)	EDA-based:§4.1.3.5
S-AntiSAT [54]	App-level input pattern protection + generic restore circuitry	*variant* (w.r.t. input patterns list)	*variant* (w.r.t. input patterns list)	EDA-based:§4.1.3.5
CAS-Lock [55]	Variant `AND-OR` tree as variant corruptible circuitry	■■□□□	*variant* (w.r.t. `OR` gates)	EDA-based:§4.1.3.5, CASUnlock:§4.1.3.4

[*1] Per each logic locking technique, there might be more successful attacks than the ones listed here, and we listed the most notable ones directly applicable to.



(a) Anti-SAT and SARLock **(b)** Stripped-based Point Function

Fig. 8: The Structure of Point Function Techniques.

using *cube stripper* module as demonstrated in Fig. 8(b). The *restore* unit builds the flipping and masking circuitry. In the stripped function techniques, for each incorrect key, there exists a very limited number of input patterns (e.g. 1) *plus* one extra input pattern (caused by the stripped function) that corrupts the POs (double point corruption). In fact, approaches with no stripping accomplish the flipping at one point (one input pattern per an incorrect key), while others with enabled stripping do the flipping at two points.

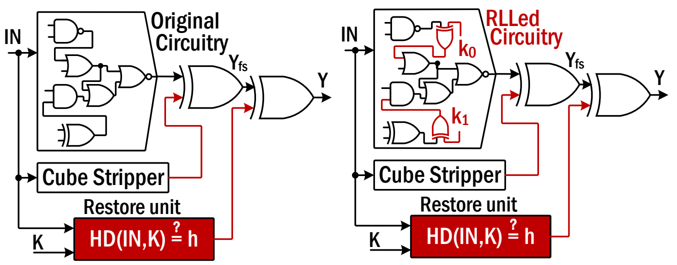
Point function techniques could be categorized as XOR-based techniques. In SFLL-flex [51], LUTs are also engaged for minterm generations of corruption/correcting sub-circuits. Except for SFLL-HLS [56] that is implemented at the high-level (HLS), all techniques in this category are implemented at the gate level (post-synthesis). Since the point function sub-circuitry will be added for a limited (e.g. 1) number of POs and corrupts the PO for a very limited number of input patterns, the corruptibility of this breed of obfuscation is very low. However, there also exist more recent point function techniques that try to overcome the low corruptibility of such techniques with the introduction of different flipping (corrupting) circuits [53], [54], [55]. Table 3 summarizes the main specification of point function techniques. As shown, two new studies, i.e., sparse prime implication and Valkyrie (EDA-based) break all the variants in this group, showing the big structural issue behind this breed of logic locking.

3.2.1 *Compound Logic Locking*

The point function logic locking limits the corruption per each incorrect key to (i) a very little set of input patterns, and (ii) observable at a very little set of POs[7]. Hence, this breed suffers from a very low output corruption that significantly undermines its strength. Hence, a new paradigm was first introduced after point function techniques, in which the composition of different logic locking has been discussed, known as **compound** logic locking techniques. In compound logic locking, two or more different logic locking techniques can be used and applied simultaneously to a circuit *if and only if* none of them weaken the other ones. For instance, since the locking parts of the point function techniques are completely decoupled from the original part, as demonstrated in Fig. 8, and since the primitive logic locking techniques provides high corruptibility, as demonstrated in Fig. 9, these two can easily be combined to mitigate the low-corruptibility issue of point function techniques, and also getting the benefit of high resiliency from point function techniques.

The concept of compound logic locking can be used for any feasible combination. For instance, in [57], a bilateral logic encryption has been introduced in which where a

[7] The best case (in terms of enhancing the security) is that *ONE* input pattern can only rule out *ONE* incorrect key, and the corruption per each incorrect key can be witnessed at only *ONE* PO.



(a) Stripped-based Point Function **(b)** Compound (SFLL+RLL)

Fig. 9: The Structure of Compound Logic Locking Techniques.

TABLE 4: Specification of Compound Logic Locking Techniques*1.

Logic Locking	Mechanism	Overhead	Corrupt	Attacked by
X+Y	Combination of primitive + point function	■■□□□	■■■	compound attacks:§4.1.2.4, Approximate: §4.1.2.2

*1: This table is only for primitive + point function techniques.
X: Any primitive logic locking technique
Y: Any point-function logic locking technique

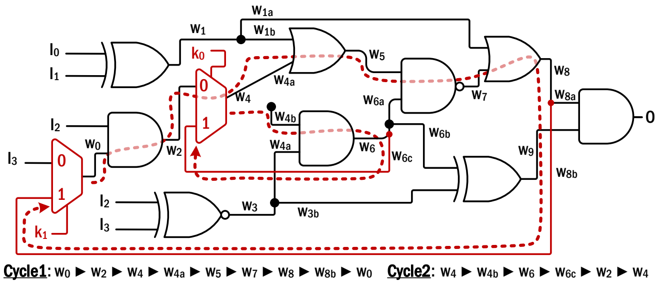


Fig. 10: An Example of Cylic Obfuscation using 2-to-1 MUXes.

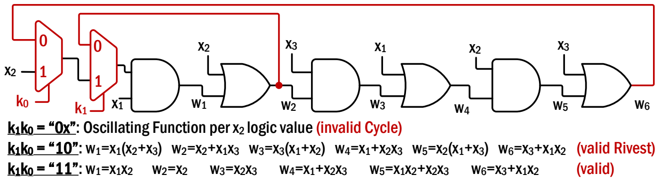

x$_2$
k$_0$
k$_1$
x$_1$
x$_2$
x$_3$
W$_1$
W$_2$
W$_3$
W$_4$
W$_5$
W$_6$
k$_1$k$_0$ = "0x": Oscillating Function per x$_2$ logic value (invalid Cycle)
k$_1$k$_0$ = "10": W$_1$=x$_1$(x$_2$+x$_3$) W$_2$=x$_2$+x$_1$x$_3$ W$_3$=x$_3$(x$_1$+x$_2$) W$_4$=x$_1$+x$_2$x$_3$ W$_5$=x$_2$(x$_1$+x$_3$) W$_6$=x$_3$+x$_1$x$_2$ (valid Rivest)
k$_1$k$_0$ = "11": W$_1$=x$_1$x$_2$ W$_2$=x$_2$ W$_3$=x$_2$x$_3$ W$_4$=x$_1$+x$_2$x$_3$ W$_5$=x$_1$x$_2$+x$_2$x$_3$ W$_6$=x$_3$+x$_1$x$_2$ (valid)

Fig. 11: Rivest Sub-circuit as Decoy-based Cyclic Logic Locking.

low-corruption logic locking technique and a routing-based approach have been integrated to show the effectiveness of compound logic locking against a wider range of attacks. Table 4 shows general definition of compound techniques.

3.3 Cyclic-based Logic Locking

As its name implies and as shown in Fig. 10, cyclic logic locking will add key gates that control the possibility of adding/removing combinational cycles into the circuit. Having combinational cycles will add difficulties for the CAD tools (like synthesis and timing analysis) to deal with such circuits. Many CAD tools do not allow the designers to have combinational cycles in the circuit. However, the designer would be able to handle the combinational cyclic paths during the physical design in a manual manner, like adding constraints for false paths. Hence, combinational cycles are used commonly as a means of logic locking recently. In cyclic-based logic locking, different approaches are considered through different studies:
(i) *Adding false cycles*: In this case, similar to examples demonstrated in Fig. 10, some combinational cycles have been added into the design that should not be remained once the circuit is unlocked [58]. Having such cycles in the design creates uncertainty, glitches, and malfunction while the correct key is not provided.
(ii) *Adding misguiding combinational cycles*: Since the design must have no combinational cycles by itself, some approaches engage a set of valid misguiding combinational cycles, such as Rivest circuits [59] as shown in Fig. 11, to violate such basic assumption [60], [61], in which cycles are part of the original functionality. However, template-based logic sub-circuits will be used for building this kind of cycle in a design that makes them vulnerable to structural analysis.
(iii) *Exponential increase of cycles*: To increase the complexity of cyclic-based logic locking, some studies have been evaluated the possibility and techniques that can be applied to exponentially increase the number of cycles *w.r.t.* the number of feedbacks [60], [62]. Exponentially increasing the number of cycles, will exponentially enhance the complexity of cyclic analysis. However, the existing approaches will also raise the area overhead significantly.

We are also witnessing some other studies that are using other techniques for increasing the complexity of cyclic-based logic locking, such as the implementation of cyclic-locking using memristor-CMOS logic circuits, and inserting cycle pairs (duplicating sub-circuits), which are useful for hiding and misguiding the adversary for any kind of analysis on added/inserted cycles [63], [64], [65].

TABLE 5: Specification of Cyclic Logic Locking Techniques.

Logic Locking	Mechanism	Overhead	Corrupt	Attacked by
Cyclic [58]	adding false combinational cycles into the circuit	■■□□□	■■■	CycSAT:§4.1.2.5
SRCLock [60]	exponentially increasing false combinational cycles w.r.t. the number of feedbacks	■■■□□	■■■	CP&SAT:§4.1.2.7
memristor-based cyclic [63]	unresolvable combinational cycles + memristor cells for camouflaging cycles	■■□□□	■■■	iCySAT:§4.1.2.5
CycSAT-Unresolvable [61]	unresolvable combinational cycles	■■□□□	■■■	iCySAT:§4.1.2.5, BeSAT:§4.1.2.5
SAT-hard cyclic [62]	exponentially increasing false combinational cycles w.r.t. the number of feedbacks + rivest cycles + non-occuring cycles	■■■□□	■■■	NONE
LOOPLock [64]	creating pairs of cycles that increasing the complexity of cycle analysis	■■□□□	■■■	NONE

In general, since re-routing is required to generate the combinational cycles, all cyclic logic locking techniques use key-based MUX gates at different levels of abstraction. Also, in some cases, key-based XOR gates are used to build the model. All existing cyclic logic locking techniques are implemented at the gate level or transistor level. Since rerouting increases the correlation of wiring in different logic cones of the circuit, the corruptibility of this group of logic locking would be high. Table 5 shows the main specification of this breed of logic locking. Although some of these techniques are not broken so far, they are not considered as a mainstream in logic locking since it results in challenging industry adoption specifically in implementation and physical based EDA tools.

3.4 LUT/Routing-based Logic Locking

Some logic locking techniques derive the benefit from the full configurability of look-up-tables (LUTs). Relying on the fact that a u-input LUT can build all 2^{2^u} possible functions, as demonstrated in Fig. 4, in existing LUT-based logic locking techniques, some actual logic gates of the original design are replaced with LUTs (same size or larger LUTs), and the initialization (configuration) values of the LUTs is considered as the secret (key) and would be initiated after the fabrication. In existing LUT-based logic locking, the followings have been investigated as the crucial factors [66], [67], [68], [69]:

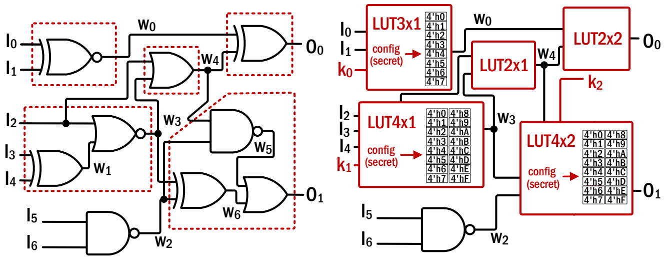
(i) *Size of LUTs*: A point-to-point LUT-based replacement has been implemented in the existing LUT-based techniques, in which each selected gate will be replaced by a *same-size* (LUT2x1, LUT2x2 in Fig. 12) or *larger* (LUT3x1, LUT4x1, and LUT4x2 in Fig. 12) LUTs. Enlarging LUTs (Increasing the number of inputs) will exponentially increase the complexity of the logic locked circuit, however, it also increase the area/power overhead exponentially. In enlarged LUTs, the extra (unused) inputs can be used as the key to expand the complexity space (e.g., $k_{0:2}$ in Fig. 12).

(ii) *Number of LUTs*: The number of gates selected to be replaced with same-size/larger LUTs directly affects the complexity of the logic locked circuit, either functionally or structurally. However, similar to the side-effect of LUT size, using more LUTs will significantly increase the overhead of the logic locking approach.

(iii) *Replacement strategy*: The location of replacement in LUT-based logic locking plays a crucial role in making the resultant circuit more complex in the domain of logic locking. The work LUT-lock [67] has investigated how a good replacement strategy can render a higher grade of complexity while a heuristic approach has been used.

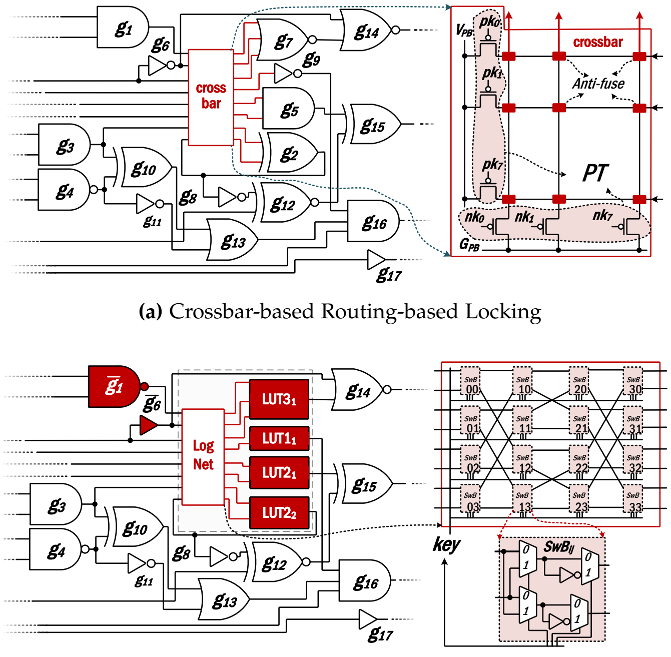
In general, although LUT-based logic locking can provide more reliable resilience against existing attacks, it extremely suffers from incurred overhead, which limits the application of this breed of logic locking. The Existing LUT-based techniques are all implemented at the gate level or transistor level and based on the placement strategies used for LUT insertion, the corruptibility of these techniques is also high.

Similar to LUT-based logic locking, which can be built using a MUX-based (tree of MUXes) structure, numerous studies have exploited the concept of MUXes for another breed of logic locking, known as routing-based locking. In routing-based locking, false/decoy/invalid paths could be added using re-routing modules in different levels of abstraction. For instance, at the gate level of abstraction, MUXes could be used for re-routing, and at layout-level, metal-insulator-metal (MIM) could be used for re-routing between metal layers as a means of logic locking.



(a) Stripped-based Point Function **(b)** Compound (SFLL+RLL)

Fig. 12: LUT-based Logic Locking Technique.



(a) Crossbar-based Routing-based Locking

(b) Logarithmic-based Routing-based Locking

Fig. 13: Routing-based Locking Technique.

Dealing with the complexity of the routing modules has been evaluated for a long time as a part of physical P&R in both FPGAs/ASICs [70], [71], [72]. Hence, this complexity is borrowed here as a means of locking in these techniques. But the concept of routing-based locking was firstly derived from the usage of wiring/connection concealment in split-manufacturing techniques [73], [74], [75]. In these techniques, different forms of re-routing, such as perturbation, lifting nets, etc., have been engaged to hide the connections between BEOL and FEOL. Hence, a similar approach has been used in a circuit for locking purposes. For instance, Fig. 13 shows two different routing-based techniques, i.e. Cross-lock [76] and full-lock [77]. As demonstrated, in routing-based locking, some wires of the design will be selected and the routing module, whose configuration is the locking key, will conceal the connection between sources and sinks.

Unlike point function techniques that decrease the pruning power of the existing I/O query-based attacks (e.g., the SAT attack) by exponentially increasing the required number of I/O pairs, routing-based techniques show how a routing module can extremely increase the complexity of design per each stimulus [77]. Routing-based locking techniques have investigated different factors that significantly affect the outcome of this breed of locking solutions as listed here:

(i) *Abstraction Layer of Implementation*: Routing modules can be implemented at different layers of abstraction. Moving towards lower levels (e.g., layout) significantly increases the implementation efforts and challenges, however, if implemented correctly at lower levels, it incurs considerably lower overhead. Techniques like Cross-lock and interconnect-based locking are implemented at the layout level [76], [78]. Full-lock implements routing module at the gate level [77]. An extension on Full-lock is also implemented at gate-level [79], and Interlock is also implemented at both gate level and transistor-level [80].

(iii) *Topology of Routing Modules*: Once a routing module is in place, the topology of interconnections is a determining characteristic. Hence, in routing-based locking techniques, different topologies have been used, such as crossbar in [76], [78], logarithmic networks in [77], [79], [80], [81], and irregular [82]. The type of topology can directly affect the resiliency of the countermeasure against different attacks. For instance, logarithmic networks with deeper layers can extremely increase the complexity of the model in any graph-based and routing-based analysis. However, the topology can negatively affect the overhead, particularly delay of timing paths selected for routing locking.

(ii) *Cycle-involvement in Routing-based Locking*: Based on the wire selection in routing-based locking techniques, cycles might be created once an incorrect key will be applied. For instance, as shown in Fig. 13, the output of $\texttt{g2/LUT2}_2$ is back and connected to the input of routing module. Although the creation of cycles will increase the complexity of locking techniques, to avoid any design/implementation flow challenges dealing with cycles, in more recent techniques [79], [80], since actual timing paths of the design are embedded into routing modules, there exists no possibility of cycle creation.

(iv) *Logic Embedding in Routing Module*: To increase the complexity of routing modules, recent works [77], [79], [80] have investigated the embedding of logic gates into (switch) layers of routing module. For instance, switch boxes of `LogNet` in Fig. 13(b) provides configurable inversion. Hence, Gates like `g1` and `g5` is toggled (stripped) and the `LogNet` will recover the correct logic values. Hence, in these techniques, with the full configuration capability, stripping the functionality of the original circuit would be an extra available option.

Table 6 summarizes the specification of routing-based locking countermeasures. In comparison with other breeds, routing-based locking will incur higher overhead. However, more recent studies, such as coarse-grained eFPGA-based IP redaction (§3.9) show that this form of locking that inherits full reconfigurability for locking purposes still has significant potential as a means of logic locking. As demonstrated, all existing routing-based locking solutions are broken, mostly by very recent ML-based attacks described in 4.3.3.

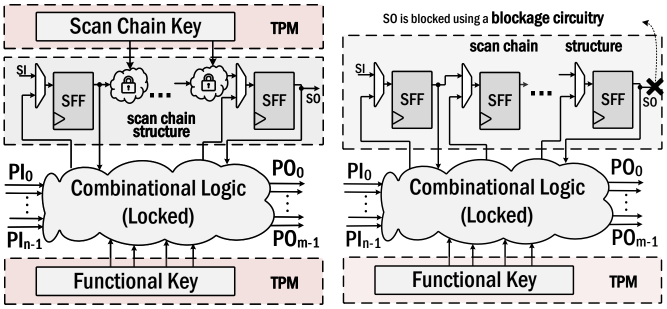
TABLE 6: Specification of Routing-based Locking Techniques.

Logic Locking	Mechanism	Overhead	Corrupt	Attacked by
Cross-Lock [76]	Insertion of key-based layout-level cross-bar with combinational cycles	■■□□□	■■■	CP&SAT, NNgSAT: §4.1.2.7
Dot Connection [78]	Insertion of key-based layout-level cross-bar with combinational cycles	■■□□□	■■■	CP&SAT, NNgSAT: §4.1.2.7
Full-Lock [67]	Insertion of gate-level key-based logarithmic routing modules with inversion stripping	■■■■■	■■■	CP&SAT, NNgSAT: §4.1.2.7
Modeling routing [67]	Insertion of gate-level key-based logarithmic routing modules with logic embeded-ded	■■■■■	■■■	Untangle: §4.3.3
InterLock [67]	Insertion of gate or transistor-level key-based logarithmic routing modules with logic embedded + stripping	■■■□□	■■■	Untangle: §4.3.3

3.5 Scan Chain Logic Locking/Blocking

Providing the access to the internal parts of the circuit for test/debug purposes is almost inevitable in modern/complex ICs. As discussed previously in §2.2, design-for-testability (DFT)-based synthesis in ASIC design flow provides this capability for the designers by adding scan (register) chain structures into the circuit. The DFT-based scan chain architecture has been widely used in most modern ICs. Even in cryptographic circuits, which have very sensitive information, such as encryption key, to have a high fault coverage, the test/debug step requires access to the scan chain to control and observe the internal states of the design-under-test (DUT). The full controllability and observability requirement in DFT-based (scan-based) testing, however, might pose security threats to ICs with security assets, such as locked circuits that keep their own secret, i.e., the unlocking key. Hence, DFT access allows the adversary to split and divide the bigger problem into a set of independent smaller problems by splitting the whole circuit into `c` smaller and only combinational logic parts, whose access to their register elements are available via scan chains (e.g., `CL1` and `CL2` of Fig. 6 through $\{\texttt{SI}_1, \texttt{SE}_1, \texttt{SO}_1\}$ and $\{\texttt{SI}_2, \texttt{SE}_2, \texttt{SO}_2\}$, respectively). We further demonstrate that scan chain access availability is one of the main assumptions of the SAT attack, discussed in §4.1.2.1.

Since restricting the scan chain access can enlarge the problem space domain from small combinational sub-circuits to a whole large sequential circuit, a breed of logic locking techniques focus on different methodologies that only and primarily target the locking of the scan chain architecture, known as scan-based logic locking techniques [83], [84], [85], [86], [87], [88]. By using these techniques, the scan chain pins, i.e. scan-enable (*SE*), scan-in (*SI*), and particularly scan-out (*SO*) would be limited/restricted for any unauthorized access, and these approaches do not allow the adversary to get the benefit these pins, and they lose the chance of direct/independent controlling/observing the combinational parts of the circuit. Similar to scan-based logic locking techniques, some approaches *BLOCK* the access to the scan chain pins, particularly *SO* [36], [37], [38], [40], [42], called scan blockage techniques. The scan block-



(a) Scan-based Locking

(b) Scan Blockage

Fig. 14: Scan Chain Locking/Blocking Technique.

age will happen based on a sequence of specific operations in the scan chain structure. For example, assuming that (part of) the key is loaded into the scan chain after activation, switching *SE* to 1 (enabling shift mode) might be perilous. Hence, shift operation would be limited after the activation. Fig. 14 shows the top view of both scan locking and scan blockage techniques. Compared to the scan-based logic locking techniques, the scan blockage techniques incur less area overhead. However, they have some limitations during the test phase, such as limiting the functional test and increasing the test time and complexity.

The following describes the main specifications and principles of scan chain locking/blockage techniques:

(i) *Combination with Functional Logic Locking*: Scan chain locking/blockage is orthogonal to other functional logic locking techniques. In fact, the scan-based logic locking techniques only lock the scan structure. So, even while an incorrect key is initiated in the circuit, it only affects the scan chain functionality and has no impact on the functionality of the circuit. Hence, these techniques need to be combined with one of the other discussed combinational logic locking techniques, and the existing techniques mostly use one of the primitive logic locking techniques. Some techniques use RLL [37], some other techniques use SLL [38], [83], [89], and one scan blockage technique is combined with truly-RLL (TRLL) [42]. Hence, the corruptibility of these approaches is dependent on the engaged functional logic locking techniques.

(ii) *Implementation Abstraction Layer*: Since scan chain locking/blockage techniques must be applied to the scan chain structure, they only could be done after design-for-testability (DFT) synthesis. So, these techniques could be implemented at the gate level, the transistor level, or the layout level. All the existing techniques are implemented at the gate level.

(iii) *Test Coverage/Overhead*: Scan chain locking/blockage can negatively affect the testability metrics, e.g., test coverage, resulting in the reduction of reliability of the circuits. Hence, one crucial criterion in this breed is to keep the test coverage as close as possible to that of the original circuit before scan locking/blocking. Additionally, in some of the blockage techniques, extra test pins have been added to support high testability coverage and efficient resiliency [36], [37], [38], [40]. However, adding extra pins can extremely increase the die size of the manufactured chip.

TABLE 7: Specification of Scan-based Locking/Blocking Techniques.

Logic Locking	Type	Mechanism	Overhead	Attacked by
DOS [83], [89]	Locking by PRNG	Dynamic LFSR-based Shuffling and Toggling with Shadow Scan Insertion	■■□□□	ScanSAT: §4.2.3.1
Encrypt-FF [84]	Locking by XOR	XOR-based key gate insertion within the scan chain	■□□□□	ScanSAT: §4.2.3.1
Robust design for security (R-DFS) [40]	Blockage	A custom secure DFF for key storage + SO blockage circuitry for post-activation	■■□□□	shift&leak: §4.2.3.3
dynamic-EFF [85]	Locking by PRNG	Adding PRNG and malfunctioning by PRNG for incorrect keys	■■□□□	DynUnlock: §4.2.3.2
Extended R-DFS [37]	Blockage	A custom secure DFF for key storage + SO blockage circuitry for post-activation	■■□□□	shift&leak: §4.2.3.3
seql [86]	Locking by XOR	XOR-based key gate insertion between functional and scan chain paths	■□□□□	ScanSAT: §4.2.3.1
kt-DFS [38]	Blockage	A custom secure DFF for key storage + SO blockage circuitry for post-activation	■■□□□	NONE
DisORC [42]	Blockage	SO Blockage circuitry with full shift disable after activation + disabling shift + oracle dishonesty	■■□□□	NONE
DOSC [88]	Locking by PRNG + counter	Dynamic LFSR-based Shuffling and Toggling with Shadow Scan Insertion	■■□□□	NONE

(iv) *Locking Operation Method*: In scan chain locking, the locking part can operate statically or dynamically. In the dynamic approach, components like LFSR and/or PRNG have been engaged that changes the configuration at run-time [83], [85], [88], [89]. However, in techniques with statically scan chain locking [84], [86], the configuration is always fixed. The dynamicity -if implemented correctly- is always a huge bar for the adversary, specifically, once they rely on I/O query-based attacks, and dynamicity invalidates all learned information acquired on previous acts and enforces the adversary to restart the process.

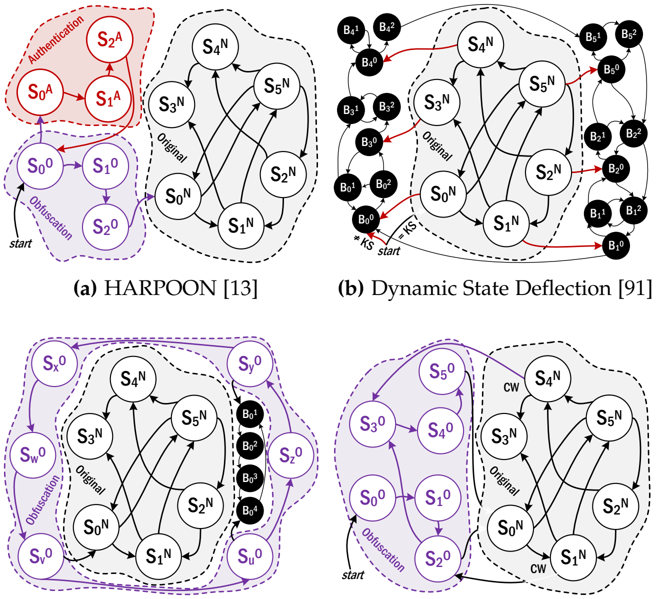
(v) *The Security of Scan Chain Architecture*: Scan chain locking/blocking brings some modification into the structure of scan chain(s). This modification particularly comes from scan blockage techniques. For instance, with a more robust scan chain architecture, the designer might integrate all regular FFs and key-dedicated FFs into a common scan chain. However, this can undermine the security of the logic locking key, regardless of locking techniques, if a leakage possibility could be found through the modified scan chain structure or cells. So, the security of the scan chain must be evaluated and guaranteed while scan chain locking/blockage is applied to the circuit.

Table 7 summarizes the main specification of existing scan-based locking/blocking techniques. At the moment, a combination of one scan-based logic locking or scan blockage with functional logic locking techniques shows high resiliency and received significant attention. Scan-based protection extremely increases the size and complexity of formulation for any attacking model on logic locking. We further discuss that this type of locking is an integral part of reliable and security-guaranteed logic locking techniques against all existing threats.

3.6 FSM/Sequential Logic Locking

One basic assumption in all previously discussed logic locking techniques' threat model is that the availability of the scan chain is NOT restricted. However, in a notable portion of real ICs/SoCs, this availability is blocked for critical security reasons. Assuming that the scan chain access is already limited/restricted, some logic locking techniques focus on locking the whole circuit, which could be considered as sequential logic locking techniques. Most of them focus on the locking of the state (FSM) of the circuit. In existing FSM locking techniques [13], [14], [90], [91], [92], [93], [94], [95], the original FSM has been targeted and altered in different ways, as demonstrated in Fig. 15: (i) adding few extra sets (modes) of states to the original state transition graph (STG), such as locking/authentication mode states, (ii) adding traps such as black hole, (iii) altering the deepest states of the circuit that makes the timing analysis longer and more complex, (iv) adding shadow states which acts like decoy states, and (v) making the FSM combinational fan-in-cone sub-circuitry key-dependent.

In the primitive FSM-based logic locking techniques, there exists no dedicated port/pin/wiring for key values, and the traversal sequence of these extra added states (sequence of input patterns), like traversal of locking/authentication modes, is the locking/authentication key, and a correct traversal allows the user to reach and traverse the original part of the FSM. Hence, unlike all other logic locking techniques, key inputs are implicitly added, and we can consider this form of logic locking as key-less logic locking. Also, the output generated by the correct traversal of authentication states serves as a watermark. In addition to these groups, a set of studies evaluate FSM locking without adding any extra state. However, the complexity and overhead (area) of this approach is higher compared to other schemes [92]. More recent FSM-based logic locking studies also evaluate the combination of state traversal and key-based logic locking [93], [94], [95]. Table 8 summarizes the main specifications of existing FSM-based logic locking techniques.



(a) HARPOON [13] **(b)** Dynamic State Deflection [91]

(c) Hardware Active Metering [90] **(d)** Interlocking Obfuscation [14]

Fig. 15: FSM Locking Technique.

TABLE 8: Specification of FSM-based Logic Locking Techniques.

Logic Locking	Mechanism	Overhead	Corrupt	Attacked by
HARPOON [13]	Adding extra authentication and obfuscation states before the original initial state	*variant* (w.r.t. size of added states)	High at new extra states	RANE:4.2.2.2, Fun-SAT: 4.2.2.3
FSM Interlocking [14]	Adding traps per each original state for incorrect key or sequence	■■□□□	■■■	Fun-SAT: 4.2.2.3
Active Metering [90]	Adding extra obfuscation states + black holes for specific transitions	*variant* (w.r.t. size of added states)	High at new extra states	Fun-SAT: 4.2.2.3
Dynamic Deflection [91]	Inserting code-word transitions based on input pattern + extra states before initial state	*variant* (w.r.t. size of added states)	High at new extra states	Fun-SAT: 4.2.2.3
FSM reconfig [96]	reconfigurable logic for FSM circuitry (Only on FPGA)	■■■□□	■■■	NONE
DFSSD [93]	Adding faults at deep states using counter + point function	■□□□□	■□□	NONE^{*1}
JANUS/HD [94], [95]	Concealment of State transition circuitry using a configurable control unit	■■□□□	■■■	NONE

^{*1}: It could be vulnerable to structural-based removal attack, if no camouflaging is in place for some locking gates.

3.7 Behavioral Timing-based Locking

Unlike all previous techniques that focus on the functionality of the design, some logic locking techniques went one step further and lock the behavioral properties of the circuit, such as timing. For example, DLL [15] introduces a custom tunable delay and logic gate, demonstrated in Fig. 16(a) which could have different delays based on the value of the key. Since most of the CAD tools are dealing with Boolean nature, it would be hard for the adversary to deal with this form of ambiguity. Some more recent techniques use multi-cycle paths, key-controlled clock handling, and latch-based structure, in which the timing of the circuit would be changed in an asynchronous/clock-gated manner [97], [98], [99], [100], [101]. For instance, in latch-based and asynchronous-based techniques [99], [100], the time of storage of data in the FFs are asynchronized and controlled with the key. More particularly in data flow obfuscation [100], as demonstrated in Fig. 16(c), *key-based c-element* of asynchronous circuits has been used to control the timing/flow of data within the design. So, without having the correct key, the flow of data movement will be changed within the circuit, which consequently corrupts the functionality and may result in appearing some halt in the circuit.

The timing-based and latch-based technique shows promising results against different adversary' actions. However, due to the lack of full EDA (electronic design automation) tool support for asynchronous designs, replacing the flip-flops with latches and implementing asynchronous

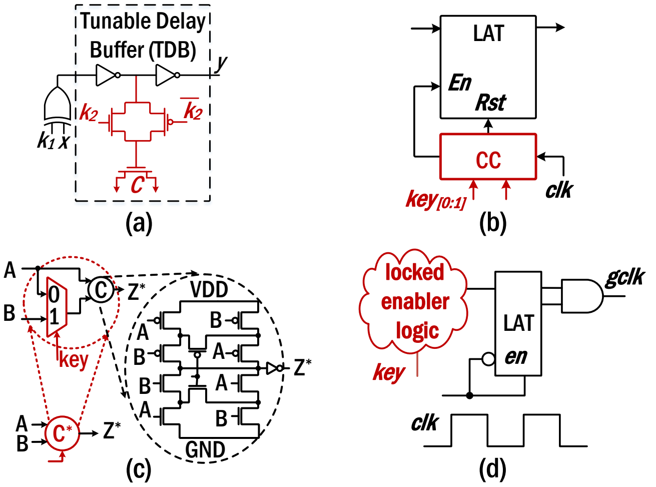


Fig. 16: Behavioral Timing-based Locking: (a) Delay-based Locking [15], (b) Latch-based Locking [99], (c) Asynchronous Locking [100], (d) Clock-gated Locking [101].

TABLE 9: Specification of Timing-based Logic Locking Techniques.

Logic Locking	Mechanism	Overhead	Corrupt	Attacked by
Delay Locking [15]	Insertion of tunable delay buffers that control both function and delay of key gates	■■□□□	■■■	SMT:§4.1.2.6
Latch-based [99]	adding key-based latches + decoy-based latch insertion	■■□□□	■■■	NONE
Data-flow [100]	controlling the data flow using asynchronicity + decoy latches + decoy logic	■■□□□	■■■	NONE
O'Clock [101]	locking the clock gating circuitry + transition-based stripping functionality	■□□□□	■■■	NONE

∗: Dependent to Logic Locking Technique Selected for Functional Locking

latch-based designs raise burdensome challenges in the IC design process, and it makes the usage of asynchronicity almost impractical for complex SoCs. A very recent study, called O'clock, relies on widely-used clock-gating techniques and targets the clock-gating enabling circuitry for obfuscation purposes with full EDA-based support. One of the main features of these timing-based techniques is that they control and manipulate the time of data capturing at storage elements, i.e., at FFs. Hence, the adversary cannot track and follow the exact and correct timing of data capturing, and there is no additional benefit for the design team to restrict/block the scan chain. So, unlike the scan locking or blockage, they can keep scan chain available (imperative to perform in-field debug and test but exploited by SAT attack [45]) to the untrusted foundry and end-users while resisting a wider range of I/O query-based attacks, like SAT-based attacks. Table 9 summarizes the main specifications of existing timing-based logic locking techniques. As demonstrated, except for delay locking that is broken using a theory-based I/O query-based attack, called SMT attack [102], all others are not broken, showing the robustness of this breed of logic locking.

3.7.1 beyond-CMOS and Mixed-Signal Logic Locking

In comparison to CMOS technology, emerging technologies, such as spintronics, memristors, FinFet, CNTFETs, and NWFETs, which are also compatible to be integrated with CMOS technology, promise and provide unique properties that can be engaged for security purposes, features like variability/randomness, run-time reconfigurability or polymorphic behavior. The utilization of these features can help to obtain resilience against reverse engineering, to build unique PUF/TRN generation units, to protect IPs, and to build masking against side-channel leakage [103]. Hence, a set of existing logic locking (and more on camouflaging) techniques have used such technologies. Many of these approaches utilize these technologies to achieve (i) *reconfigurability/dynamicity* that helps to invalidate continuous analysis on the locked circuit (requires frequent restart), (ii) *resiliency against reverse engineering* that helps against diverse acts by the adversary for revealing the secrets, such as probing or retrieving the unlocking key, or retrieving the netlist by the untrusted foundry. The utilization of emerging technologies for IP protection through logic locking can be summarized as follow:

(i) Spintronic-based: With a non-volatile switching mechanism and other related concepts like spin-transfer torque, spin electronics technology can provide both computation and storage/memory capabilities (STT) [104], [105], [106]. All-spin logic (ASL) for camouflaging [107], spintronics-based reconfigurable LUTs [68], [108], [109], [110], fully programmable polymorphism based on giant spin-Hall effect (GSHE) [111], [112] are some approaches that leverage this technology for locking+camouflaging at lower overhead compared to CMOS-based counterparts.

(ii) *Memristor-based*: Memristor-based (memory-resistor) cells are basically able to retain their internal resistive state w.r.t. the voltage/current applied, which can be used for building Boolean logic [113], [114]. In [63], [115], the concept of reconfigurable (polymorphic) cyclic logic locking has been proposed in which the memristor cell(s) has been used to protect against adversaries in the foundries and test facilities.

(iii) *FET-based*: Tunnel field-effect transistors, carbon nanotube field-effect transistors (CNTFET), and Nanowire FETs (NWFETs) are other leading emerging technology candidates to replace CMOS FinFET and DRAM technologies, which are compatible with CMOS technology [116], [117], [118], [119], [120]. Similarly, these technologies could be used for implementing polymorphic gates (PLG) as a means of locking+camouflaging [121], [122], [123], [124]. In [123], silicon nanowire (SiNW) FETs are utilized to implement different PLGs for making the locking part less traceable. The authors of [121] propose silicon NWFETs for camouflaging+locking, by using controlled ambipolarity of NWFETs, helping to build primitives like NAND, NOR, XOR, and XNOR functions.

Apart from beyond-CMOS technologies, some approaches also investigated the applicability and application of logic locking in analog and mixed-signal (AMS) circuits, which is a large subclass of analog ICs, including data converters, phase-locked-loops (PLLs), radio frequency (RF) transceivers, etc. [125], [126], [127], [128], [129], [130], [131]. Many of these approaches consider the locking of digital

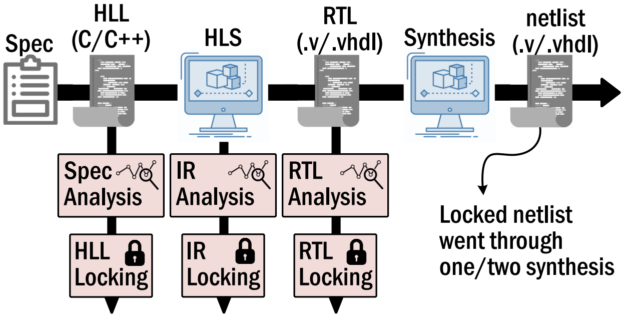


Fig. 17: High Level Logic Locking Technique.

parts that directly or indirectly affects the analog side [125], [126], [131]. However, to expand the locking over both analog as well as digital sections, other approaches target modules like analog-to-digital or digital-to-analog (ADC/DAC) converters [127]. Another study uses chaotic computing and proposes a mixed-signal locking by adopting chaogate [132] to get two important advantages of this mechanism, i.e. (1) Ability to build all 2^{2^2} functions of a 2-input Boolean gate using a single chaotic element (like LUT but at lower overhead), and (2) Dynamic function update capability [130]. At the layout level, few studies also investigate the sizing and hiding as a camouflaging approach by means of fake contacts in the active geometry of the layout components [129].

3.8 High-level Logic Locking

Although a high portion of existing logic locking techniques is implemented at the gate level (or even transistor/layout level), they may be incapable of targeting all semantic information (defined and described in higher level, e.g., RTL or HLS) since logic synthesis and optimizations will convert/simplify/twist much of it into the netlist. Hence, to (1) be able to directly target any algorithmic and semantic of the design for locking purposes, and (2) to get the benefit of synthesis and conversion done by the optimizations for twisting logic locking part into the original part(s), a bunch of recent studies utilizes new methodologies for locking at a higher level of abstractions [56], [133], [134], [135], [136], [137], [138], [139], [140].

Another advantage of high-level logic locking techniques is that they can potentially protect the design against a wider range of untrusted entities. As shown in Fig. 2, with the notion of insider threats (e.g., a rogue employee at the design house), high-level locking allows the design team to have the IP locked and protected from far earlier stages of IC design flow. As demonstrated in Fig. 17, high-level locking can be categorized into the following groups. Also, Table 10 covers the main specification of existing countermeasures in this breed of logic locking.

(i) *High-level Locking before Synthesis/Transition*: Some of the approaches apply the logic locking at the highest level (e.g., C/C++) code before synthesizing the design through the HLS steps [136]. In such cases, the HLL code provided to the HLS tool is already locked, and conversion/absorption/transformation on locked semantic will happen through (i) the HLS intermediate steps, including allocation, scheduling, and binding, as well as (ii) RTL synthesis.

(ii) *High-level Locking+Synthesis (HLS extension)*: Some studies implement and integrate locking with the intermediate steps of the HLS engine [56], [133], [135], [140]. Such approaches analyze intermediate representations (IRs) generated through the HLS flow and will extend these IRs by applying the locking part. In these approaches, the HLL code is not locked, but the RTL generated by the HLS tool is locked, and the locked RTL will face conversion/absorption/transformation through RTL synthesis

(iii) *Register-Transfer Locking (post-HLS or direct RTL)*: Techniques in this sub-group apply logic locking at RTL level, either on the output of the HLS engine, i.e. the RTL generated by the HLS tool [134], [138] or RTL designed/developed directly by the designers [139]. This group analyzes the specification of RTL (like abstract syntax tree) or other graph-based representation to find the best candidates for selection and locking, and then the locked RTL will go through the RTL synthesis.

(iv) *Compound Locking (High level + Gate level)*: We also witness a compound form of logic locking in which RTL locking and gate-level have been engaged as a single countermeasure. In [137], RTL locking is combined with a scan-based logic locking to thwart threats induced by both untrusted foundry and malicious users.

Moving from gate level to higher levels allows the designers to target higher order elements, such as semantic information: (1) *Constants*: Sensitive hard-coded information through the computation (e.g., filter coefficients, encryption IVs, etc.). (2) *Arithmetic Operation*: Critically determining functional arithmetic operation (e.g., multiplications, shifting, adding, subtraction, etc.). (3) *Conditional Branches*: Branches defining the execution flow (based on the control flow graph). (4) *Function Calls*: Important function calls that build the main hierarchy of the design. (5) *Memory Access*: Address, read, and write for the memory blocks with sensitive information. Based on these elements targeted for locking, all high-level logic locking techniques can provide high corruptibility. Table 10 summarizes the specification of existing logic locking at higher level.

3.9 eFPGA-based IP-level Locking

Some recent studies have investigated a coarse-grain form of logic locking, in which redaction by the usage of embedded FPGA (eFPGA) has been done at SoC-level [141], [142], [143], [144]. In this case, after selecting specific (targeted) module(s), they will be replaced with fully reconfigurable soft embedded eFPGA or ready-made eFPGA hard macro. So, the eFPGA, with configurable logic blocks (CLBs) containing look-up tables (LUTs), flip-flops, and routing logic, can be fabricated and programmed to realize the desired functionality. In this case, the secret of the design would be the bitstream that determines the functionality of eFGPA, and the adversary must recover the complete bitstream to implement the correct functionality in each eFPGA.

The eFPGA-based IP redaction can be considered as a superset of LUT/routing-based logic locking, as described in §3.4. Because of the symmetric structure and uniform CLB/LUT distribution, structural attacks are difficult to be

TABLE 10: Specification of High-Level Locking Techniques.

Logic Locking	Type	Mechanism	Overhead	Attacked by
TAO [133]	within HLS	Targeting loops, constants, branches, and manipulation of HLS scheduling	*variant* (w.r.t.) features	SMT: §4.1.2.6
SFLL-HLS [56]	within HLS	point function locking through HLS flow	■□□□□	SPI: §4.1.3.5
ASSURE [134]	at RTL	Targeting branches, constants, and arithmetic operations	■■■□□	SMT: §4.1.2.6
HOST [138]	at RTL	creating RTL with numerous abstraction behavior + building FSM-based SMT-hard instance	■■□□□	NONE
HLock [136]	at HLL	Targeting function calls, control flow, arithmetic operation, constants + ILP optimization	■□□□□	NONE
Fortifying RTL [137]	at RTL & Gate-level	Targeting branches, constants, and arithmetic operations + gate-level scan blockage	■□□□□	NONE

∗: This group is combination of locking and camouflaging techniques.

applied. From the functionality point-of-view, fully configurability and growth of bitstream size result in a significant enhancement of resiliency against I/O query-based attacks. A recent exploration has investigated eFPGA Parameters For IP Redaction, in which it is shown that similar to routing-based locking, coarse-grained eFPGA IP redaction is an SAT-hard instance that cannot be broken using any of the existing I/O query-based attacks. However, compared to LUT/routing-based logic locking, their incurred overhead is even getting worse [144].

4 Logic Locking: Attacks

This section provides a more comprehensive evaluation of the existing successful attacks introduced so far on logic locking. Based on the models and assumptions previously discussed in Section 2.2, all attacks on logic locking could be categorized into different groups as demonstrated in Fig. 18. The categorization of attacks on logic locking is heavily dependent on the availability of the target chip in activated/unlocked mode (oracle). The availability of oracle will help the adversary to build up more algorithmic attacks. Most of the *oracle-guided* attacks could be also known as algorithmic attacks, in each, a systematic flow has been proposed that results in the exposure of either logic locking key or the correct functionality of the locked circuit. Almost all attacks in this category could be considered as *(weak) invasive*, and they require to have access to the netlist of the locked circuit. Based on the applicability, they could be categorized into attacks on *combinational* circuits (when scan access is available) or *sequential* circuits (when scan access is NOT available). However, *oracle-less* attacks, on the other hand, mostly rely on CAD tools such as synthesis tools, or physical attributes of the circuit such as side-channel information to accomplish the attack flow. Based on the structure/mechanism, there exists three main subcategories of *oracle-less* attacks, which is *structural*, *tampering*, and *probing* attacks. Fig. 19 illustrates almost all existing attacks on logic locking and their categorizations. In the following of this survey paper, we thoroughly evaluate the members of each sub-group of this hierarchy, and we will cover the following crucial information per each sub-group:
(1) A uniform assessment on each attack algorithm based on their proposed mechanism.
(2) The main purpose of each attack, the capability as well as applicability of each attack on the existing countermeasures.
(3) The challenges and limitations of each attack in terms of implementation effort/feasibility, applicability, design time/complexity, overhead, etc.
(4) Existing or potential countermeasures that could break each attack.

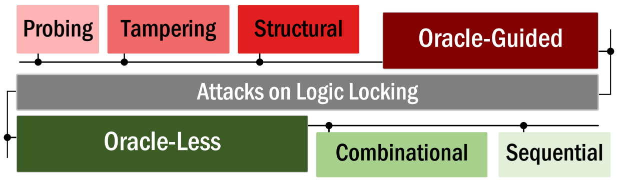


Fig. 18: The General Categorization of the Attacks on Logic Locking.

4.1 Oracle-Guided (OG) on Combinational Circuits

The name of each category used in Fig. 18 represents the attack model used for that category. Here the oracle-guided (OG) attacks on combinational circuits show that the de-obfuscation attacks of this category have been introduced based on the following assumptions:
It is oracle-guided: The attacker requires to have access to one additional activated/unlocked version of the chip (oracle).
It is on combinational circuits: Since almost 100% of real application ICs are sequential, it implies that having access to the DFT structure, i.e. scan chain pins, is available to provide the access to each combinational part (CL) of the circuit.
It is invasive[8]: The attacker requires to have access to the netlist of the locked circuit (locked GDSII at the foundry or reverse-engineered of chip acquired from the field/market).

4.1.1 OG Combinational ATPG-based Attacks

Once the logic locking key is incorrect, function corruption will happen, and since the propagation of corrupted signal(s) resembles fault propagation flow, the possibility of using testability and fault analysis attributes was firstly investigated as a means of attack on logic locking. These de-obfuscation attacks exploit almost the same techniques/algorithms that are widely used for automatic test pattern generation (ATPG), such as exhaustive testing, randomness used with algorithmic methods for testing and debug, symbolic difference check, and path sensitization methods [145].

4.1.1.1 Sensitization Attack: Similar to the logic-level path sensitization in ATPGs with three main steps, i.e. (1) fault sensitization, (2) fault propagation, and (3) line justification, the sensitization attack [12] consists of the same steps. In the sensitization attack, each key bit at any arbitrary gate will be treated as a stuck-at fault. Then, it will be propagated to

[8] Except for one preliminary de-obfuscation attack, called *hill-climbing*, all other oracle-guided de-obfuscation attacks on combinational circuits require access to the netlist.

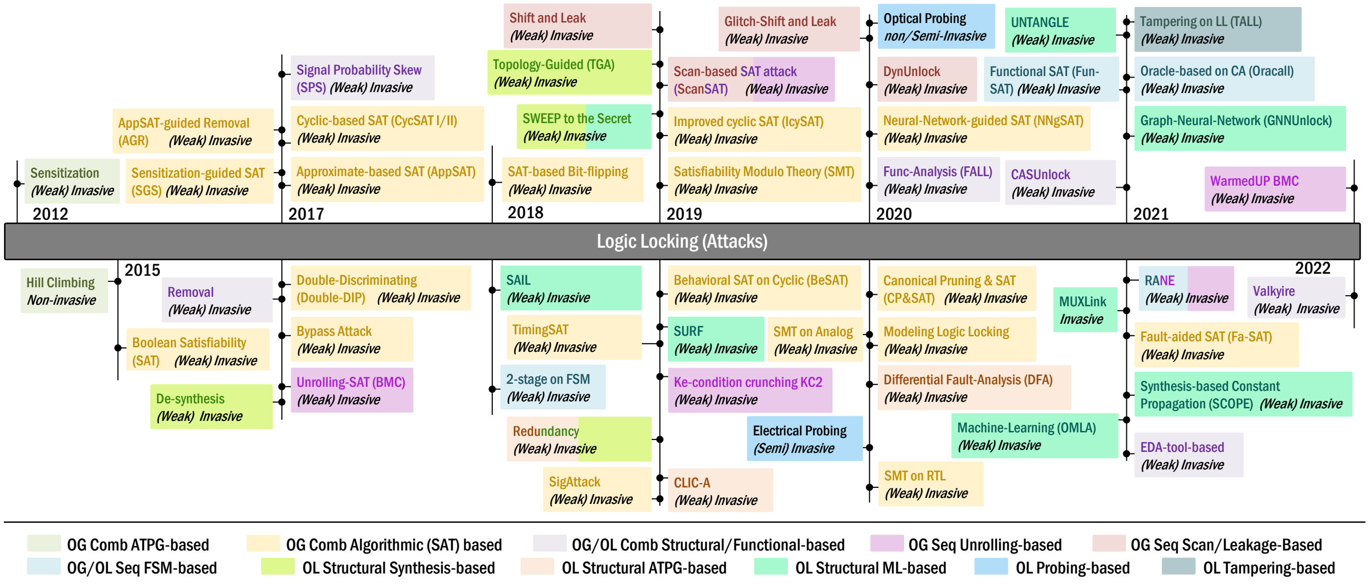

Sensitization (Weak) Invasive
2012
Sensitization-guided SAT (SGS) (Weak) Invasive
AppSAT-guided Removal (AGR) (Weak) Invasive
Signal Probability Skew (SPS) (Weak) Invasive
Cyclic-based SAT (CycSAT I/II) (Weak) Invasive
Approximate-based SAT (AppSAT) (Weak) Invasive
2017
Shift and Leak (Weak) Invasive
Topology-Guided (TGA) (Weak) Invasive
SWEEP to the Secret (Weak) Invasive
SAT-based Bit-flipping (Weak) Invasive
2018
Glitch-Shift and Leak (Weak) Invasive
Scan-based SAT attack (ScanSAT) (Weak) Invasive
Improved cyclic SAT (IcySAT) (Weak) Invasive
Satisfiability Modulo Theory (SMT) (Weak) Invasive
2019
Optical Probing non/Semi-Invasive
UNTANGLE (Weak) Invasive
DynUnlock (Weak) Invasive
Functional SAT (Fun-SAT) (Weak) Invasive
Neural-Network-guided SAT (NNgSAT) (Weak) Invasive
Func-Analysis (FALL) (Weak) Invasive
CASUnlock (Weak) Invasive
2020
Tampering on LL (TALL) (Weak) Invasive
Oracle-based on CA (Oracall) (Weak) Invasive
Graph-Neural-Network (GNNUnlock) (Weak) Invasive
WarmedUP BMC (Weak) Invasive
2021
Logic Locking (Attacks)
Hill Climbing Non-invasive
2015
Removal (Weak) Invasive
Boolean Satisfiability (SAT) (Weak) Invasive
De-synthesis (Weak) Invasive
Double-Discriminating (Double-DIP) (Weak) Invasive
Bypass Attack (Weak) Invasive
Unrolling-SAT (BMC) (Weak) Invasive
SAIL (Weak) Invasive
TimingSAT (Weak) Invasive
2-stage on FSM (Weak) Invasive
Redundancy (Weak) Invasive
SigAttack (Weak) Invasive
Behavioral SAT on Cyclic (BeSAT) (Weak) Invasive
SURF (Weak) Invasive
SMT on Analog (Weak) Invasive
Ke-condition crunching KC2 (Weak) Invasive
Electrical Probing (Semi) Invasive
CLIC-A (Weak) Invasive
Canonical Pruning & SAT (CP&SAT) (Weak) Invasive
MUXLink Invasive
Modeling Logic Locking (Weak) Invasive
Differential Fault-Analysis (DFA) (Weak) Invasive
Machine-Learning (OMLA) (Weak) Invasive
SMT on RTL (Weak) Invasive
2022
RANE (Weak) Invasive
Valkyire (Weak) Invasive
Fault-aided SAT (Fa-SAT) (Weak) Invasive
Synthesis-based Constant Propagation (SCOPE) (Weak) Invasive
EDA-tool-based (Weak) Invasive
OG Comb ATPG-based
OG Comb Algorithmic (SAT) based
OG/OL Comb Structural/Functional-based
OG Seq Unrolling-based
OG Seq Scan/Leakage-Based
OG/OL Seq FSM-based
OL Structural Synthesis-based
OL Structural ATPG-based
OL Structural ML-based
OL Probing-based
OL Tampering-based

Fig. 19: Attacks on Logic Locking Techniques.

TABLE 11: Classification of Key Gates in Sensitization Attack [12].

Term	Description	Strategy used by attacker
Runs of KGs	Back-to-Back KGs	Replacing by a Single KG
Isolated KGs	No Path between KGs	Finding Unique Pattern per KG (Golden Pattern (GP))
Dominating KGs	$k1$ is on Every Path between $k0$ and POs	Muting $k0$, Sensitizing $k1$
Concurrently Mutable Convergent KGs	Convergent at a Third Gate, Both can be Propagated to POs	Muting $k0/k1$, Sensitizing $k1/k0$
Sequentially Mutable Convergent KGs	Convergent at a Third Gate, One can be Propagated to POs	Determining $k1$ by GP, Update the Netlist, Target $k0$
Non-Mutable Convergent KGs	Convergent at a Third Gate, None can be Propagated to POs	Brute Force Attack

the primary/scan output (PO/SO) using (1) fault sensitization and (2) fault propagation. After determining the input pattern (state) that propagates the value of the key to the PO/SO, the attacker applies the same input pattern (state) to the oracle and since the correct key value is already loaded in the oracle, the correct key value will be propagated to the PO/SO. Then, the attacker could observe and record this output as the value of the sensitized key.

The main purpose of the sensitization attack is to break RLL [11] as a member of primitive logic locking solutions described at §3.1. The attack results on RLL show that the sensitization attack could determine individual key values of the RLL-locked circuit in a time linear with respect to the size of the key. It should be noted that, in the sensitization attack, the propagation of a key bit to the PO/SO is heavily dependent on the location of the key. Hence, they classify key gates based on their location and discuss corresponding attack strategies for each case. The summary of strategies and techniques used in the sensitization attack is reflected in Table 11.

There exist some limitations/challenges in the sensitization attack: (1) Similar to path sensitization in ATPGs, it is only applicable for acyclic combinational circuits. In the presence of feedback, it faces an infinite loop. (2) The efficiency of this attack would be low if the key gates are located at non-synthesizable points. Thus, in SLL, as another member of primitive logic locking solutions [12], non-synthesizable points have been widely exploited, to introduce a countermeasure against this attack.

4.1.1.2 Random-based Hill-Climbing Attack: In [146], another ATPG-based attack has been introduced that finds specific test patterns to apply them to the locked circuit and by observing the responses, it can lead the preliminary guessed (random) key values to the correct ones. Unlike the sensitization attack [12] and all other OG attacks on combinational logic locking, the hill-climbing attack does not require netlist access and could be considered as a *non-invasive* attack. It uses a randomized local key-searching algorithm to search the key that can satisfy a subset of correct input/output patterns. It first selects a random key value and then at each iteration, the key bits that are selected randomly, are toggled one by one. The target is to minimize the frequency of differences between the observed and expected responses. Hence, a random key candidate is gradually improved based on the observed test responses. When there is no solution at one iteration, the algorithm resets the key to a new random key value. The main purpose of this attack is to break the very first logic locking technique, i.e. RLL [11]. However, in many cases, it faces a very long execution time with no results. This happens for two main reasons which significantly undermine the success rate of this attack: (1) The key will be initiated randomly, and (2) The complexity of the attack will be increased drastically, particularly when the key size is large or the key bits are correlated.

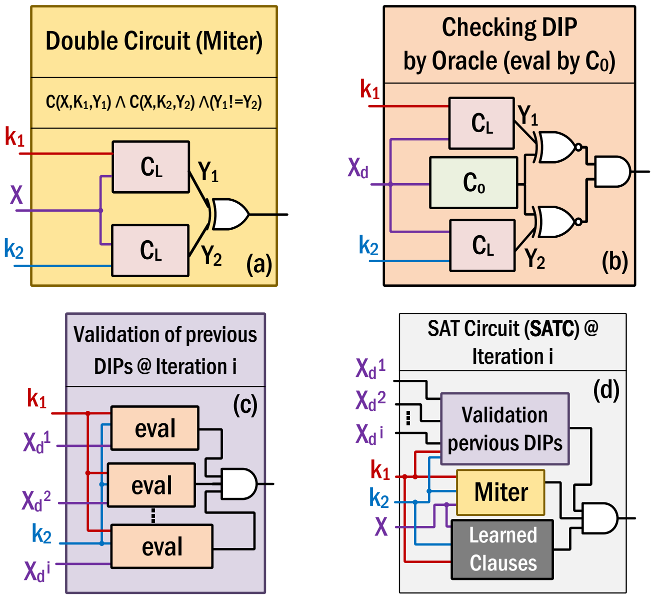


Fig. 20: The SAT Attack Iterative Flow [45], [46]: (a) building miter (double circuit), (b) finding DIP + eval matching with the Oracle, (c) Validating DIPs at Iteration i for all previously found DIPs, (d) SAT circuit (SATC) at iteration i.

4.1.2 OG Combinational Algorithmic (SAT)-based Attacks

Solving a Boolean *satisfiability* problem is the process of finding the satisfying assignment for a Boolean expression or equation. Many ATPG tools use Boolean *satisfiability* solver, SAT solver, for generating test patterns, and currently, it is one of the fastest ATPG algorithms, particularly for large-size circuits. In 2015, Subramanyan *et al.* [45] propose a new and powerful attack on logic locking that still gets the benefit of the SAT solver, which is widely used in ATPGs. The engagement of the SAT solver as a means for attacking the locked circuits has got the most attention in recent years for some important reasons, such as the strength, the performance of the attack, and the scalability. As shown in Fig. 19, the SAT-based attacks on logic locking, either combinational or sequential, are the largest by count. In this section, we will review the SAT-based attacks that focus on combinational locked netlist.

4.1.2.1 Conventional SAT Attack: The SAT attack was first introduced by Subramanyan *et al.* [45]. At the same time, El Massad *et al.* proposed the same technique, in which a SAT solver is engaged for attacking the combination logic locked netlist [46]. Getting inspired by the miter (distinguisher) circuit that is widely used for formal verification, The SAT attack uses a specific duplication mechanism to break the logic locked netlists. The main steps of the SAT attack have been demonstrated in Fig. 20.

As shown in Fig. 20(a), in the SAT attack, the attacker first duplicates the locked circuit and builds a double circuit (miter). The miter is used for finding an input (x) that for two different key values, i.e. $\mathtt{k}_1$ and $\mathtt{k}_2$, this input generates two different outputs ($\mathtt{y}_1$ and $\mathtt{y}_2$ is XORed). The key values ($\mathtt{k}_1$ and $\mathtt{k}_2$) and the primary input pattern x, will be found by a SAT solver query. Such input is referred to as the discriminating/distinguishing input pattern (DIP). Each DIP ($\mathtt{x}_d\mathtt{[i]}$) will be checked by the oracle (*eval* by applying $\mathtt{x}_d\mathtt{[i]}$ to $\mathtt{c}_o$), as shown in Fig. 20(b), assuring that for a previously found DIP, two different keys generate the same output value (part of the correct key pool for current iteration). Each iteration of the SAT attack finds a new DIP and each adds a new *eval* check to the whole problem. All *eval* checks are then *AND*ed together, as shown in Fig. 20(c), expanding the constraint to all previously found DIPs. In each iteration, the SAT solver tries to find a new DIP and two key values that satisfy the miter and the all aggregated (ANDed) *eval* constraints as shown in Fig. 20(d). This iterative process continues until the SAT solver cannot find a new DIP. At this point, any key that generates the correct output for the set of previously found $\mathtt{x}_d$s is the correct key. Algorithm 1 provides an algorithmic representation of the SAT attack and its iterative structure for finding all DIPs.

Algorithm 1 SAT-based Attack Algorithm [45]

```
function SAT_ATTACK(Circuit $C_L$, Circuit $C_O$)
    $i \leftarrow 0$;
    $F_0 \leftarrow C_L(x, k_1, y_1) \wedge C_L(x, k_2, y_2)$;
    while $SAT(F_i \wedge (y_1 \neq y_2))$ do
        $x_d[i] \leftarrow$ sat_assign($F_i \wedge (y_1 \neq y_2)$);
        $y_d[i] \leftarrow C_O(x_d[i])$;
        $F_{i+1} \leftarrow F_i \wedge C_L(x_d[i], k_1, y_d[i]) \wedge C_L(x_d[i], k_2, y_d[i])$;
        $i \leftarrow i+1$;
    $k_c \leftarrow$ sat_assign($F_i$);
```

The main purpose of the SAT attack was to break the primitive logic obfuscation techniques, including RLL [11], SLL [12], and FLL [44], as described in §3.1. For all these locking techniques, the SAT attack was able to rule out a significant number of key values at each iteration (by finding each DIP), and it was able to break them within a few minutes.

The traditional SAT attack has received significant attention in recent years, and numerous studies demonstrated the limitations/challenges of this powerful attack. Point function techniques (§3.2) show how the strength of DIPs could be reduced to minimize its pruning power. Cyclic locking techniques (§3.3) show the weakness of the SAT attack while combinational cycles are engaged as a means of logic locking. Routing-based locking techniques (§3.4) and eFPGA IP-level redaction techniques (§3.9) show the effectiveness of complex structures as some fully configurable universal models that could be resilient against the SAT attack and its derivatives. Behavioral timing-based techniques (§3.7), such as delay-based logic locking, show how non-Boolean logic locking techniques cannot be modeled using the SAT attack. Scan-based (§3.5) and FSM-based (§3.6) countermeasures also demonstrate how the limiting access to the DFT structure and sequence-based locking can be engaged to break the SAT attack. Also, other behavioral timing-based techniques, like latch-based and clock-gating locking (§3.7), show how capturing time of scan chain cells can be manipulated that conceal exact read/write of the scan chain for applying the SAT attack. However, since the introduction of the SAT attack, we witness a cat-and-mouse game in this domain, where each defense (attack) is trying to break another attack (defense) by revealing their vulnerabilities.

4.1.2.2 Approximate-based SAT Attack: Point function logic locking techniques are the first group of countermeasures that are resilient against the SAT attack. As shown in

Fig. 8, these techniques will corrupt only a few (e.g. one) PO when the key is incorrect, and the output corruptibility happens for only a limited number of input patterns. Hence, for a limited number of POs (e.g. one PO), the circuit works as the original circuit, and for a rare set of input patterns, the output would be corrupted at rare POs. However, for a wide range of applications, such as image processing engines, this quite low output correction could be ignored at outputs (e.g. missing a few pixels). Hence, unlike the traditional SAT attacks, in which the exactness of the extracted key is guaranteed, in [147], [148], AppSAT and Double-DIP have been proposed, respectively, in which by relaxing this constraint, it is demonstrated that finding an approximated key values could minimize the error rate when the circuit is locked using point function techniques.

The overall flow of the AppSAT is demonstrated in Algorithm 2, which is implemented based on the SAT attack and random testing. AppSAT [147] uses the *probably-approximate-correct* (PAC) model for formulating approximate learning problems. Unlike the traditional SAT attack whose termination condition is when no more DIPs could be found by the SAT solver, the AppSAT termination condition is based on the output error rate of a set of stimuli. AppSAT will be ended in any early step in which the error falls below a certain limit for a randomly selected set of stimuli. If this condition happens, the key value that satisfies the current set of constraints will be recognized as an approximated key with a specified error rate. Some heuristic methods are used in AppSAT for (1) estimating the error of large functions, to avoid any computation complexity, and (2) finding the minimal set of DIPs that lead to the targeted approximated key with the satisfied error. These methods play an important role in terms of the performance of the AppSAT attack.

AppSAT works perfectly fine on the logic locking techniques with low output corruption. Hence, some studies evaluate the robustness of point function techniques when they are combined with the primitive techniques, called compound techniques (§3.2.1). The compound logic locking techniques provide the most benefit of both categories, i.e. the high output corruption achieved by the primitive techniques, as well as the SAT resiliency achieved by the point function techniques. In such cases, AppSAT guarantees that the key of the primitive techniques will be extracted correctly, and the key related to the point function techniques is approximated that meets the error rate requirement.

Double-DIP [148] is an extension of the AppSAT attack in which during each iteration, the discriminating input should eliminate at least two sets of wrong keys. The reason for finding two incorrect sets of keys is for distinguishing between the keys that corresponded to the primitive technique from that of the point function technique. Since point function techniques corrupt the POs for a few input patterns (e.g. *ONE*) when an incorrect key corrupts the POs for more than one set of keys, it corresponds to the primitive logic locking. Otherwise, it corresponds to the point function technique. The overall flow of the Double-DIP attack has been illustrated in Algorithm 3.

Algorithm 2 AppSAT Attack Algorithm [147]

```
function APPSAT_ATTACK(Circuit $C_L$, Circuit $C_O$)
    $i \leftarrow 0$;
    $F_0 \leftarrow C_L(x, k_1, y_1) \wedge C_L(x, k_2, y_2)$;
    while $SAT(F_i \wedge (y_1 \neq y_2))$ do
        $x_d[i] \leftarrow$ sat_assign($F_i \wedge (y_1 \neq y_2)$);
        $y_d[i] \leftarrow C_O(x_d[i])$;
        $F_{i+1} \leftarrow F_i \wedge C_L(x_d[i], k_1, y_d[i]) \wedge C_L(x_d[i], k_2, y_d[i])$;
        $i \leftarrow i+1$;
        every $n$ rounds do
        for each ($x \in$ Random Patterns) do
            if $C_L(x, k_1, y) \neq C_O(x)$ then
                $FailedPatterns \leftarrow FailedPatterns + 1$;
                $F_{i+1} \leftarrow F_{i+1} \wedge (C_L(x, k_1, y) = C_O(x))$;
                $i \leftarrow i+1$;
        if error ¡ ErrorThreshold then
            return $k_1$ as an approximate key
    $k_c \leftarrow$ sat_assign($F_i$);
```

The effectiveness of the Double-DIP attack has been illustrated on the SARLock+SSL, which represents an attack on a compound of point function and primitive techniques. The output result of the Double-DIP attack is an approximated key, in which the key of the primitive technique (SSL) is guaranteed to be correct, and the key of the point function technique (SARLock) is an approximated key that meets the error rate requirement.

In general, considering the exactness of the attack, and assuming that compound logic locking is in place, since approximate-based attacks only guarantee the key corresponding to the primitive logic locking, these attacks could reduce the problem from the compound technique to a single point function technique with an approximated key.

Algorithm 3 Double-DIP Attack Algorithm [148]

```
function DOUBLEDIP_ATTACK(Circuit $C_L$, Circuit $C_O$)
    $i \leftarrow 0$;
    $F_0 \leftarrow C_L(x, k_1, y_1) \wedge C_L(x, k_2, y_2) \wedge C_L(x, k_3, y_1) \wedge C_L(x, k_4, y_2)$ ;
    while $SAT(F_i \wedge (y_1 \neq y_2)) \wedge (k_1 \neq k_3)) \wedge (k_2 \neq k_4))$ do
        $x_d[i] \leftarrow$ sat_assign($F_i \wedge (y_1 \neq y_2)) \wedge (k_1 \neq k_3)) \wedge (k_2 \neq k_4)$);
        $y_d[i] \leftarrow C_O(x_d[i])$;
        $F_{i+1} \leftarrow F_i \bigwedge_{j=1}^{4} C_L(x_d[i], k_j, y_d[i])$;
        $i \leftarrow i+1$;
    $k_c \leftarrow$ sat_assign($F_i$);
```

4.1.2.3 Bypass Attack: The bypass attack [149] is another attack that gets the benefit of low corruptibility in point function techniques when they are not mixed with the primitive logic locking techniques. The bypass attack instantiates two copies of the obfuscated netlist using two randomly selected keys and builds a miter circuit that evaluates to 1 only when the output of two circuits is different. The miter circuit is then fed to a SAT solver looking for such inputs. The SAT returns with a minimum of two inputs for which the outputs are different. These input patterns are tested using an activated IC (oracle) validating the correct output. Then, a bypass circuit is constructed using a comparator that is stitched to the primary output of the netlist which is unlocked using the selected random key, to retrieve the correct functionality if that input pattern is applied. The Bypass attack works well when the SAT-hard solution is not mixed with the traditional logic locking mechanism since its overhead increases very quickly as output corruption of logic locking increases.

4.1.2.4 Other Attacks on Compound Logic Locking: Due to the robustness of compound logic locking techniques, they received significant attention for a short time after

Algorithm 4 Bit-flipping Attack Algorithm [150]

```
function BITFLIPPING_ATTACK(Circuit $C_L$, Circuit $C_O$)
    for each $j$ < Fixed-iteration do
        $k_A \leftarrow$ a random key;
        for each bit b $\in k_A$ do
            $k_B \leftarrow k_A$ while bit b flipped;
            $i \leftarrow 0$; $F_0 \leftarrow C_L(x, k_A, y_A) \wedge C_L(x, k_B, y_B)$;
            while $SAT(F_i \wedge (y_A \neq y_B))$ do
                $x_d[i] \leftarrow$ sat_assign($F_i \wedge (y_A \neq y_B)$);
                $F_{i+1} \leftarrow F_i \wedge (x \neq x_d[i])$;
                $i \leftarrow i+1$;
                if HD > Threshold then
                    b is in $k_1$,
                    break;
        $j \leftarrow j + 1$;
    $k_2 \leftarrow$ all key bits / $k_1$;                ▷ Separation is Done. Fixing $k_2$.
    $k_1 \leftarrow$ SAT_ATTACK ($C_L$, $C_O$);   ▷ Find Primitive Keys by SAT.
    $C_L^* \leftarrow$ update_netlist($C_L$ — $k_1$)
    return (BYPASS_ATTACK($C_L^*$);
```

their introduction. Similar to the Double-DIP attack, many studies try to reveal the security vulnerabilities of the compound logic locking techniques. In [150], the bit-flipping attack has been introduced which relies on the fact that keys corresponding to primitive technique could be separated from the keys of point function technique. This could be achieved based on the hamming distances of outputs of the double circuit. The bit-flipping attack is motivated by the output corruption observation, which shows in primitive techniques, an incorrect key causes substantial corruption at POs. However, the output corruption of point function techniques is very small (e.g. with hamming distance equals with *ONE*). Hence, the calculation of the hamming distance (HD) of the double circuit's output could help to distinguish between two sets of keys. The overall structure of the bit-flipping attack has been described in Algorithm 4. First, for each key bit, some DIPs will be found by the SAT solver, and by checking the HD at the double circuit outputs, the usage (primitive or point function) of the key could be determined. Then, after the separation of the keys into two groups, the bit-flipping attack fixes point function keys, as a random number, and uses a SAT solver to find the correct key values used for the primitive technique. After finding keys of the primitive technique, similar to Double-DIP, the problem from the compound technique is reduced to a single point function technique that could be evaluated using approximate-based attacks for an approximated key, or as described in §4.1.2.3, bypass attack can recover the correct functionality using a bypass circuitry.

AppSAT guided removal (AGR) attack is another attack that targets compound logic locking techniques, particularly Anti-SAT as the part of point function [151]. As its name implies, the AGR attack merges the AppSAT attack with a simple structural analysis on the locked netlist (as a post-processing step). However, unlike AppSAT, the AGR attack recovers the exact correct key. In the AGR attack, the AppSAT is first used to find the approximated key, in which the correctness of the primitive technique keys is guaranteed. Then, AGR targets the remaining key bits that belong to the point function logic locking, such as the Anti-SAT block, through a simple structural analysis.

As demonstrated in Fig. 8, one of the main weaknesses of the point function techniques, when they are neither combined with the primitive techniques nor stripped, is that the locking part of the circuit, i.e. flipping and masking circuitry, is fully decoupled from the original parts of the circuit. Hence, a simple structural analysis can recognize the added macros and gates for locking purposes. This is what is called the possibility of removal attack on logic locking discussed in §4.1.3.1. In AGR, as shown in Algorithm 5, in its post-processing steps, AGR finds the gate (G) at which the keys corresponded to point function are converged (decoupled point function locking part), located at the transitive fanout of these keys. AGR identifies the candidates for gate G by using structural analysis for all gates in the circuit and then sort these candidates based on the number of key inputs that converge at a gate and pick the gate G from all candidates, which has the most number of key inputs converge to that gate. Then since the G gate would be the output of the flipping/masking circuitry in the point function techniques, it should have no impact on the functionality when the key is correct. So, the attacker re-synthesizes the design with the constant value for the output of G gate (removing all fan-in cone) and retrieves the correct functionality.

Fault-aided SAT-based attack [152] is another approach working on a specific compound logic locking in which a routing-based locking is combined with a point-function technique [57]. In this attack, faults will be inserted at some fault locations in the locked netlist, and then by applying the SAT attack with a tight timeout, the SAT solver helps to find the key while the fault insertion led to a SAT problem size reduction.

4.1.2.5 SAT-based Attacks on Cyclic Locking: In §3.3, we showed the complexity/challenges behind the usage of combinational cycles in the circuit as a means of logic locking [58], [60], [61], [62], [63], [64], [65]. SAT attack input must be in directed acyclic graph (DAG) format. Hence, adding cycles will violate this assumption. The incapability of the SAT attack for dealing with combinational cycles is the main motivation behind these techniques, which results in breaking the SAT attack either by (1) trapping it in an infinite loop, or (2) forcing it to exit with a wrong key depending on whether the introduced cycles make the circuit stateful or oscillating.

Numerous studies have evaluated the resiliency of these techniques, thereby introducing new attacks on cyclic logic locking [153], [154], [155]. In CycSAT [153], the key combinations that result in the formation of cycles are found in a pre-processing step. These conditions are then translated into problem augmenting CNF formulas, denoted as cycle avoidance clauses, the satisfaction of which guarantees no cycle

Algorithm 5 AGR Attack Algorithm [151]

```
function AGR_ATTACK(Circuit $C_L$, Circuit $C_O$)
    #Cand ← num_gates($C_L$)
    while (#Cand ¿ 1 and !Timeout) do
        AppSAT_Attack();                                   ▷ 4 times
        Candidates ← {};
        for each gate $\in C_L$ do
            if $gate_i$ has the selected property then
                Candidates ← Candidates + 1;
    $G \leftarrow$ Find_Max_key_count(Candidates);
    $C_{Lock} \leftarrow$ remove_TFI($C_L$, $G$);  ▷ remove Transitive FanIn of $G$
    return $C_{Lock}$;       ▷ $C_{Lock}$: $C_L$ after removing Anti_SAT block
```

Algorithm 6 CycSAT Attack on Cyclic Locked Circuits [58]

```
function CYCSAT_ATTACK(Circuit $C_L$, Circuit $C_O$)
    $W = (w_0, w_1, ...w_m) \leftarrow$ FindFeedback($C_L$);
    for each $(w_i \in W)$ do
        $F(w_i, w_i') \leftarrow$ no_ structural_ path($w_i$);
    $i \leftarrow 0$; $NC$(k)=$\wedge_{i=0}^{m} F(w_i, w_i')$
    $C_L^*$(x, k, y) $\leftarrow C_L$(x, k, y) $\wedge$ $NC$(k);
    $F_0 \leftarrow C_L^*$(x, $k_1$, $y_1$) $\wedge$ $C_L^*$(x, $k_2$, $y_2$);
    while $SAT(F_i \wedge (y_1 \neq y_2))$ do
        $x_d$[i] $\leftarrow$ sat_assign($F_i \wedge (y_1 \neq y_2)$);
        $y_d$[i] $\leftarrow C_O$($x_d$[i]);
        $F_{i+1} \leftarrow F_i \wedge C_L$($x_d$[i], $K_1$, $y_d$[i]) $\wedge$ $C_L$($x_d$[i], $k_2$, $y_d$[i]);
        $i \leftarrow i+1$;
    $k^* \leftarrow$ sat_assign($F_i$);
```

in the netlist. The cycle avoidance clauses are then added to the original SAT circuit CNF and the SAT attack is invoked. CycSAT has been introduced in two variants: (i) CycSAT-I, as shown in Algorithm 6, for the cases whose original circuit is acyclic, and (ii) CycSAT-II on a weaker assumption that the original circuit might be cyclic. However, the complexity that CycSAT-II might deal with is much higher making this variant less effective. Also, since all cyclic-based operations are based on structural analysis on circuits, it still faces exponential complexity when the number of cycles grows exponentially w.r.t. the number of feedbacks.

The lack of scalability to assess all cycles in the pre-possessing of the CycSAT attack [153] leads to building incorrect pre-processed circuits, which might keep different types of the cycle even after pre-processing. It results in trapping the SAT attack invoked after the pre-processing step. Hence, the more advanced version of cyclic-based SAT attacks was introduced [154], [155] to remedy such shortcoming(s). In BeSAT [154], with a run-time behavioral analysis at each iteration, as demonstrated in Algorithm 7, it detects repeated DIPs when the SAT is trapped in an infinite loop. Also, when SAT solver cannot find any new DIP, a ternary-based SAT is used to verify the returned key as a correct one, preventing the SAT from exiting with an invalid key. icySAT [155] also can produce non-cyclic conditions in polynomial time w.r.t. the size of the circuit, avoiding the potentially exponential runtime still witnessed in BeSAT. Also, icySAT improves the attacks on cyclic logic locking techniques for cases whether the original circuit is cyclic if the feedback dependencies are re-convergent, or whenever the types of the cycles are oscillating.

4.1.2.6 Theory-based SAT Attack: Since the SAT attack receives the input in conjunctive normal form (CNF), it (or any of its derivatives as described previously) works perfectly fine if the logic locking is of Boolean nature. Hence, a set of recent studies lock the properties of the circuit, which cannot be translated to CNF, such as timing-based or non-CMOS or mixed-signal logic locking as all demonstrated in §3.7.

Azar *et al.* opens a new attack category on behavioral logic locking, which relies on the capability of modeling different theories in satisfiability modulo theory (SMT) solver [102]. The SMT solvers, as the superset of the SAT solvers, can combine the SAT solver with one or more theory solvers, and theory solvers, based on their specification, with richer language sets, can model beyond Boolean nature as well. As a case study, to demonstrate the benefit of the SMT solver as a richer means of attack on logic locking, a graph theory solver has been engaged in the SMT attack, showing how delay-based logic locking could be formulated and broken. Based on the structure of SMT solvers, the attack shows how theory solvers could be engaged as a pre-processor (eager approach) or as a co-processor (lazy approach) to assess and break the problem. One mode known as the *lazy* mode of this attack is illustrated in Algorithm 8. As demonstrated, function *GenTCE* is used and invoked to generate the timing constraints (key-based hold/setup constraints) based on the graph theory solver.

Although the SMT attack demonstrates a general model for attacking behavioral logic locking techniques, Chakraborty *et al.* also proposes TimingSAT that shows a mechanism for modeling delay-based logic locking [156] using the pure SAT solvers. TimingSAT demonstrates the possibility of converting a non-Boolean logic locking problem into its Boolean counterpart. Hence, it consists of two main steps: (1) using a pre-processing mechanism for circuit unrolling approach that helps to capture the timing information in the form of Boolean functions. (2) Then, the locked circuit with captured timing information in Boolean form could be solved using the SAT solver. TimingSAT could be categorized as a simplified formulation of eager approach in SMT, in which with no theory solver, the capture of timing has been accomplished with more challenges using a Boolean-based remapped representation.

The usage of a few theory solvers is investigated in the SMT attack on another side to show the extensibility and richness of the SMT attack on some logic locking techniques, such as compound and cyclic logic locking. Also, further studies on SMT show its capability on other breeds of logic locking as well [157], [158]. In [157], a new SMT attack has been introduced on analog circuits, in which based on SMT formulations, it takes polynomial time (irrespective of the key size) for breaking the analog locking, and it is applicable to the ubiquitous presence in wireless communication networks: Gm-C BPF, LC oscillator, quadrature oscillator, and class-D amplifier, as well as a memristor-based protection technique. Similarly, [158] utilizes SMT-based formulation

Algorithm 7 BeSAT Attack on Cyclic Locked Circuits [154]

```
function BESAT_ATTACK(Circuit $C_L$, Circuit $C_O$)
    $W = (w_0, w_1, ...w_m) \leftarrow$ FindFeedback($C_L$);
    for each $(w_i \in W)$ do
        $F(w_i, w_i') \leftarrow$ no_ structural_ path($w_i$);
    $i \leftarrow 0$;
    $NC$(k)=$\wedge_{i=0}^{m} F(w_i, w_i')$
    $C_L^*$(x, k, y) $\leftarrow C_L$(x, k, y) $\wedge$ $NC$(k);
    $F_0 \leftarrow C_L^*$(x $k_1$, $y_1$) $\wedge$ $C_L^*$(x, $k_2$, $y_2$);
    while $SAT(F_i \wedge (y_1 \neq y_2))$ do
        $x_d$[i] $\leftarrow$ sat_assign($F_i \wedge (y_1 \neq y_2)$);
        $y_d$[i] $\leftarrow C_O$($x_d$[i]);
        $F_{i+1} \leftarrow F_i \wedge C_L$($x_d$[i], $k_1$, $y_d$[i]) $\wedge$ $C_L$($x_d$[i], $k_2$, $y_d$[i]);
        if ($x_d$[i] in DIP) and ($C_L$($x_d$[i], $k_1$) $\neq$ $y_d$[i])) then
            $F_{i+1} \leftarrow F_{i+1} \wedge (k_1 \neq \hat{k}_1) \wedge (k_2 \neq \hat{k}_1)$;
        else if ($x_d$[i] in DIP) and ($C_L$($x_d$[i], $k_2$) $\neq$ $y_d$[i]) then
            $F_{i+1} \leftarrow F_{i+1} \wedge (k_1 \neq \hat{k}_2) \wedge (k_2 \neq \hat{k}_2)$;
        $i \leftarrow i+1$;
    while $SAT_{k_1}$($F_i$) do                                  ▷ Correct Key: $\hat{k}_c$
        if Ternary_SAT($F_i$, $k_c$) then
            $F_i \leftarrow F_i \wedge (k_1 \neq \hat{k}_c)$
        else
            $k^* \leftarrow \hat{k}_c$;
            break;
```

Algorithm 8 SMT Attack on DLL (Lazy Approach) [102]

```
function SMTLazy_Attack(Circuit $C_L$, Circuit $C_O$)
    $C^*_L \leftarrow$ toBOOLEAN($C_L$);                          ▷ Replace TDK with Buffer
    $i \leftarrow 0$; $F \leftarrow C^*_L(x, k_1, y_1) \wedge C^*_L(x, k_2, y_2)$;
    $G^*_L \leftarrow$ toGRAPH($C_L$);                      ▷ Wires = Edges, Gates = Vertices
    $F_T \leftarrow$ GenTCE($G^*_L$)                                ▷ Theory Learned Clauses
    $F_{SMT} \leftarrow F \wedge F_T$;                                         ▷ SMT Clauses
    while $SMT(F_{SMT})$ do                                   ▷ $x_d[i], k_1, k_2$, CC
        $y_d[i] \leftarrow C_O(x_d[i])$;
        $F \leftarrow F \wedge C^*_L(x_d[i], k_1, y_d[i]) \wedge C^*_L(x_d[i], k_2, y_d[i])$;
        $F_{SMT} \leftarrow F \wedge CC$; $i \leftarrow i+1$;
    $k^* \leftarrow$ smt_assign($F_{SMT}$);
```

```
    function GenTCE(Graph $G^*_L$)
        Inputs $\leftarrow$ find_start_points($G^*_L$);
        Outputs $\leftarrow$ find_end_points($G^*_L$);
        $T_{CE}(k) \leftarrow []$;
        for each ((Sp, Ep) $\in$ (Inputs, outputs)) do
            Upper(Sp,Ep)(k) $\leftarrow$ !(distance_leq(Sp, Ep, $t_{cd}$));          ▷ Hold
            Lower(Sp,Ep)(k) $\leftarrow$ distance_leq(Sp, Ep, $t_p$);            ▷ Setup
            R(Sp,Ep)(k) $\leftarrow$ Lower(Sp,Ep)(k) $\wedge$ Upper(Sp,Ep)(k);
            $T_{CE}(k) \leftarrow T_{CE}(k) \cup$ R(Sp,Ep)(k);
        return $T_{CE}(k)$
```

and algorithm to break RTL-level logic locking. The algorithm models an RTL design as an RTL finite state machine with datapath (RTL-FSMD), and then it is abstracted out the details of the hardware into a behavioral program on which SMT solving has been invoked.

4.1.2.7 SAT-based Attacks on LUT/Routing Locking:

LUT-based and specifically routing-based locking, as described in §3.4, significantly increase the complexity of inner calculations of the SAT solver leading to an extremely long runtime per *each iteration* of the SAT attack [76], [77], [78]. These techniques rely on building symmetric interconnection into the locked portion of the circuit, extremely increasing the depth of the SAT search tree. The building block of the existing solutions in this category are the key-programmable routing blocks, each has its topology, such as crossbar or permutation (logarithmic) network. Having such structures sends the corresponded CNF far away being *under/over constrained*, and when the SAT problem is a medium-length CNF, it brings difficulties for the SAT solver.

However, some recent studies introduce some SAT-based attacks that narrowly work effectively on this breed of locking [79], [80], [159]. In [80], the authors propose a *canonical prune-and-SAT* (*CP&SAT*) attack, which exploits a bounded variable addition (BVA) pre-processing step to reduce the size and complexity of the CNF representation of the key-programmable routing blocks used for routing-based obfuscation. After reduction using the BVA, the reduced CNFs (corresponded to numerical bound problems) will be merged again with the circuit's CNF, and the SAT solver could be executed on the reduced CNF version. The main steps of the CP&SAT attack are: (1) It first models routing blocks as the most special case of numerical bound problems[9]; (2) Then by using BVA on the numerical bound problems, helping to reduce the size of the routing blocks as the input to the SAT solver; (3) The reduced problem will be solved by the SAT attack. The approach in [79] proposes the same mechanism with an extension on key-based LUTs. Although these attacks work perfectly fine on routing blocks, their efficiency will be drastically mitigated when routing blocks are twisted with extra logic [80].

In [159], a neural-network-guided SAT attack (*NNgSAT*) has been introduced that examines the capability and effectiveness of a message-passing neural network (MPNN) for solving different types of complex structures, e.g., big multipliers, large routing networks, or big tree structures. In NNgSAT, as demonstrated in Fig. 21, MPNN has been engaged and trained on the specific SAT problems to be used as a guide for the SAT solver within the SAT attack. The MPNN-based SAT solver is called in parallel with the actual SAT solver per each SAT iteration. Based on a pre-defined threshold time, if the actual SAT solver could find the satisfying assignment before the time threshold, the MPNN-based SAT solver will be skipped, and for the next step, both solvers will be called again. However, in those cases that the SAT solver could not find the satisfying assignment before the time threshold, a part of the predicted satisfying assignment by the MPNN-based SAT solver, which has the highest literal votes, will be extracted as a new (guiding) learned constraint, to help the actual SAT solver for finding the precise satisfying assignment.

4.1.3 OG Combinational Structural/Functional Attacks

The oracle-guided test-based and SAT-based attacks on combinational logic locking are completely designated for evaluating and revealing the vulnerabilities of the logic locking techniques using the functional attributes of the circuit. However, there exists another set of de-obfuscation attacks that try to concentrate on both structural and/or functional properties of the logic locking to break the existing countermeasures. This group of attacks reveals that to have a well-designed and robust logic locking technique, it

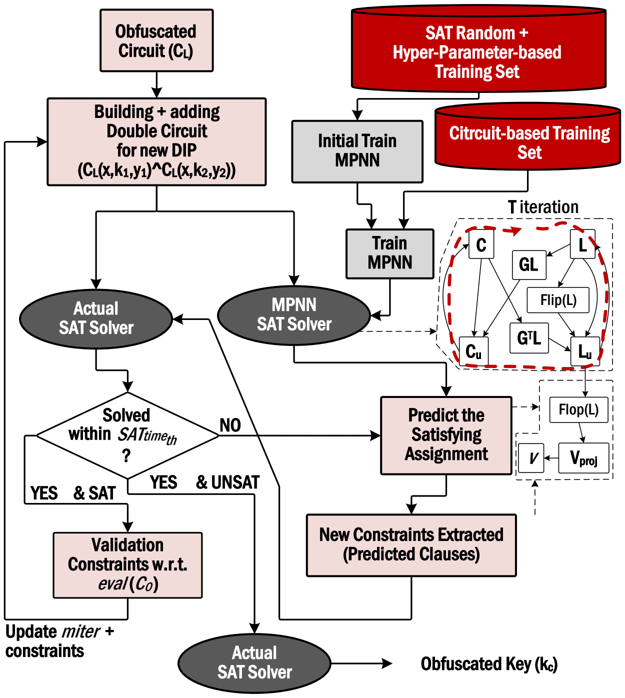

Obfuscated Circuit (C_L)
SAT Random + Hyper-Parameter-based Training Set
Building + adding Double Circuit for new DIP ($C_L(x,k_1,y_1)$^$C_L(x,k_2,y_2)$)
Initial Train MPNN
Citrcuit-based Training Set
T iteration
Train MPNN
C
L
GL
Flip(L)
C_u
G^TL
L_u
Actual SAT Solver
MPNN SAT Solver
Solved within $SAT time_{th}$?
NO
Predict the Satisfying Assignment
Flop(L)
V
V_{proj}
YES & SAT
YES & UNSAT
Validation Constraints w.r.t. eval(C_O)
New Constraints Extracted (Predicted Clauses)
Update miter + constraints
Actual SAT Solver
Obfuscated Key (k_c)

Fig. 21: The Major Steps of NNgSAT Attack [159].

[9] most special case of numerical bounds is when among p variable only *ONE* variable is true, called at-most-1 constraint. It resembles the routing blocks in which for each output, it could be connected to only one of the routing block's input.

needs to be evaluated through different forms of analysis, e.g. functional and structural analysis.

4.1.3.1 Removal Attack: As shown in Fig 8, for implementation of flipping circuit in point function techniques, the locking circuitry is completely decoupled from the original circuit. A removal attack could identify and remove/bypass the locking circuitry to retrieve the original circuit and to remove dependence on key values [151]. The removal attack was first presented to detect and remove SARLock [47]. In the presence of removal attack, different studies investigated the SAT-resistant solutions that are hard to be detected and removed (preventing removal by pure structural analysis), one of the examples of which was Anti-SAT [48].

As we discussed previously, some SAT-based attacks on compound logic locking techniques, such as Bit-flipping and approximate-based attacks (§4.1.2.2, §4.1.2.4), can reduce the logic locking problem to a point function problem. Now with the introduction of the removal attack, such attacks could be integrated with the removal attack to completely break the compound logic locking techniques. For instance, the AGR attack discussed in Section 4.1.2.4 integrates AppSAT with removal to break compound logic locking techniques.

4.1.3.2 Signal Probability Skew (SPS) Attack: The Signal Probability Skew (SPS) attack [160] leverages the structural traces to identify and isolate the Anti-SAT block as a point function technique [48]. *Signal probability skew* (SPS) of a signal x is defined as $s = P_r[x = 1] - 0.5$, where $P_r[x = 1]$ indicates the probability that signal x is 1. The range of s is $[-0.5, 0.5]$. If the SPS of signal x is closer to zero, an attacker has a lower chance of guessing the signal value by random. For a 2-input gate, the signal probability skew is the difference between the signal probability of its input wires. In Anti-SAT, the flipping-circuit is constructed using two complementary circuits, g and $\overline{g}$, in which the number of input vectors that make the function g equal to 1 (p) is either close to 1 or $2^n - 1$. These two complementary circuits converge at an AND gate G. Considering this structure, and the definition of SPS, the *absolute difference of the signal probability skew* (ADS) of the inputs of gate G is quite large, noting that the SAT resilience is ensured by this skewed p. After detecting the gate G, removal can be applied. Algorithm 9 shows the SPS attack flow, which identifies the Anti-SAT block's output by computing signal probabilities and searching for the skew(s) of arriving signals to a gate in a given netlist.

Algorithm 9 SPS Attack Algorithm [160]

```
function SPS_ATTACK(Circuit $C_L$)
    $ADS_{arr} \leftarrow \{\}$;
    for each gate $\in C_L$ do
        $ADS_{arr}$(gate$_i$) $\leftarrow$ Compute_ADS($C_L$, gate$_i$);
    $G \leftarrow$ Find_Maximum($ADS_{arr}$);
    $Y \leftarrow$ Find_value_from_skew($G$);      ▷ Correct value of Anti_SAT output
    $C_{Lock} \leftarrow$ remove_TFI($C_L$, $G$, $Y$);      ▷ Transitive FanIn of the gate $G$
    return $C_{Lock}$                  ▷ $C_{Lock}$: $C_L$ after removing Anti_SAT block
```

4.1.3.3 Functional Analysis (FALL) Attack: Since point function techniques are revealed to be vulnerable against structural-based attacks, the stripping functionality (SFLL) is used to invalidate the circuit after removal, as described in §3.2. SFLL is an extended version of point function techniques, in which the original circuit is modified for at least one input pattern (cube) using a *cube stripping unit*, demonstrated in Fig. 8(b). As shown, Y_{fs} is the output of the stripped circuit, in which the output corresponding to at least one input pattern is corrupted. The restore unit not only generates the flip signal for one input pattern per each wrong key, but it also restores the stripped output to recover the correct functionality on Y. Note that applying removal attack on restore unit recovers Y_{fs}, which is not the correct functionality. In addition, SFLL-HD is able to protect $\binom{k}{h}$ input patterns that are of HD h from the k-bit secret key, and accordingly uses HD checker as a restore unit.

The main aim of the FALL attack is to exploit the resiliency of the SFLL technique [161]. In this attack model, the adversary is assumed to be a malicious foundry that knows the locking algorithm (SFLL) and its parameters (h) in SFLL-HD. The FALL attack is carried out in three main stages and relies on structural and functional analyses to determine the potential key values of a locked circuit:

(i) *Detection of comparator*: The FALL attack tries to find all nodes which are the results of comparing an input value with key input. It is done by using a comparator identification check. Such nodes ($nodes_{RU}$), which contains these particular comparators, are very likely to be part of the functionality restoration unit. The set of all inputs that appear in these comparators, should be in the fan-in cone of the cube stripping unit. Then, it finds a set of all gates whose fan-in-cone is identical to the members of $nodes_{RU}$. This set of gates must contain the output of the cube stripping unit.

(ii) *Functional analysis of candidate nodes*: The attacker applies functional analysis on the candidate nodes suggested by and collected from step (i) to identify suspected key values. Broadly speaking, the attacker uses functional properties of the cube stripping function used in SFLL, to determine the values of the keys. Based on the author's view, this function has three specific properties. So, per each property, they have proposed a specific attack algorithm. So, three attack algorithms exploit the unateness and hamming distance properties of the cube stripping functions. The input of these algorithms is circuit node c, computed from the first stage, and the algorithm checks if c behaves as a hamming distance calculator in the cube stripping unit of SFLL-HD. If the attack is successful, the return value is the protected cube.

(iii) *SAT-based key confirmation*: They have proposed a SAT-based key confirmation algorithm using a list of suspected key values and I/O oracle access, that verifies whether one of the suspected key values computed from the second stage, is correct.

4.1.3.4 CASUnlock Attack: CASUnlcok [162] is another structural-based attack that is specifically proposed to exploit cascaded logic locking, a.k.a. CASLock [55]. CASLock is a point function technique that proposes a variant cascaded AND-OR tree that builds variant corruptibility for a point function technique, and it is resilient against all previously discussed attacks. This attack works in two ways: (i) By relying on structural traces left after the synthesis process, this attack shows how the flipping circuitry in CASLock can be pinpointed after re-synthesis allowing the adversary to recover the original IP. Similar to FALL, it also traces the fan-out of all key inputs, and then pinpoints the convergent gate that is the output of the flip signal. (2) it exploits the connectivity of key inputs, thereby enabling the SAT attack to decipher the secret key with only a polynomial number of queries.

4.1.3.5 EDA-based (SPI/Valkyrie) Attack: Sparse prime implicant (SPI) attack is another synthesis-based (EDA-based) mechanism that reveals the structural vulnerability in point function techniques [163]. In almost all point function techniques, the approach is relying on the assumption that the underlying EDA synthesis tool used by the semiconductor industry can effectively conceal the structure of flipping/masking/stripping circuitry in the DUT. However, in SPI, a comprehensive exploration has been done through different industrial/academic synthesis tools, including Cadence Genus, Synopsys Design Compiler, Synopsys Synplify, Xilinx Vivado, Mentor Graphics Precision RTL, and ABC, invalidating this assumption. So, similar to what was witnessed in CASUnLock, SPI also confirms that the structural traces left through the EDA tool can be exploited for breaking all point function techniques. SPI is specifically relying on the notion of prime implicants that is the underlying elements in logic optimization and redundancy reduction once the sum of products (SOP) format has been engaged for the circuit representation. In SPI, different specifications and properties have been defined and exploited around this notion to show how protected input patterns in the point function can be revealed. Valkyrie [164] almost follows the same direction, in which a security diagnostic tool has been introduced that checks for structural vulnerabilities. Also, they propose a circuit-recovery attack that shows how an adversary can exploit identified vulnerabilities by the diagnosis tool to recover the original functionality. Similar to the SPI attack, the analysis by Valkyrie invalidates the assumption that the design team can rely on the synthesis tool for absorbing the point function sub-circuitry into the original part.

4.1.4 Summary of OG Combinational Attacks

As shown in Fig. 7, the first *four* logic locking sub-groups, i.e. the primitive, the point function, the cyclic-based, and the LUT/routing-based, assume that the scan access for the adversary could be *OPEN*. Hence, the adversary has the capability of targeting each combinational part (CL) directly. This is the main motivation of all functional-oriented oracle-guided attacks on combinational circuits, which rely on I/O query-based techniques. On another side, analysis of the locked designs based on the structural traces reveals that many of the logic locking techniques suffer from this form of vulnerability, as demonstrated by the more recent structural attacks on point function techniques [163], [164]. As reviewed in this Section, many of the logic locking techniques in the first *four* logic locking sub-groups have been broken using these attacks. Table 12 reflects the current status of oracle-guided combinational attacks. For each attack, Table 12 tries to concisely answer to these *four* questions: (1) How the attack flow works; (2) Which logic locking techniques could be broken using the attack; (3) What is the limitation and challenges of the attack; and (4) What is the existing/potential countermeasure against the attack?

4.2 Oracle-Guided (OG) on Sequential Circuits

Since the availability of the scan chain undermines the robustness of many logic locking techniques once one of the oracle-guided attacks is in place, as demonstrated in Fig. 7, there exist other sub-groups, such as scan-based logic locking, scan blockage, and sequential logic locking, in which the availability of the scan chain is targeted to be restricted/blocked. In such scenarios, assuming that the oracle is still available, the adversary access would be limited to the PI/PO of the oracle. Therefore, all of the previously discussed functional-oriented I/O query-based de-obfuscation attacks will fail to evaluate and break the locked circuits with limited scan chain access. However, further studies on logic locking show that restricting access still cannot guarantee robustness against state-of-the-art threats. In this Section, we will holistically review the attacks that follow such assumptions, and we call them oracle-guided de-obfuscation attacks on sequential circuits. This group of attacks follows these assumptions:

It is oracle-guided: The attacker requires to have access to one additional activated/unlocked version of the chip (oracle).

It is on sequential circuits: It implies that having access to the DFT structure, i.e. scan chain pins, is limited/blocked/locked, and the adversary access is limited to PI/PO of the oracle.

It is invasive: The attacker requires to have access to the netlist of the locked circuit (locked GDSII at the foundry or reverse-engineered of chip acquired from the field/market).

4.2.1 OG Sequential Unrolling-based Attacks

In an unrolling-based attack, a derivative of the SAT attack that works on sequential circuits has been engaged to evaluate the robustness of logic locking techniques with restricted scan access. The preliminary version of the SAT-based sequential de-obfuscation attack was first introduced in [165]. The sequential SAT attack shows how the SAT solver could still be engaged to break the logic-locked circuits with limited scan chain access while unrolling is used as a preprocessing step. The sequential SAT attack uses an iterative method to prune the search space, similar to the SAT attack. Due to the limited access to internal registers, instead of seeking a DIP in each iteration, it instead finds a sequence of input patterns X denoted as discriminating input *sequence* (X_{DIS}) that can produce two separate outputs for two different keys. To build the sequence using the SAT attack, the sequential SAT gets the benefit of unrolling/unfolding to create a combinational equivalent circuit. Then, the SAT solver will be used for a specific depth to generate the DIS. For instance, Fig. 22, shows a sequential circuit unrolled for τ clock cycles. In this case, the SAT could be invoked to return a DIS with a length of τ. In comparison with combinational SAT attack, Fig. 23 shows the main steps of the SAT-based sequential de-obfuscation, in which unrolling determines the length of DISes.

In an unrolling-based SAT attack, the generation of unrolled instances per clock cycle (as a query) can be also done by a bounded-model-checker (BMC). Hence, some studies also refer to this group of attacks as BMC attacks on sequential logic locking. So, to accomplish the unrolling step, with determining the boundary, the BMC engine could be invoked

TABLE 12: Overview of Oracle-guided Attacks on Combinational Circuits.

Attack	Mechanism	Applicable to^{*1}	Limitation & Challenges	Countermeasures
Sensitization [12]	(1) Each key bit is considered as a stuck-at fault, (2) applying fault sensitization, (3) Finding a test pattern that propagates fault to PO/SO, (4) Applying test pattern to the oracle to observe the correct key	some RLL [11]	Only applicable to acyclic combinational circuits, (2) very limited w.r.t. key size and key location	SLL [12], FLL [44]
SAT [45], [46]	(1) Building Double (Miter) Circuit, (2) Finding DIP using SAT solver for two sets of keys, (3) Comparing with Oracle (ruling out wrong Key(s)), (4) Finding all DIPs, (5) Finding correct key using all DIPs	RLL [11], FLL [44], SLL [12], Lut-based [66]	Circuit must (1) be only acyclic, (2) have open scan access, (3) be only Boolean, (4) have no complex arithmetic units	All except primitive
AppSAT [147], Double-DIP [148]	(1) Running the SAT attack for u iterations, (2) Checking the output error rate for a set of input patterns after u iterations, (3) Returning the key if error rate is below a threshold, (4) Otherwise go back to step 1	Point Function, Point Function + Primitive	(1) Finding the minimal set of input patterns which excite the error, (2) Error rate analysis	G-AntiSAT [53], S-AntiSAT [54], CASLock [55], SFLL-felx/rem [51], [52]
Bit-Flipping [150]	(1) Decoupling keys (point function vs. primitive) based on POs' hamming distance per each DIP, (2) Fixing point-function keys, (3) Finding primitive keys using SAT, (4) Applying Bypass attack for point-function keys	Point Function + Primitive	(1) Not applicable if keys of primitive and point function is twisted, (2) Not applicable if function is stripped	TTLock [50], G-AntiSAT [53], S-AntiSAT [54], CASLock [55], SFLL-felx/rem [51], [52]
CycSAT [153], BeSAT [154], icySAT [155]	(1) Adding cyclic-based constraints to the SAT circuit before invoking the SAT attack (as a pre-processing step), (2) running the SAT attack on a cyclic-defused SAT circuit, (3) Recording/modifying some constraints to avoid infinite loops	cyclic [58], [63]	(1) Adding cycles exponentially w.r.t. the number of feedbacks, (2) Adding stateful/oscillating cycles, (3) asynchronous circuits	SAT-hard cyclic [60], [61], [62], Cross-lock [76], Full-lock [77], DF obfuscation [100]
Bypass [149]	(1) Selecting a random key for the obfuscated netlist, (2) Using SAT to find DIPs based on the selected key, (3) Applying DIPs to the Oracle, (4) Adding an extra circuit at the POs to re-flip the incorrect outputs (Bypassing)	Point Function (only low corruptible	(1) The selected random key might have numerous DIPs, (2) Needs to find all DIPs to consider the attack as an accurate one	TTLock [50], G-AntiSAT [53], S-AntiSAT [54], CASLock [55], SFLL-felx/rem [51], [52]
SMT [102]	(1) Modeling behavioral obfuscation using one theory solver (like graph for timing or RTL (FSMD) or BitVector for hamming distance), (2) Integrating theory solvers with the Boolean SAT solver, (2) Co-solving using theory+SAT solvers	DLL [15]	(1) Theory solvers are limited, (2) theory is hard to be modeled in some behavioral obfuscation techniques	Complex models
NNgSAT [159]	(1) Using message-passing neural network as a classifier to predict the satisfiability, (2) Guiding the actual SAT solver based on the output (prediction) of the message-passing neural network	Cross-lock [76], Full-lock [77], Interconnect [78]	(1) Misguide rate could be high if trained on small set of benchmarks, (2) Accuracy could be low when dynamicity is in place	——
CP&SAT [80]	(1) Simplifying the routing modules using cardinality constraint as a pre-processor, (2) Run the SAT attack on simplified locked netlist	Interconnect [78], Cross-lock [76], Full-lock [77]	Inefficient optimization on strongly twisted logic and routing locking	Interlock [80]
Removal [151]	(1) Finding the obfuscation module (wrapper one) using structural analysis, (2) Extracting the original circuit wrapped by isolated obfuscation module	Anti-SAT [48], SARLock [47]	(1) Inefficient if obfuscation module and original design is twisted using fake (irremovable) logic, (2) When the function is stripped	TTLock [50], G-AntiSAT [53], S-AntiSAT [54], CASLock [55], SFLL-felx/rem [51], [52]
SPS [160]	(1) Computing absolute signal probability skew for each gate, (2) finding the gate which has the maximum skew, (3) determining the correct output based on the location of the selected gate, (4) Revealing the key based on the correct output	Anti-SAT [48]	Inefficient if the sub-circuits of flipping circuitry is (1) not complement of each other, (2) built using LUTs	TTLock [50], G-AntiSAT [53], S-AntiSAT [54], CASLock [55], SFLL-felx/rem [51], [52]
FALL [161]	(1) Finding comparator of flipping circuitry using structural analysis, (2) Finding sub-circuit of restoration unit and cube stripping, (3) finding the suspected key values based on the selected sub-circuit using functional analysis, (4) Verify the key using SAT solver	TTLock [50], SFLL [51]	(1) Hard to find the units through structural analysis when re-synthesis is applied, (2) Hard (almost impossible) if camouflaged gates are used	CASLock [55], SFLL-felx/rem [51], [52]
EDA-based [163], [164]	Analysis of structural traces happening through EDA tools, like synthesis flow	All point functions	——	——

^{*1} is able to break

to model the locked circuit as an FSM, and the specification could be formalized by temporal logic properties. So, the BMC could be exploited as an alternative approach to do the symbolic model checking before invoking the SAT procedure. Algorithm 10 demonstrates the overall procedure of the primitive sequential SAT attack, once the unrolling is done using the BMC engine. $C(x, k, y)$ indicates the locked circuit generating output sequence y using input sequence x and the key value k, and $C_{BlackBox}(x)$ refers to the output sequence of the oracle for the same input sequence. After building the model from the locked circuit, the attack instantiates a BMC to find the x_{DIS}. Then, the model would be updated with a new constraint to guarantee that the next pair of keys, which will be discovered in the subsequent attack iterations, produce the same output for previously discovered x_{DIS}. The iterations continue until no further x_{DIS} is found within the boundary of b. After reaching the boundary, if the algorithm passes three criteria, the key could be found with one more SAT instantiation. The boundary could be extended if termination conditions are not met. The primitive sequential SAT attack in [165] specified three main termination conditions for this attack:

(1) Unique Completion (UC): This condition verifies that the key generated by the algorithm is unique. The attack is successfully ended, and the key is the correct one if there is only one key that meets all previous DISes.

(2) Combinational Equivalence (CE): If there is more than one key for all previously found DISes (non-unique key), the attack checks the combinational equivalency of the remaining keys. In this step, the D/Q of FFs are considered as pseudo

PO/PI allowing the attacker to treat the circuit as combinational. The resulting circuit is subjected to a SAT attack, and if the SAT solver fails to find a different output or next state for two different keys, it concludes that all remaining keys are correct and the attack terminates successfully.
(3) Unbounded Model Check (UMC): If both UC and CE fail, the attack checks the existence of a DIS for the remaining keys using an unbounded model checker. This is an exhaustive search with no limitation on bound (or the number of unrolls). If no DIS is discovered, the existing set of DIS is a complete set, and the attack terminates. Otherwise, the bound is increased and previous steps are repeated.

Algorithm 10 Sequential Attack on Obfuscated Circuits [165]

```
$b = initial\_boundary, Terminated = False$;
$Model = C(x, k_1, y_1) \wedge C(x, k_2, y_2) \wedge (y_1 \neq y_2)$;
while not $Terminated$ do
    while $(x_{DIS}, k_1, k_2) \leftarrow BMC(Model, b) = T$ do
        $y_f \leftarrow C_{BlackBox}(x_{DIS})$;
        $Model = \wedge C(x_{DIS}, k_1, y_f) \wedge C(x_{DIS}, k_2, y_f)$;
    if UC$(Model, b) \vee$ CE$(Model, b) \vee$ UMC$(Model)$ then
        $Terminated$;
    $b = b + boundary\_step$;
```

As demonstrated in Fig. 23, the SAT circuit (SATC) requires an update per each iteration, in which the newly learned clauses will be added to the list of previously found clauses. Hence, with more iteration, the size of the SAT problem will be increased drastically. In sequential SAT, it gets worse because the SAT problem will be expanded in two dimensions, i.e. iterations and unrolling. Hence, sequential SAT attack runs into the scalability issues as it relies on two subroutines which are in **PSPACE** and **NP**, thereby, failing to terminate for even moderately small circuits, which contain only a few thousand gates. Hence, some more recent studies investigate different possibilities to mitigate this issue. Shamsi *et al.* propose a fast sequential de-obfuscation attack, called KC2 [166], which implements different dynamic optimization tweaks, such as incremental SAT-solving, key-condition sweeping, negative key-condition compression, etc., to improve and accelerate primitive sequential SAT attack. More recently, the open-source toolset RANE [167] has offered a unified framework with unique interfaces and can take advantage of the capabilities (like scalability) of commercial formal verification tools such as Cadence JasperGold and Synopsys Formality for de-obfuscation at different stages with fewer difficulties. Although these approaches show significant improvement, more studies still show the lack of scalability induced by the de-obfuscation process, especially, for large circuits, e.g., micro-controllers and processors. Another recent study has also explored the effectiveness of different SAT/BMC-related guiding mechanisms, such as restart and initialization, on the scalability of this breed of attacks [168]. In this study, warming up the BMC before its invocation has been investigated. The exploration shows that for a significant part of the experience, warm-up as an initialization guide can improve the performance of this type of attack.

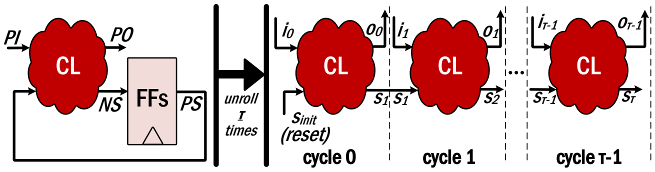


Fig. 22: Unrolling the Sequential Circuit for τ Cycles.

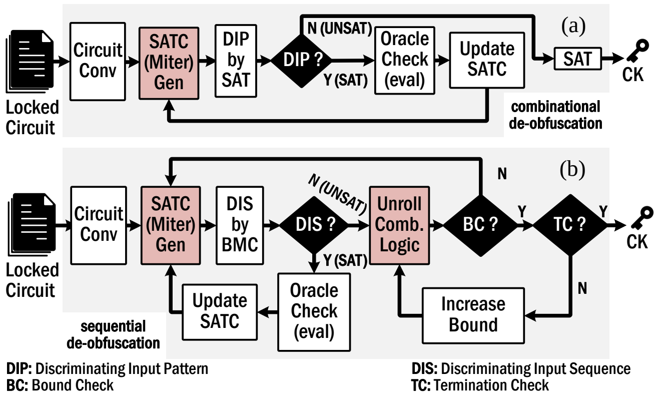


Fig. 23: Combinational SAT vs. Sequential SAT Attack.

4.2.2 OG/OL Sequential on FSM Locking

When FSM locking is in place [13], [14], [91], [93], [94], [95], unlike all other logic locking techniques, there might exist no wiring, input, or logic, as the explicit declaration of the key (or key gate). Hence, none of the above-mentioned attacks work on FSM locking, where all these attacks assumptions rely on some formulation around key or key gates.
4.2.2.1 2-stage (OL/OG) Attack on FSM Locking: To assess the strength of FSM locking techniques with the above-mentioned specification, however, different studies evaluated the possibility of deploying a 2-stage attack on locked FSMs [92], [96]. Algorithm 11 shows the overall flow of 2-stage attacks on FSM locking that is composed of three main steps:
(i) *topological/structural analysis*: (described in line 2-13 of Algorithm 11), which is a detection algorithm to find FFs that are responsible for storing the state values (separating them from datapath FFs). The topological analysis is mostly derived from [169], which identifies FFs whose input contains a combinational feedback path from their output. Then, it reduces the set of possible state FFs by (a) grouping the FFs controlled by the same set of signals, and (b) finding strongly connected components (SCC) using Tarjan's algorithm [92], [96], [170], [171].
(ii) *functional analysis*: (described in line 14-21 of Algorithm 11) that finds the state transition graph (STG) based on the list of FFs found in step 1. In the functional analysis stage (stage 2), the attacker attempts to re-calculate and extract the STG. This is done by first attempting to find the initial state, and then identifying the reachable states by creating a reduced binary decision diagram (BDD) or using a SAT solver.
(iii) *Matching/Extracting Original FSM*: In this case, based on the extracted STG, the original part of FSM will be retrieved based on some behavioral specifications. In most

Algorithm 11 2-stage on FSM Locking [92], [96]

```
function FSM_EXTRACT(Circuit $C_L$)
    $SFF \leftarrow []$;                                     ▷ State Flip Flops
    $RS \leftarrow$ classify(FFs);           ▷ Classifying FFs into Register Sets
    for each $set \in RS$ do
        $set \leftarrow set$ - notSCC($set$);        ▷ Keeps Strongly Connected Components
        if is_splittable(set) then
            $RS \leftarrow \{RS - set\} \cup$ split($set$);
    $CLFP \leftarrow$ find_feedback_circuits($C_L$, Reg_Sets);
    for each $set \in RS$ do
        $set \leftarrow set$ - notInfDep($set$);     ▷ Keeps Intersected Influence/Dependence
        $set \leftarrow set$ - InputIndependt($set$);       ▷ Check Control Metrics
        update($CLFP$);
        SFF $\leftarrow$ SFF $\cup$ $set$;
    $S_0 \leftarrow$ initial_state(state_regs); $SQ \leftarrow []$;          ▷ State Queue
    $SQ \leftarrow SQ \cup S_0$; $STF \leftarrow []$;          ▷ State Transition Table
    while $SQ \neq []$ do
        state $\leftarrow SQ$.dequeue();
        for each $DIP$ do                                 ▷ DIP found by SAT
            if eval(state_regs, $DIP$, state) $\notin SQ$ then
                $SQ$.enqueue(nx_state);
                $STF \leftarrow STF \cup \{$state, $DIP$, nx_state, $PO\}$
    return $SQ$, $S_0$, $STF$;              ▷ States, Initial, Transition Func.
```

FSM locking solutions, the adversary can readily distinguish the original part of the FSM from either extra added states or extra state transitions, leading to extracting the original FSM. In this step, for complex cases that are hard to distinguish between original and extra states, some random-based stimuli will be matched with the oracle.

In this attack, topological analysis is a crucial factor as the success rate of this attack. Because for cases that the topological analysis cannot retrieve a correct set of FFs as the state holders, the returning set might have a sub-set of actual state FFs with some data FFs, and in this case, behavioral analysis on re-calculated STG might lead to an incorrect FSM.

4.2.2.2 RANE Attack on FSM Locking: API-based invocation of different solvers and formal verification tools in RANE attack allows this framework to formulate multiple threat models and attack flows [167]. Hence, a specific model of FSM locking, i.e., HARPOON-based FSM locking, has been modeled by the RANE attack that shows more scalability versus the 2-stage attacks on FSM. In this model, the initial state has been formulated as the secret (unknown parameter or key variable), and the formal tool has been invoked to find this secret. Then, the formal tool has been called once more to find the unlocking sequence reaching to the initial state. By using these two steps, RANE shows how FSM locking could be still modeled using a similar approach as defined for the primitive sequential SAT attack with a dedicated key variable. Algorithm 12 shows how this attack has been formulated in the RANE framework. Although this new model could provide better scalability compared to 2-stage attacks on FSM, since it is BMC-based (expansion through two dimensions) their experiments still show the scalability issue for larger circuits.

4.2.2.3 Functional Corruptibility-Guided on FSM Locking: In [172], another derivation of SAT-based attack has been studied that is applicable to key-less FSM-based sequential logic locking, called Fun-SAT. In Fun-SAT, the minimum number of unrollings needed to find the correct secret (here the unlocking sequence) will be estimated that will allow the attack to directly jump to the depth close to the final satisfying assignment(s). This method has been realized using bounded-depth function corruptibility (FC) analysis, and the whole attack consists of two major steps, which are FC analysis and SAT solving. Bypassing the gradual unrolling in Fun-SAT can improve the attack performance by up to 90x, showing how the conventional BMC/unrolling suffers from the scalability issue.

4.2.2.4 ORACALL on Cellular Automata guided Locking: In [173], a new attack on a specific FSM locking has been introduced, in which the locking of the FSM has been done using cellular automata cells [174], and each FSM transition is protected using a secret key that resembles black holes in traditional FSM locking techniques [14]. The overall flow of ORACALL is also similar to other unrolling-based and FSM-based attacks, in which a specific sequence of the pattern will be applied to the circuit as the pre-processing step, and then SAT attack will be called to retrieve the correct key related to each transition.

4.2.3 OG Sequential Scan/Leakage-based Attacks

Similar to attacks described in §4.2.1 and §4.2.2, since a sub-set of defenses target scan locking or blockage (§3.5), a sub-set of studies eventually evaluated and revealed the vulnerabilities of this breed of locking.

4.2.3.1 ScanSAT on Scan Locking: ScanSAT aims to break the scan-based logic locking techniques. In scan-based logic locking, since the availability of scan is *locked*, the adversary has only access to the PI/PO. Hence, similar to the primitive sequential SAT attack, KC2, and RANE, ScanSAT [175] is based on the fact that the complex τ-cycle transformation of scan-based locking could be modeled by generating an unfolded/unrolled combinational equivalent counterpart of the scan-locked circuit. In this case, the locking parts added into the scan path become part of the resultant combinational circuit. Hence, the adversary faces a combinational (unrolled) locked circuit with key gates at the pseudo-primary I/Os of the circuit. The combinational equivalent of a locked circuit in Fig. 24(a) is provided in Fig. 24(b), where the locking on the stimulus and the response are modeled separately as combinational blocks driven by the same scan locking key. In general, ScanSAT models the locked scan chains as a logic-locked combinational circuit, paving the way for the application of the combinational SAT attack to

Algorithm 12 RANE Attack Model on FSM Locking

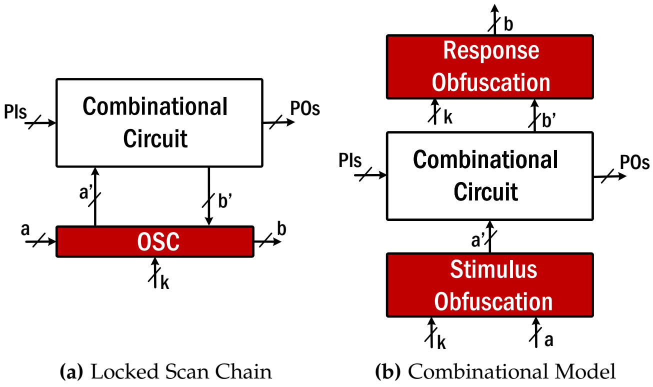
```
—————— Finding Secret 1 (init state) ——————
$Model \leftarrow C_{seq}(x, s_{init1}, y_1) \wedge C_{seq}(x, s_{init2}, y_2)$;
while !UC($Model$) $\wedge$ !CE($Model$) $\wedge$ !UMC($Model$) do
    $DIS_i \leftarrow Formal(Model \wedge (y_1 \neq y_2))$;
    $y_i \leftarrow C_{BlackBox}(DIS_i)$;
    $Model \wedge= C_{seq}(DIS_i, S_{init1}, y_i) \wedge C_{seq}(DIS_i, S_{init2}, y_i)$;
—————— Finding Secret 2 (unlocking sequence) ——————
$Model \wedge= CE^0_u(us_0, s_{rst}, y_{US0}, s_{us1})$;
$i \leftarrow 1$;
while $Formal(Model \wedge (s_{us_i} = \hat{s}_{init})) \rightarrow Fail$ do
    $Model \wedge= CE^i_u(us_i, s_{usi}, y_i, s_{us_{i+1}})$;
    $i \leftarrow i + 1$;
return $Formal(Model \wedge (s_{us_i} = \hat{s}_{init}))$    ▷ {init state, unlocking sequence}
```

reveal the key (sequence of unrolled), unlocking the scan chains, and thus, restoring access to the oracle.

In addition to de-obfuscating the statically scan-locked circuit, ScanSAT is also able to be applied on the dynamically scan-locked circuit that is locked by DOS architecture [83]. In DOS architecture, an LFSR has been engaged to generate runtime dynamic keys. In oracle-guided attacks, since the output of the netlist evaluation, such as DIPs and DISes, are continuously checked with the oracle, dynamically change of the configuration in the oracle will corrupt this oracle-oriented evaluation, and results in failure of such attacks. In ScanSAT, it is assumed that after successfully reverse engineering, the LFSR structure, and consequently its polynomial is known to the adversary. Hence, finding the seed of LFSR and the update frequency parameter (p as the time interval of updating the key based on the LFSR output), which is the only secret in DOS architecture, would lead to deriving all the keys that are dynamically generated on the chip.

A simple method to identify p is to apply the same stimulus pattern repeatedly from the SI and observe the response through the SO. The point is that after p capture operations, by repeatedly applying the same stimulus, the response would be different because of the updated key; thus, most likely, there will be a noticeable change in the observed response, helping detect the update operation on the key.

After finding p, the same approach that was used for static scan obfuscation would be used in this case. The difference now is that the SAT attack could be executed for at most p iterations (after p iterations, the key is updated). If more than p DIPs are required to identify a dynamic key, the SAT attack needs to be terminated prematurely upon p DIPs. Another SAT attack must be executed subsequently to identify the next dynamic key in the sequence still within p iteration. Since the updated key is generated by the LFSR whose polynomial is known for the adversary, independent SAT attack runs on each dynamic key reveals partial information of the seed; thus, the information from independent SAT attack runs by gradually gathering information about the seed in every run, and finally, by incorporating into the ScanSAT model, the relationship between the seed and the keys would be revealed.



(a) Locked Scan Chain **(b)** Combinational Model

Fig. 24: Converting a Locked Scan Chain to its Combinational Counterpart [175].

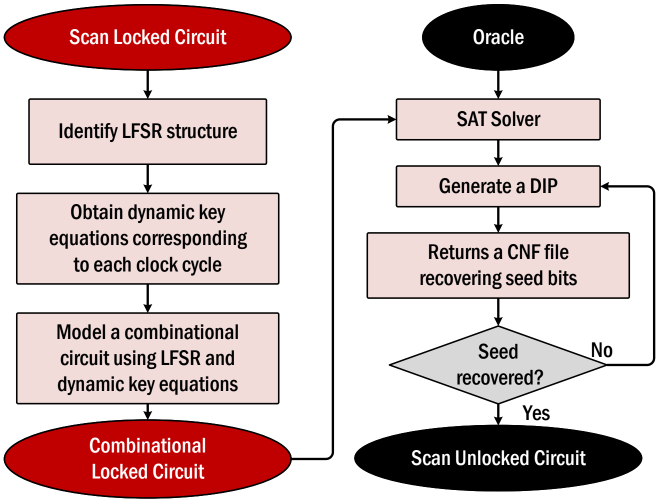

Scan Locked Circuit
Identify LFSR structure
Obtain dynamic key equations corresponding to each clock cycle
Model a combinational circuit using LFSR and dynamic key equations
Combinational Locked Circuit
Oracle
SAT Solver
Generate a DIP
Returns a CNF file recovering seed bits
Seed recovered?
No
Yes
Scan Unlocked Circuit

Fig. 25: Flowchart for the DynUnlock Attack [176].

4.2.3.2 DynUnlock on a Specific Dynamic Scan Locking: As a countermeasure against ScanSAT attack, dynamic encrypt flip-flop (EFF-Dyn) [85] combines scan locking approach from EFF [84] and a PRNG, to introduce dynamicity in the design. In EFF-Dyn, based on the value of scan controlling signal, i.e. scan enable (SE), the source of the key to the circuit would be changed. In the test mode, the test key must be provided externally, and in case of a mismatch with the locking key embedded in the circuit, there exists a PRNG that updates the key in every clock cycle, thereby controlling the key gates dynamically. However, similar to LFSR, the structure of PRNG and its polynomial would be known for the adversary after successfully reverse engineering. Hence, DynUnlock [176] proposes a similar approach to find the seed of the PRNG in EFF-Dyn.

Assuming that the structure of PRNG is similar to an LFSR, in DynUnlock [176], as demonstrated in Fig. 25, it first starts by reverse-engineering the LFSR circuit and obtaining the equations corresponding to each clock cycle. Next, it determines the location of key gates inserted between the SFFs. Then it models this sequential logic circuit into a combinational circuit with SFFs replaced with inputs and outputs. Once modeling is complete the combinational obfuscated counterpart circuit, with seed bits acting as primary key inputs, is fed to a SAT solver, which provides a DIP and its corresponding output pattern. In [176], the authors carry out the attack for just one capture cycle. To recover more bits, they restart the LFSR circuit and obtain a new DIP and its corresponding output pattern from the SAT solver, and recover more seed bits. they repeat the restart step until all the seed bits have been recovered, or the remaining seed bits can be brute-forced.

4.2.3.3 Leakage-based Attacks on Scan Blockage: Due to the failure of scan obfuscation architectures against sequential SAT attacks, more recent studies evaluate and reveal the effectiveness of the scan chain blockage after activation of the obfuscated circuit. However, the structure of the augmented scan chain and the infrastructure used around them for protection might lead to a different form of security leakage. For instance, the first scan blockage architecture called R-DFS was first introduced in [40], in which the obfuscation key is stored in a custom-designed scan (storage) cell. Based on the value of the SE pin and the new pin called $Test$, the key values could be loaded into SCs either directly from TPM or through SI, and the SO will be blocked after activation to avoid any secret leakage. However, the shift-and-leak attack [37] breaks the R-DFS by exploiting the availability of the shift-in process through the SI, and the capability of reading out the PO through the chip pin-outs in the functional mode.

To remedy the leakage issue, the authors in [37] proposed modification to the R-DFS (mR-DFS), which blocks any shift operation after the obfuscation key is loaded from the TPM, removing the ability of an adversary to apply the shift-and-leak attack. However, the work in [38] illustrates how the architectural drawbacks of the modified infrastructure can lead to another glitch-based shift-and-leak attack, which allows an adversary to still leak the logic locking key through the PO even if the shift operation was disabled.

4.2.4 *Summary of OG Sequential Attacks*

The oracle-guided attacks on sequential circuits mostly rely on three different models to successfully break logic locking techniques: (1) unrolling mechanism that still allows the adversary to get the benefit of satisfiability, and it can be done manually or by using BMC. (2) structural/functional analysis of the locked netlist particularly for FSM-based logic locking, and (3) leakage possibility while the scan chain structure is manipulated. In general, compared to combinational attacks, the biggest shortcoming of sequential attacks is the scalability issue of the attack particularly on large circuits. This is when unrolling or BMC is in place, or some structural/functional analysis has been done, or leakage scenarios are targeted to be modeled. Table 13 tries to concisely answer to these *four* questions for attacks on sequential circuits: (1) How the attack flow works; (2) Which logic locking techniques could be broken using the attack; (3) What is the limitation and challenges of the attack; and (4) What is the existing/potential countermeasure against the attack?

4.3 Oracle-Less (OL) Attacks

Unlike almost all previous attacks, where access to oracle is one of the basic assumptions in threat modeling, realizing such scenarios might be hard or even impossible in so many cases. For instance, a malicious end-user has stolen one activated chip from the field, and no other copy of the chip is available to the public. Given such scenarios, a significant portion of attacks, which are known as oracle-less attacks, are those assuming there is no/hard possibility for the adversary to get access to an additional activated (unlocked) chip, and the threat model has been defined in a way that the adversary has access to (i) only the reverse-engineered netlist of the chip (locked GDSII at foundry), or (2) only one packaged chip (unlocked/activated) that can go through physical reverse-engineering if needed (malicious end-user). Although this group of attacks can be more promising as they do not need the oracle, since there is no reference model, we witness the notion of key *guessing/prediction* because there is no explicit way of confirming the recovered key. Additionally, many of these approaches recover the key partially. So, unlike oracle-guided attacks in which time of the attack is important, here the key coverage percentage is a pivotal metric for the evaluation of the success rate of the attack. Hence, some of these attacks suggest that having an unlocked system is recommended to perform functional tests and to verify the correctness of the extracted key [177]. Based on the structure of oracle-less attacks, they can be categorized into three main sub-categories: (i) structural, which could be (a) synthesis-based, (b) ATPG-based, or (c) ML-based, (ii) tampering-based, and (iii) probing-based. In the following, these sub-groups will be discovered holistically.

4.3.1 *OL Structural Synthesis-based Attacks*

Some attack approaches exploit the structural changes induced through the synthesis process. These attacks trace the changes through the synthesis process once the design is fed by different (guessed) key vectors. This breed of attack reveals structure-based shortcomings of logic locking techniques, and the main goal of them is to show that once incorrect (or a specific) key value will be applied to the design, the re-synthesis process on the new design is constrained by the key value reveals some information (based on structural analysis) about the correctness of the applied key values.

4.3.1.1 Desynhtesis Attack: As its name implies, the main aim of the desynthesis attack is to eliminate incorrect keys by desynthesizing the locked netlist [178]. In this attack, a hill-climbing approach has been engaged to re-synthesize the locked netlist for different key guesses. The key guess that yields the maximum similarity between the locked netlist and its resynthesized versions are considered as the correct key. Fig. 26 shows a simple example of how the desynthesis attack works based on similarity factor after re-synthesis constrained by key value. Ideally, the best scenario that guarantees 100% key recovery is brute-force key testing (exhaustively applying all key combinations followed by re-synthesis). But, even if the attacker exhaustively tests all key values, for two major reasons, the success rate of such an attack would be very low: (1) The attack uses a heuristic dis-similarity factor, and eliminate keys with the smallest dis-similarity (as the incorrect keys). Hence, the efficiency of the algorithm used for checking the dis-similarity crucially affects the success rate. (2) Randomness and imperfection of the commercial synthesis tool per each invocation can be misguiding in different cases, thereby reducing the success rate of this attack.

4.3.1.2 SWEEP Attack: In SWEEP [177], which is a constant-propagation attack, the basic idea of the attack is very similar to the desynthesis attack, which is to assign a constant key-value to one key input and synthesize the obfuscated design with that key value. Based on the synthesis report, the attack identifies any structural design features that are correlated to the correct key values. Finally, the correct value of the analyzed key input is identified by comparing them to the synthesis report of the original obfuscated design using a scoring algorithm. SWEEP uses machine learning for the detection and tracing of the features related to the correct key and can be a member of OL structural ML-based attack (§4.3.3) as well. The main steps of the SWEEP attack are (1) training phase and dataset generation, (2) constant propagation, synthesis, and feature

TABLE 13: Overview of Oracle-Guided Attacks on Sequential Circuits.

Attack	Mechanism	Applicable to[*1]	Limitation & Challenges	Countermeasures
Unroll/BMC [165], [166], [167], static scanSAT [175]	(1) Unrolling the sequential circuit (FFs are pivot) for u times (u-cycle combinational counterpart), (2) Apply the SAT attack on u-time unrolled circuit, (3) Finding DIPs and unroll more if there is no more DIP in u-time unrolled circuit, (4) Stop based on termination strategies (UC, CE, or UMC)	sequential lock, Encrypt-FF [84], seql [86]	(1) scalability is low (large number of unrolling and/or large circuit), (2) complexity of unrolling is high in multi-clock and clock-gated designs	DOS [83], dyn-EFF [85], latch/clock-based [99], [100], [101]
dynamic scanSAT [175]	(1) Identify the key update frequency by applying the same stimulus pattern, and observing the response, (2) Extracting the secret seed by applying SAT on unrolled circuit with known update frequency (with updated keys), (3) Extracting original functionality by knowing seed and update frequency	DOS [83]	(1) Hard to apply if different keys generate the partial same response, (2) Impractical for dynamic seed (changing the seed over the time), (3) Impractical if LFSR is replaced with TRNG	Dynamic EFF [85], DisORC [42], DOSC [88]
2-stage [92], [96]	(1) Decoupling state FFs from datapath FFs using structural analysis, (2) Extracting the state transition graph of FSM(s) based on the list of FFs found in step 1, (3) Revealing the original parts of the FSMs	FSM Locking [13], [14], [91]	(1) Hard to apply when datapath and state FFs are strongly connected, (2) Challenging if unreachable states used as trap, (3) Impractical if part of the logic is locked (compound)	DFSSD [93], JANUS/HD [94], [95]
RANE FSM [167], FunSAT [172]	(1) setting the initial state of FSM locking as the key, (2) Unrolling the circuit for a specific number of cycles that required to reach initial state, (3) use SAT/BMC to find initial state as well as number of cycles required (unlocking sequence)	FSM Locking [13], [14], [91]	(1) Inapplicable if intermediate states are locked, like adding black holes for intermediate states, (2) Inapplicable if FSM is generated at the run-time	JANUS/HD [94], [95]
DynUnlock [176]	(1) Finding the LFSR/PRNG equations corresponding to each clock cycle, (2) Finding the location of key gates inserted between the SFFs, (3) Replacing SFFs with PIs/POs (de-sequencing), (4) Applying the SAT attack	Dynamic-EFF [85]	(1) Impractical for dynamic seed (changing the seed over the time), (3) Impractical if LFSR/PRNG is replaced with TRNG	——
Shift-and-Leak [37]	(1) Finding leaky SFFs which are synthesizable/propagatable, (2) Shifting in state/key into cells, (3) Moving the key into the leaky SFF using shift mode, (4) Observing the key at PO by applying a ATPG-generated pattern	FORTIS [36], R-DFS [40]	(1) Hard to leak the keys if leaky SFFs are very limited (based on the topological sort of the gates), (2) Not applicable if all key storage are placed and mapped ahead of SFFs in the chains	mR-DFS [37], kt-DFS [38], DisORC [42]

[*1] is able to break

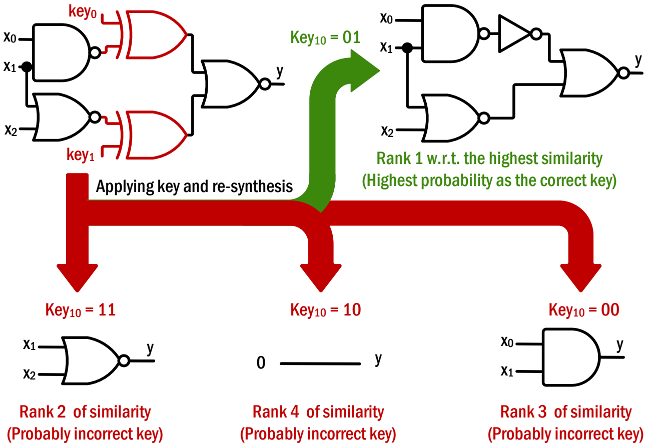


Fig. 26: How Key-Constrained Desynthesis Works [178].

extraction from the constrained designs, (3) key correlation analysis that determines feature weighting algorithm, and how the decision will be made, and finally (4) test time which generate the initial predicted key.

4.3.1.3 Redundancy Attack: The work in [179] proposes a redundancy attack, which is based on the observation that an incorrect key often results in a circuit with significantly more logic redundancy compared to the correct circuit. The redundancy attack determines the likely value of key bits individually by comparing the levels of logic redundancy for each logic value. In the first step, all the key bits are initialized as unspecified until their values are determined. Then the logic value of one key bit is enumerated, and a redundancy identification tool will extract untestable faults for each logic value assignment. The attack assesses the likelihood of a key bit by comparing the changes to the number of untestable faults for a key-bit value of zero and a key-bit value of one. The authors of [179] showed that this attack can recover more than half of the key bits in both RLL and SLL. Similar to previous structural synthesis-based attacks this attack uses key guessing and reduction after constraining. However, they rely on the fact that an incorrect key can modify the circuit's structure in a way that may reduce the manufacturing test coverage. But this reduction should not happen in the original design, and consequently narrow down the key space based on the number of untestable faults per each constraint. Hence, since it is a combination of structural and test-based analysis, it can also be a member of structural ATPG-based attacks (§4.3.2).

4.3.1.4 Topology-Guided Attack: Topology-Guided Attack (TGA) [180], [181] relies on identifying repeated functions for determining the value of a key bit. The attack is based on the observation that the basic functions in a logic cone are generally repeated multiple times in a circuit, such as basic arithmetic function units, shift registers, counters, etc. These functions are denoted as function units. If one or more key gates are placed in an instance of repeated function unit (FU) during the locking of a circuit, the original netlist can be recovered by searching the equivalent function units (EFUs) with all hypothesis keys. For finding the EFUs in a locked netlist, this attack uses an efficient depth-first search (DFS). If a match is found in the netlist, the hypothesis key becomes the actual key bit. For example, in a four-bit ripple carry adder that consists of eight identical one-bit half adders (HA), HA can be treated as a FU. If one of these HA is locked using an XOR gate, an adversary only needs to find an original HA, and then match this with the locked HA to recover the key value.

4.3.2 OL Structural ATPG-based Attacks

ATPG-based attacks are mostly inspired by test concepts to discover the logic locking key values. It is shown that ATPG is a reliable choice for discovering the key values since it is readily available from multiple commercial vendors and it is able to handle large circuits due to its development over the last 50 years. Redundancy attack described previously in §4.3.1.3 is one of the attacks in this sub-group that tries to guess the incorrect key while the manufacturing (stuck-at) test coverage for a constrained re-synthesized netlist is high.

4.3.2.1 Differential Fault Analysis Attack: The differential fault analysis (DFA) attack [182] is an ATPG-guided stuck-at fault-based attack, showing how an adversary can determine the logic locking key by injecting faults at the key registers, which hold the key value during normal operation. In this attack model, the first step is to select an input pattern, that produces complementary results for both fault-free and faulty circuits. The faulty circuit is the same chip, only with a particular fault inject to keep all the key registers or interconnects to faulty value. Note that the selected input pattern must sensitize only one key bit to the primary output(s). To obtain the specific input test patterns, their method relies on stuck-at faults based on constrained ATPG. Then, the input pattern will be applied to (1) fault-free and (2) faulty circuits, and the responses of them will be gathered. The output responses are XORed to find any mismatch and based on the output of the XOR gate the key value can be predicted. The authors of [182] showed that at most $|K|$ test patterns are required to recover the entire secret key of size $|K|$.

4.3.2.2 CLIC: CLIC-A [183], [184] also uses commercial ATPG to recover key-input values in locked combinational and sequential circuits. There are four methods currently included in CLIC-A. The third and fourth methods are oracle-less, meaning they only require the netlist, which will be discussed in this section. The third method, which is applicable to combinational locking and targeting key-dependent faults [183], targets faults that require multiple key inputs for sensitization. To find these faults, ATPG is first performed on the locked netlist with the unsolved key inputs all constrained to do-not-care values (X). Faults that require the key inputs for detection will be reported as an ATPG failure. Each fault in this set of key-dependent faults is targeted using the second round of ATPG. If a test is found, the generated test is stored, along with the fault to be further analyzed for the key value. The test analysis method used for the generated tests differs depending on the lock type. The fourth method included in CLIC-A solves a key sequence from a sequentially locked circuit [184]. The insight used in the fourth method is that there must exist numerous faults that require the key sequence for detection through sequential ATPG. In this method, the sequential circuit ATPG is typically accomplished by first unrolling the circuit to form a combinational mode. Then CLIC-A targets single stuck-at faults on one combinational frame at a time. This method is specifically effective at solving a key from a circuit locked with an entrance FSM (e.g., [13]).

4.3.3 OL Structural ML-based Attacks

A more recent trend has been opened in the area of logic locking that evaluates the utilization of machine-learning (ML) for both defensive and attacking sides. From the attacking point-of-view, in many of these ML-based attacks, unlike seeking for functional recovery, structural analysis through ML has been done aiming to either find the (correct) key that corresponded to the correct structure or retrieve the original structure by removing the transformations introduced by locking [22], [177], [185], [186], [187], [188], [189], [190], [191].

Structural attacks, such as SAIL [185], GNNUnlock [187], and Snapshot [22] aim at discovering the design intent using machine learning. The authors of [185], were motivated based on this observation that obfuscation introduces sparse and local structural changes in a design and the changes are very deterministic. Hence, they proposed the SAIL attack [185], which exposes a vulnerability in logic locking by learning the predictable, localized structural changes that are introduced by the obfuscation process and uses machine learning to learn the deterministic rules applied by commercial CAD tools during synthesis. More precisely, in the SAIL attack, the Pre- and post-resynthesized locked designs are provided as training data to train the Change Prediction Model (CPM) and the Reconstruction Model (RM). Given a netlist subgraph (considering netlist as a graph) extracted near the selected key input, CPM predicts whether a structural change has occurred. If a change is predicted, RM is utilized to locally revert the structural changes after the re-synthesis. SAIL attack is applicable to traditional logic locking techniques (e.g., RLL, FLL, and SLL).

SnapShot [22] is another structural analysis attack on logic locking, which utilizes artificial neural networks to directly predict a key bit value from a locked synthesized gate-level netlist. The authors in [22] claim that SnapShot can be applied to a wider range of schemes, such as a MUX-based locking scheme. Another example of machine learning-based attack is GNNUnlock [187], which leverage graph neural networks (GNNs) that learn the common structural features of the protection logic added by point function techniques, such as SFLL-HD [51], TTLock [50], and Anti-SAT [48]. GNNUnlock employs the following techniques: 1) netlist-to-graph transformation to capture each gate's functionality and connectivity in the gate-level netlist, 2) GNN learning on locked circuits, and 3) Post-processing rectification procedure to rectify any potential misclassifications to enhance the accuracy further and remove the identified protection logic effectively.

More recently, some ML-based structural attacks targeted routing-based locking techniques [190], [191]. In [190], the key-extraction has been formulated as a link prediction problem, and the prediction has been done using a graph neural network (GNN). Link prediction is a generic problem that can be modeled using GNN, and in this attack, since routing locking will hide the connectivity/wiring, this missing connections are modeled as missing link problems. Then by enclosing subgraph extraction, GNN-based link prediction has been performed. Similar approach has been used in [191], in which another MUX-based routing locking is targeted [192], called D-MUX. Although D-MUX is introduced as a MUX-based routing locking resilient against ML-based attack, the work in [191] still shows that link prediction modeling can break this countermeasure as well. The generality of the link prediction model allows the adversary to formulate different logic locking techniques. For routing-based locking, the target links are connections around MUXes, and for other techniques, it could be either gates or other connections.

4.3.4 OL Tampering Attacks

In the IC supply chain, there are potential adversaries who have the capability and access to insert Hardware Trojans into the IC design, which is called tampering attacks. Hardware Trojans, are malicious modifications to the original circuitry inserted by adversaries to exploit secret key information of the design, such as logic locking key. The TAAL attack [193] is based on implanting a hardware Trojan in the netlist. In [193], the authors showed that any logic locking techniques that rely on the stored secret key can be broken by inserting a hardware Trojan. The attacking approach of the TAAL attack is to tamper the locked netlist to extract the secret key information and leak the secret key to an adversary once activated. The authors of [193] present three types of TAAL attacks that extract the secret key differently using hardware Trojans placed at different locations in the netlist. In T1 type TAAL attack, an adversary can extract the key from a locked netlist without knowing the details of the logic locking technique used to protect the circuit as it directly leaks the secret key from the tamper-proof memory. In this attack model, the trigger is constructed using a 3-input AND gate along with an inverter placed before one of the AND gate inputs, and the payload is delivered to the primary output of the circuit using a 2-input MUX. Under normal operation, the multiplexer propagates the correct circuit functionality at the output. Once the Trojan gets activated, the output of AND gate becomes 1, which leads to the extraction of the secret key through the multiplexer at the output. T2 type and T3 type TAAL attacks rely on the activation and propagation of the secret key to the primary output and improve the complexity of detecting an attack.

4.3.5 OL Probing Attacks

Physical attacks, such as electrical probing attacks and optical probing attacks can impose the threat of exposing security-sensitive information to an adversary. For example, the electrical probing attack directly accesses the internal wires of a security-critical module and extracts sensitive information in electronic format. Therefore, electrical probing is considered as a contact-based method. Electrical probing attacks can be classified into front-side probing, which is carried out through the upper metal layers, and back-side probing, which is mounted through the silicon substrate. Attackers usually deploy focused ion beam (FIB) for probing attacks. FIBs use ions at high beam currents to mill a narrow cavity, and get access to the target wire. Active shielding could be considered as a countermeasure against front-side probing attack, in which a shield that carries signals is placed on the top-most metal layer to detect holes milled by FIB; however, it has its limitations [194].

On the other hand, optical probing techniques are often used in back-side probing to capture photon-emission phenomena during transistor switching. By passively receiving and analyzing the photons emitted from a specific transistor, the signal processed by that transistor can be inferred. In addition to photon emission analysis, laser voltage technique (LVX), or electro-optical frequency modulation (EOFM), are also used during back-side attacks. These techniques illuminate the switching transistors and observe the reflected light. The work in [20] shows that optical probing is a threat for logic locking as this method can extract the locking key in a contact-less manner; without using invasive methods, like FIBing or circuit edit, and contact-based method, like electrical probing. With that in mind, security against optical analysis mostly concerns protecting the backside of the chip, such as adding a backside polishing detector [194].

4.3.6 Summary of Oracle-Less (OL) Attacks

Oracle-less attacks on logic locking mostly focus on the evaluation of design specifications after purposefully manipulating the logic locking part. The Manipulating of the logic locking part can be done using different mechanisms: one can apply the constraining of the key value (synthesis-based and ATPG-based), one can insert malicious behavior (tampering-based), on can injecting faults, etc. Also, the specification mostly is related to structural specification, testability specification, or ML-based feature extraction. So, in this breed of attacks, the adversary applies the manipulation, gathers the information about the targeted specifications, and decides/calculates/predicts the key value. On the other side, probing attacks, either electrical or optical, can be done on a design while the key is loaded (or going to be loaded), and no modification or manipulation for observing the logic locking key is required, which makes this model of attack a real threat against all existing logic locking techniques. With the introduction of ML-based and probing-based attacks in recent years, current directions of logic locking faced some changes to be responsive against these tighter and harder threat models. In the following, we will cover some of these directions that require more investigation w.r.t. the existing attack models. Table 14 tries to concisely answer to these *four* questions for oracle-less attacks: (1) How the attack flow works; (2) Which logic locking techniques could be broken using the attack; (3) What is the limitation and challenges of the attack; and (4) What is the existing/potential countermeasure against the attack?

5 What to Expect from Future Studies

Unlike the very first experiment(s) on logic locking that promise the security of design against reverse-engineering, IP piracy, and IC overproduction, we just demonstrate through this survey paper that this proactive countermeasure, regardless of its potential, has been challenged continuously for almost two decades, thereby we are witnessing a non-stop cat-and-mouse game between logic locking countermeasures (defenses) as well as de-obfuscation approaches (attacks). Considering multiple factors, including but not limited to (i) all threats introduced so far on logic locking, (ii) the shortcoming and architectural drawbacks of existing approaches, (iii) getting the benefit of cutting-edge science and technologies applicable at this domain, and (iv) further modernization happening in semiconductor industries, it seems that the introduction of a standalone but comprehensive solution for addressing all the vulnerabilities in logic locking is almost beyond the bounds of possibility.

TABLE 14: Overview of Oracle-Less Attacks.

Attack	Mechanism	Applicable to^{*1}	Limitation & Challenges	Countermeasures
Desynthesis [178]	(1) Applying a random key to the netlist, (2) Re-synthesizing the netlist constrained with the random key, (3) Comparing (structural) the re-synthesized netlist with the raw locked netlist, (4) Guessing the correct key based on the (structural) similarity ratio	RLL [11], FLL [44], SLL [12]	(1) Hard to apply if original gates are combined/twisted with key gates, (2) Depending on the optimizations used in Re-synthesis, it might lead to an incorrect key	Interlock [80], TRLL [42], eFPGA [141], [142]
SWEEP [177], SCOPE [195]	(1) Assigning a key value (0/1) to one key bit, (2) synthesizing w.r.t. the assigned key value, (3) Analyzing the re-synthesized netlist for any design features that are correlated to the correct key value, (4) Indicating the correct key value based on a feature scoring algorithm	Random MUX-based Locking	(1) Low accuracy for large number of key gates (strongly connected MUX-based paths), (3) Complex training phase for larger circuits.	Shielding [196], UNSAIL [197], eFPGA [141], [142]
DFA [182]	(1) Building miter using fault-free circuit and laser-based faulty circuit (fault at key-registers), (2) Finding a DIP using ATPG (sensitize one bit of key), (3) Guessing the key based on the response	Any Locking Technique	(1) Hard to inject fault when backside of die is obstacled using metal coating, (2) More than one laser source is required	——
CLIC-A comb [183]	(1) Finding key-dependent faults via ATPG constrained to don't care values, (2) Targeting each key-dependent fault by ATPG, (3) Analyzing the test generated corresponded to faults for guessing the key value	Primitives (§3.1), point function (§3.2)	(1) Nonscalable for high number of key-dependent faults per cone, (2) Dependent to The availability of don't care	——
CLIC-A seq [184]	(1) Unrolling the netlist to build combinational counterpart for u cycles, (2) Finding the faults and guessing the keys as described in CLIC-A comb	FSM locking [13], [14], [91]	(1) Hard to be scalable for large circuit, (2) Not scalable for faults at deep sequence	DFSSD [93]
Redundancy [179]	(1) Assuming all key values as unspecified, (2) Enumerating logic value of one key bit, (3) Extracting untestable faults for each logic value, (4) Indicating the key bit value based on the number of untestable faults	RLL [11], SLL [12], Point Function + Primitive	(1) Hard to apply if original gates are twisted with key gates, (2) Not applicable to MUX-based and LUT-based locking	Interlock [80], TRLL [42], Shielding [196]
TGA [180]	(1) constructing unit functions (repeated) corresponding to the hypothesis key, (2) match unit functions with exisitng unit functions, (3) guessing the key based on the constructed topology	Primitive locking techniques	(1) Low success rate if unique modules are locked, (2) Highly dependent to the topology and structure of the circuit	MUX/LUT locking, eFPGA [141], [142]
SAIL [185]	(1) Train a {change prediction Model} and {reconstruction} model using pre- and post-resynthesized locked designs, (2) Predicting the changes occurred using the trained ML	XOR-based logic locking	Unsuccessful (1) if parts of logic locking is added manually (post-synthesis), (2) on MUX-based and LUT-based locking	eFPGA [141], [142], UNSAIL [197]
GNNUnlock [187]	(1) Generating a dataset on targeted locking technique, (2) Translating dataset to graph (netlist to graph), (3) Training the graphNN, (4) Testing on targeted netlist using trained graphNN	Point Function	(1) Training set and test set must be identical, (2) Unsuccessful against dynamic nature (re-configurability)	——
Utangle, MuxLink [190], [191]	(1) Formulating the key-extraction as a link prediction problem, (2) Applying the link prediction using GNN, (3) A post-processing to determine correct vs. false links	InterLock [80], D-MUX [192]	Low success rate if (1) the location of locked paths are distributed, (2) paths are less co-dependent	——
TAAL [193]	(1) Inserting T1/2/3 Trojan into the netlist for extracting/propagating the key, (2) Enabling the Trojan after activation	Any locking technique	(1) Hard to insert Trojan if keys are added with no leakage to outputs, (2) Might be detectable by Trojan detection algorithms	——
Probing [20], [194]	(1) Determining the clock frequency, (2) Localizing key-gate registers/storage, (3) Acquiring optical access for probing, (4) Identifying the key loading period, (5) Extracting key value from EOFM/EOP	Any locking technique	(1) Not applicable if randomness is used during initialization, (2) Localising and simultaneous probing is hard to achieve	Nanopyramid [198], Differential Logic [199]

^{*1} is able to break

5.1 Vulnerabilities/Requirements of Logic Locking

In the following, we first describe state-of-the-art yet most concerning issues around logic locking showing why logic locking has its own certain critical setbacks. Then, we list some of the possible directions that might provide appropriate answers to the existing vulnerabilities. The major questions, yet with no answer, are as follows:

(i) *Lack of Formal Model*: The concept of logic locking with locking/unlocking key resembles that of cryptographic primitives with encryption/decryption key. However, since the introduction of logic locking, there exists no formal definition or formulation as the reference showing what exactly a logic locking countermeasure must approach to be considered (provably) secure. For instance, the point function techniques (§3.2) use the notion of **provably secure**. However, it is only against the SAT attack, and all of them are already broken by exploiting structural specification(s) (§4.1.3.5). Similarly, configurable logic and routing techniques (§3.4) propose SAT-hard instances that implicitly build provable resiliency with exponential complexity in terms of BDD analysis. However, few recent studies show their vulnerabilities against other structural specification(s) exploited by machine learning (§4.3.3). It is undeniably clear that formalizing the security of logic locking, and defining the logic locking construction that meets the formal model, has been completely elusive, which results in the introduction and evolution of numerous groups but completely independent and in an insular way.

(ii) *Dynamic/Expanding Nature of Threat Modeling*: The applicability of logic locking, as well as the success of the proposed approaches, are heavily dependent on the correct and accurate definition of the threat model. From oracle-guided to oracle-less, from scan-available to scan-protected, from invasive to non-invasive, and from untrusted foundry to the malicious user at market/field, are some of the notable bullets in the definition of threat model against logic locking. However, over time, the definition[10] might face significant changes that result in invalidating all previous studies. For instance, the notion of **insider threats**, e.g., a rogue employee at a design house, from the design team to verification, integration, etc., or nearly any individual working within any trusted entity but as a malicious actor, can completely invalidate the applicability of existing logic locking techniques, that mostly done at gate-level. In addition, unlike almost all attacks that have been done at the gate-level, with having this insider threat notion, there exists

[10] Here it is all definitions related to IC lifecycle. i.e., from IC specification to disposal.

a possibility of emerging a new thread of attacks on logic locking that have been accomplished at the RTL level, whose main aim is to get the benefit of circuit specification(s) at RTL or any high-level pre-synthesis representation for de-obfuscation purposes. This shows the lack of generality and applicability of many existing logic locking techniques, which makes them discredited in the short term with a new backdoor(s) revealed in the new threat model(s).

(iii) *Cutting-edge Technologies/Devices against Logic Locking*: As described in §4.3.5, an adversary such as a high-end untrusted foundry with cutting-edge technologies and devices, e.g., microprobing station, scanning electron microscope (SEM), and laser scanning microscope (LSM), should be more than capable of extracting the unlocking key from a chip by contact-based electrical or contactless optical probing. This form of attack is not only limited to logic locking, even cryptographic primitives with proven guarantees are subjected to such attacks. In this case, regardless of the logic locking techniques, with having the information of key management infrastructure (§2.2), the adversary can raid it using either FIB and electrical contact-based probing or contactless optical probing from the backside of the chip, and most modern chips do not have any protection mechanism for the backside of the substrate. Although probing-based attacks on logic locking completely undermine the protection/security promised by logic locking, the security of the key management infrastructure has not been evaluated meticulously, and most of the approaches left this part non-investigated by just relying on the assumption of having TPM as a reliable solution with no further details.

(iv) *Machine Learning and its Evolution*: Since 2018, the application of machine learning, specifically graph-based neural networks (GNN), has been drastically increased in the domain of logic locking. These approaches rely on some feature extraction that is more related to the structural specification(s) of the logic-locked circuit. Then, a training model will help to tune the ML (NN) for the specific logic locking targeted for de-obfuscation. Currently, many of the existing logic locking techniques have been broken by a set of ML-based attack models (§4.3.3). These attack models clearly reveal the flaws of existing logic locking techniques, particularly in terms of structural specification(s). Additionally, expanding the threat models and opening new backdoor(s) can also raise the usage of this model of attacks. For instance, assuming the notion of insider threats, and having access to higher abstract(s) of the design, as well as graph-based NN feature extraction can also be done at these higher abstractions making the existing approaches even more prone to another set of ML-based structural analysis.

5.2 Possible Research Opportunities in Logic Locking

Following is some of the possible directions that have been already taken with some shallow investigation. However, to get the highest benefit of logic locking, these items must be considered meticulously:

(i) *Security Evaluation using Formal Models*: Considering that the logic locking research domain is getting bigger, the need for a formal security definition has become inevitable. Some recent studies have evaluated the possibility of defining some security metrics using formalizing the model [51], [200]. However, these models are very limited to the construction of only a specific category (specific threat model), which is combinational construction(s) against the SAT or approximated-based SAT. However, such definitions require more generality and applicability at different threat models and assumptions. Hence, studies that accomplish formal analysis on logic locking that will be followed by the definition of generic and comprehensive security metrics are completely missing, which still needs significant attention by the community. Please note that this process has already been taken in cryptographic primitives, which results in introducing metrics like the indiscernibility of the block ciphers. Although realizing the formalism in logic locking requires a precise definition for what exactly secure logic locking is, which is hard to achieve, it can still help the designers and researchers in the community to pave the road towards what necessarily needs to be focused on.

(ii) *Multi-layer Logic Locking*: This should not come as a surprise, there is no single standalone solution for addressing all the above-mentioned vulnerabilities, meeting all requirements, and addressing all challenges in logic locking. Hence, having a methodology that spans over different stages or layers of design and implementation is one valid solution against all threats. One recent study has shown a multi-layer of security countermeasures against different threats, known as defense-in-depth [194]. The layers can be a defense against (1) existing attacks on logic locking, (2) attacks around design-for-test infrastructure, and (3) contact-based and contactless probing. The notion of multi-layer security can be used for logic locking individually. For instance, the notion of compound logic locking (§3.2.1), or combination of higher-level logic locking with gate-level logic locking (§3.8) are some of the examples of multi-layer logic locking that have been engaged for enhanced security. However, since these compound techniques are introduced to cover and break more attacks (or applicable for a wider range of threat models), they might still suffer from analyses that can lead to breaking them, e.g., compound attacks. So, the Composition of multiple logic locking techniques requires algorithmic procedures about the inter-locking correlation helping the designers to build a more comprehensive multi-layer or compound countermeasures. One intuitive solution for this case is when a primitive logic locking is combined with a scan chain locking/blockage technique, which pushes the complexity towards a two-dimension unrolling-based problem, and the attacker will be limited to only the BMC-based attacks, that hugely suffer from scalability issues in large circuits. But note that, these techniques might reduce the testability coverage. Additionally, over time, with the advances in formal methods and verification tools, the scalability issue might be mitigated in BMC-based attacks.

(iii) *Logic Locking vs. Zero Knowledge Computation*: The dynamic nature of threat modeling due to ever-expanding the threats we face through the IC lifecycle weakens the robustness and resiliency promised by the existing logic locking techniques. For instance, the notion of the insider threats and considering even any individual existing within the design house as an untrusted party reminds the concept of *zero-knowledge-proof* (ZKP) in cryptographic protocols between two parties, in which a *prover* and a *verifier* work on a statement without leaking any extra information that might expose their belongings related to that statement [201]. In logic locking, this is when the designer wants to send the logic-locked design to the integration team, and the designer has no intent to share any information about the design and the logic locking part. In this case, one solution can be enabling logic locking to also serve as a ZKP system. To do this, some requirements must be met: (1) Similar to cryptographic primitives, it requires to support fully indiscernibility, which means that for any individual at any stage of the design, the logic locked design should not provide any additional information compared to the original one. Many of the existing logic locking techniques do not meet these requirements, and they are broken because the locked design, in different ways, can provide hints and additional information to the adversary. One big example is all point-function techniques that have already broken via some structural analyses. Or another example is the high success rate of ML-based attacks that rely more on structural-based feature extractions. So, indiscernibility would be a crucial characteristic of a secure logic locking technique, and if the logic locking meets the indiscernibility, none of the structural analyses can be applicable to it. (2) Logic locking must be implemented as early as possible in the design flow, which is at higher levels of abstraction, i.e., RTL or HLS, to cover a wider range of possible threats. Some recent studies propose high-level logic locking (§3.8), however, they just show how logic locking can be migrated from structural gate-level to behavioral and semantic-level, which can be more helpful for protecting the assets in the design. But, they still leak additional information, which results in being vulnerable to newer attacks, such as SMT [158]. Although the composition of high-level logic locking with some gate-level techniques can also provide robustness against wider threats, there is still the possibility for the adversary to open numerous ways at the higher-level representation of the design to break the logic locking. (3) The logic locking part(s) must be **fine-grained**: for reducing overhead, **uniformly distributed**: to eliminate any analysis or reduction, and **with no dependency** with other locked/original part(s) of the design to guarantee that it is not location-dependent. Some of the existing logic locking techniques meet part(s) of these requirements, but there is still no logic locking that can meet all. For instance, configurable logic and routing techniques add some form of universality that meet these requirements (§3.4), but they still have no full indiscernibility (violating item 1). Also, they are mostly implemented at transistor- or layout-level (violating item 2). eFPGA-based IP redaction, which can be considered as the superset of LUT-based and routing-based techniques, is another group that meets item 1 (and partially item 2), but they still violate item 3, because they are implemented fully coarse-grained with prohibited overhead ($\sim$ 1.5x-3x for even SoC-size circuits).

6 Conclusion

For more than a decade, logic locking has been evaluated tremendously as a proactive IP protection technique against different hardware security threats, specifically IP piracy and IC overproduction. In this survey paper, we holistically reviewed all activities around logic locking, on both defensive and attacking sides. By reviewing the major pros/cons of all logic locking techniques proposed so far, this survey paper can help all researchers, IP vendors, and SoC designers, interested in logic locking to quickly navigate and identify the state-of-the-art in all breeds, either countermeasures or attacks.

References

[1] B. Shakya, M. Tehranipoor, S. Bhunia, and D. Forte, “Introduction to hardware obfuscation: Motivation, methods and evaluation,” in *Hardware Protection through Obfuscation*. Springer, 2017, pp. 3–32.

[2] M. Lapedus, “Week In Review: Manufacturing, Test, and Foundry Challenges,” https://semiengineering.com/week-in-review-manufacturing-test-163/, 2021.

[3] M. Tehranipoor, C. Wang, *Introduction to hardware security and trust*. Springer Science & Business Media, 2011.

[4] M. Rostami, F. Koushanfar, R. Karri *et al.*, “A Primer on Hardware Security: Models, Methods, and Metrics,” *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[5] A. B. Kahng, J. Lach, W. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, “Constraint-based Watermarking Techniques for Design IP protection,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 10, pp. 1236–1252, 2001.

[6] Y. Alkabani and F. Koushanfar, “Active Hardware Metering for Intellectual Property Protection and Security,” in *USENIX Security Symposium*, 2007, pp. 291–306.

[7] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, “Security Analysis of Integrated Circuit Camouflaging,” in *Proceedings of the ACM SIGSAC conference on Computer & communications security*, 2013, pp. 709–720.

[8] J. Lee, M. Tehranipoor, C. Patel, and J. Plusquellic, “Securing Scan Design using Lock and Key Technique,” in *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, 2005, pp. 51–62.

[9] J. Lee, M. Tebranipoor, and J. Plusquellic, “A Low-cost Solution for Protecting IPs against Scan-based Side-channel Attacks,” in *IEEE VLSI Test Symposium (VTS)*, 2006, pp. 6–pp.

[10] J. Lee, M. Tehranipoor, C. Patel, and J. Plusquellic, “Securing Designs against Scan-based Side-channel Attacks,” *IEEE transactions on dependable and secure computing*, vol. 4, no. 4, pp. 325–336, 2007.

[11] J. Roy F. Koushanfar, and I. L. Markov, “EPIC: Ending Piracy of Integrated Circuits,” in *Design, Automation & Test in Europe Conf. (DATE)*, 2008, pp. 1069–1074.

[12] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, “Security Analysis of Logic Obfuscation,” in *Design Automation Conference (DAC)*, 2012, pp. 83–89.

[13] R. S. Chakraborty and S. Bhunia, “HARPOON: An Obfuscation-based SoC Design Methodology for Hardware Protection,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1493–1502, 2009.

[14] A. R. Desai, M. S. Hsiao, C. Wang, L. Nazhandali, and S. Hall, “Interlocking Obfuscation for Anti-Tamper Hardware,” in *Proceedings of the Cyber Security and Information Research Workshop*, 2013, pp. 1–8.

[15] Y. Xie and A. Srivastava, “Delay Locking: Security Enhancement of Logic Locking against IC Counterfeiting,” in *Design Automation Conference (DAC)*, 2017, pp. 1–9.

[16] 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.

[17] D. P. Affairs, “DARPA Selects Teams to Increase Security of Semiconductor Supply Chain,” https://www.darpa.mil/news-events/2020-05-27, 2020.

[18] C. Helfmeier, D. Nedospasov, C. Tarnovsky, T. Krissler, C. Boit, Christian, and J.-P. Seifert, “Breaking and Entering through the Silicon,” in *ACM SIGSAC conference on Computer & Communications Security (CCS)*, 2013, pp. 733–744.

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

[20] M. T. Rahman, S. Tajik, M. S. Rahman, M. Tehranipoor, N. Asadizanjani, "The Key is Left under the Mat: On the Inappropriate Security Assumption of Logic Locking Schemes," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2020, pp. 262–272.

[21] A. Chakraborty, N. G. Jayasankaran, Y. Liu, J. Rajendran, O. Sinanoglu, A. Srivastava, Y. Xie, M. Yasin, M. Zuzak, "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.

[22] D. Sisejkovic, F. Merchant, L. Reimann, H. Srivastava, A. Hallawa, and R. Leupers, "Challenging the Security of Logic Locking Schemes in the Era of Deep Learning: A Neuroevolutionary Approach," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 17, no. 3, pp. 1–26, 2021.

[23] A. Stern, H. Wang, F. Rahman, F. Farahmandi, and M. Tehranipoor, "ACED-IT: Assuring Confidential Electronic Design against Insider Threats in a Zero Trust Environment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[24] N. Weste, D. Harris, *CMOS VLSI design: A Circuits and Systems Perspective*. Pearson Education India, 2015.

[25] J. Rajendran, O. Sinanoglu, and R. Karri, "VLSI testing based security metric for IC camouflaging," in *IEEE International Test Conference (ITC)*, 2013, pp. 1–4.

[26] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "CamoPerturb: Secure IC Camouflaging for Minterm Protection," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 1–8.

[27] M. Li, K. Shamsi, T. Meade, Z. Zhao, B. Yu, Y. Jin, and D. Z. Pan, "Provably Secure Camouflaging Strategy for IC Protection," *IEEE Transactions on CAD of integrated circuits and systems*, vol. 38, no. 8, pp. 1399–1412, 2017.

[28] B. Shakya, H. Shen, M. Tehranipoor, and D. Forte, "Covert Gates: Protecting Integrated Circuits with Undetectable Camouflaging," *IACR Transactions on Cryptographic Hardware and Embedded Systems (CHES)*, pp. 86–118, 2019.

[29] J. Rajendran, O. Sinanoglu, and R. Karri, "Is Split Manufacturing Secure?" in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2013, pp. 1259–1264.

[30] F. Imeson, A. Emtenan, S. Garg, and M. Tripunitara, "Securing Computer Hardware using 3D Integrated Circuit (IC) Technology and Split Manufacturing for Obfuscation," in {*USENIX*} *Security Symposium*, 2013, pp. 495–510.

[31] K. Vaidyanathan, B. Das, E. Sumbul, R. Liu, and L. Pileggi, "Building Trusted ICs using Split Fabrication," in *IEEE international symposium on hardware-oriented security and trust (HOST)*, 2014, pp. 1–6.

[32] K. Xiao, D. Forte, and M. Tehranipoor, "Efficient and Secure Split Manufacturing via Obfuscated Built-in Self-Authentication," in *IEEE International symposium on hardware oriented security and trust (HOST)*, 2015, pp. 14–19.

[33] D. Forte, S. Bhunia, M. Tehranipoor, *Hardware Protection through Obfuscation*. Springer, 2017.

[34] M. Yasin, J. Rajendran, O. Sinanoglu, and R. Karri, "On Improving the Security of Logic Locking," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 35, no. 9, pp. 1411–1424, 2015.

[35] Actel Corporation, "Design Security in Nonvolatile Flash and Antifuse FPGAs - Security Backgrounder," *Technical Report on Quick Logic FPGAs*, 2002.

[36] U. Guin, Q. Shi, D. Forte, and M. Tehranipoor, "FORTIS: A Comprehensive Solution for Establishing Forward Trust for Protecting IPs and ICs," *ACM transactions on design automation of electronic systems (TODAES)*.

[37] N. Limaye, A. Sengupta, M. Nabeel, and O. Sinanoglu, "Is Robust Design-for-Security Robust Enough? Attack on Locked Circuits with Restricted Scan Chain Access," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[38] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "On Designing Secure and Robust Scan Chain for Protecting Obfuscated Logic," in *Great Lakes Symposium on VLSI (GLSVLSI)*, 2020, p. 217–222.

[39] U. Guin, Z. Zhou, and A. Singh, "A Novel Design-for-Security (DFS) Architecture to Prevent Unauthorized IC Overproduction," in *VLSI Test Symposium (VTS)*, 2017, pp. 1–6.

[40] U. Guin, Z. Zhou, and A. Singh, "Robust Design-for-Security Architecture for Enabling Trust in IC Manufacturing and Test," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 5, pp. 818–830, 2018.

[41] K. Z. Azar, F. Farahmand, H. M. Kamali, S. Roshanisefat, H. Homayoun, W. Diehl, K. Gaj, and A. Sasan, "COMA: Communication and Obfuscation Management Architecture," in *Int'l Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2019, pp. 181–195.

[42] 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 CAD of Integrated Circuits and Systems*, vol. 40, no. 9, pp. 1740–1753, 2020.

[43] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "From Cryptography to Logic Locking: A Survey on the Architecture Evolution of Secure Scan Chains," *IEEE Access*, vol. 9, pp. 73 133–73 151, 2021.

[44] J. Rajendran, H. Zhang, C. Zhang, G. S. Rose, Y. Pino, O. Sinanoglu, and R. Karri, "Fault analysis-based logic encryption," *IEEE Transactions on Computers*, vol. 64, no. 2, pp. 410–424, 2015.

[45] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the Security of Logic Encryption Algorithms," in *Int'l Symp. on Hardware Oriented Security and Trust (HOST)*, 2015, pp. 137–143.

[46] M. El Massad, S. Garg, M. Tripunitara, "Integrated Circuit (IC) Decamouflaging: Reverse Engineering Camouflaged ICs within Minutes," in *NDSS*, 2015, pp. 1–14.

[47] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu, "SARLock: SAT Attack Resistant Logic Locking," in *Hardware Oriented Security and Trust (HOST) Symposium*, 2016, pp. 236–241.

[48] Y. Xie and A. Srivastava, "Mitigating SAT Attack on Logic Locking," in *IACR Conference on Cryptographic Hardware and Embedded Systems (CHES)*, 2016, pp. 127–146.

[49] 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.

[50] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu, "TTLock: Tenacious and Traceless Logic Locking," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2017, pp. 166–166.

[51] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-Secure Logic Locking: From Theory to Practice," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017, pp. 1601–1618.

[52] 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.

[53] J. Zhou and X. Zhang, "Generalized SAT-Attack-Resistant Logic Locking," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2581–2592, 2021.

[54] Y. Liu, M. Zuzak, Y. Xie, A. Chakraborty, and A. Srivastava, "Strong Anti-SAT: Secure and Effective Logic Locking," in *International Symposium on Quality Electronic Design (ISQED)*, 2020, pp. 199–205.

[55] B. Shakya, X. Xu, M. Tehranipoor, and D. Forte, "CASlock: A Security-Corruptibility Trade-off Resilient Logic Locking Scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 175–202, 2020.

[56] M. Yasin, C. Zhao, and J. Rajendran, "SFLL-HLS: Stripped-Functionality Logic Locking meets High-Level Synthesis," in *International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–4.

[57] A. Rezaei, Y. Shen, and H. Zhou, "Rescuing Logic Encryption in Post-SAT Era by Locking & Obfuscation," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, pp. 13–18.

[58] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, Y. Jin, "Cyclic Obfuscation for Creating SAT-unresolvable Circuits," in *Proceedings of the on Great Lakes Symposium on VLSI (GLSVLSI)*, 2017, pp. 173–178.

[59] M. D. Riedel and J. Bruck, "The Synthesis of Cyclic Combinational Circuits," in *Design Automation Conference (DAC)*, 2003, pp. 163–168.

[60] S. Roshanisefat, H. M. Kamali, A. Sasan, "SRCLock: SAT-resistant cyclic logic locking for protecting the hardware," in *Proceedings of the on Great Lakes Symposium on VLSI (GLSVLSI)*, 2018, pp. 153–158.

[61] A. Rezaei, Y. Li, Y. Shen, S. Kong, and H. Zhou, "CycSAT-unresolvable Cyclic Logic Encryption using Unreachable States,"

in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2019, pp. 358–363.

[62] S. Roshanisefat, H. M. Kamali, H. Homayoun, and A. Sasan, "SAT-Hard Cyclic Logic Obfuscation for Protecting the IP in the Manufacturing Supply Chain," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 4, pp. 954–967, 2020.

[63] A. Rezaei, Y. Shen, S. Kong, J. Gu, and H. Zhou, "Cyclic Locking and Memristor-based Obfuscation against CycSAT and inside Foundry Attacks," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2018, pp. 85–90.

[64] X. Yang, P. Chen, H. Chiang, C. Lin, Y. Chen, C. Wang, "LOOPLock 2.0: An Enhanced Cyclic Logic Locking Approach," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 41, no. 1, pp. 29–34, 2021.

[65] H. Chiang, Y. Chen, D. Ji, X. Yang, C. Lin, C. Wang, "LOOPLock: Logic Optimization-Based Cyclic Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2178–2191, 2019.

[66] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC Piracy using Reconfigurable Logic Barriers," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 66–75, 2010.

[67] H. M. Kamali, K. Z. Azar, K. Gaj, H. Homayoun, and A. Sasan, "LUT-Lock: A Novel LUT-based Logic Obfuscation for FPGA-bitstream and ASIC-hardware Protection," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2018, pp. 405–410.

[68] G. Kolhe, H. M. Kamali, M. Naicker, T. Sheaves, H. Mahmoodi, S. Manoj PD, H. Homayoun, S. Rafatirad, and A. Sasan, "Security and Complexity Analysis of LUT-based Obfuscation: From Blueprint to Reality," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[69] S. D. Chowdhury, G. Zhang, Y. Hu, P. Nuzzo, "Enhancing SAT-attack resiliency and cost-effectiveness of reconfigurable-logic-based circuit obfuscation," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[70] V. Betz and J. Rose, "FPGA Routing Architecture: Segmentation and Buffering to Optimize Speed and Density," in *Proceedings of the ACM/SIGDA Int'l Symposium on Field Programmable Gate Arrays (FPGA)*, 1999, pp. 59–68.

[71] C. J. Alpert, D. Mehta, and S. S. Sapatnekar, *Handbook of Algorithms for Physical Design Automation*. CRC press, 2008.

[72] S. Bose, "Methods and Systems for Placement and Routing," 2012, uS Patent 8,332,793.

[73] Y. Wang, P. Chen, J. Hu, and J. Rajendran, "Routing Perturbation for Enhanced Security in Split Manufacturing," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017, pp. 605–510.

[74] A. Sengupta, S. Patnaik, J. Knechtel, M. Ashraf, S. Garg, and O. Sinanoglu, Ozgur, "Rethinking split manufacturing: An information-theoretic approach with secure layout techniques," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 329–326.

[75] S. Patnaik, M. Ashraf, H. Li, J. Knechtel, and O. Sinanoglu, Ozgur, "Concerted Wire Lifting: Enabling Secure and Cost-effective Split Manufacturing," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 41, no. 2, pp. 266–280, 2021.

[76] K. Shamsi, M. Li, D. Z. Pan, Y. Jin, "Cross-lock: Dense layout-level interconnect locking using cross-bar architectures," in *Proceedings of the on Great Lakes Symposium on VLSI (GLSVLSI)*, 2018, pp. 147–152.

[77] 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 Design Automation Conference (DAC)*, 2019, p. 89.

[78] S. Patnaik, M. Ashraf, O. Sinanoglu, and J. Knechtel, "Obfuscating the Interconnects: Low-cost and Resilient Full-chip Layout Camouflaging," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4466–4481, 2020.

[79] J. Sweeney, M. Heule, and L. Pileggi, "Modeling techniques for logic locking," in *International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.

[80] H. M. Kamali, K. Z. Azar, H. Homayoun, A. Sasan, "InterLock: An Intercorrelated Logic and Routing Locking," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2020, pp. 1–9.

[81] A. Saha, S. Saha, S. Chowdhury, D. Mukhopadhyay, and B. Bhattacharya, "Lopher: SAT-Hardened Logic Embedding on Block Ciphers," in *Design Automation Conference (DAC)*, 2020, pp. 1–6.

[82] W. Zeng, A. Davoodi, R. O. Topaloglu, "ObfusX: routing obfuscation with explanatory analysis of a machine learning attack," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2021, pp. 548–554.

[83] D. Zhang, M. He, X. Wang, and M. Tehranipoor, "Dynamically Obfuscated Scan for Protecting IPs against Scan-based Attacks throughout Supply Chain," in *VLSI Test Symposium (VTS)*, 2017, pp. 1–6.

[84] R. Karmakar, S. Chatopadhyay, and R. Kapur, "Encrypt Flip-Flop: A Novel Logic Encryption Technique for Sequential Circuits," *arXiv preprint arXiv:1801.04961*, 2018.

[85] R. Karmakar, H. Kumar, and S. Chattopadhyay, "Efficient Key-gate Placement And Dynamic Scan Obfuscation Towards Robust Logic Encryption," *IEEE Transactions on Emerging Topics in Computing*, 2019.

[86] S. Potluri, A. Aysu, A. Kumar, "Seql: Secure Scan-locking for IP Protection," in *International Symposium on Quality Electronic Design (ISQED)*, 2020, pp. 7–13.

[87] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "SCRAMBLE: The state, connectivity and routing augmentation model for building logic encryption," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2020, pp. 153–159.

[88] M. S. Rahman, A. Nahiyan, F. Rahman, S. Fazzari, K. Plaks, F. Farahmandi, D. Forte, and M. Tehranipoor, "Security Assessment of Dynamically Obfuscated Scan Chain against Oracle-guided Attacks," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 26, no. 4, pp. 1–27, 2021.

[89] 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.

[90] F. Koushanfar, "Active Hardware Metering by Finite State Machine Obfuscation," in *Hardware Protection through Obfuscation*, 2017, pp. 161–187.

[91] J. Dofe and Q. Yu, "Novel Dynamic State-Deflection Method for Gate-Level Design Obfuscation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 273–285, 2018.

[92] T. Meade, Z. Zhao, S. Zhang, D. Z. Pan, and Y. Jin, "Revisit Sequential Logic Obfuscation: Attacks and Defenses," in *IEEE Int'l Symp. on Circuits and Systems (ISCAS)*, 2017, pp. 1–4.

[93] S. Roshanisefat, H. M. Kamali, K. Z. Azar, S. M. P. Dinakarrao, N. Karimi, H. Homayoun, and A. Sasan, "DFSSD: Deep Faults and Shallow State Duality, A Provably Strong Obfuscation Solution for Circuits with Restricted Access to Scan Chain," in *VLSI Test Symposium (VTS)*, 2020, pp. 1–6.

[94] L. Li and A. Orailoglu, "JANUS: Boosting logic obfuscation scope through reconfigurable FSM synthesis," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2021, pp. 1–11.

[95] L. Li and A. Orailoglu, "JANUS-HD: Exploiting FSM Sequentiality and Synthesis Flexibility in Logic Obfuscation to Thwart SAT Attack While Offering Strong Corruption," in *Design, Automation & Test in Europe Conf. (DATE)*, 2022, pp. 1–6.

[96] M. Fyrbiak, S. Wallat, J. Déchelotte, N. Albartus, S. Böcker, R. Tessier, and C. Paar, "On the Difficulty of FSM-based Hardware Obfuscation," *IACR Trans. on Crypto Hardware and Embedded Systems (TCHES)*, pp. 293–330, 2018.

[97] G. Zhang, B. Li, B. Yu, D. Z. Pan, and U. Schlichtmann, "Timing-Camouflage: Improving Circuit Security against Counterfeiting by Unconventional Timing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 91–96.

[98] M. Alam, S. Ghosh, S. Hosur, "TOIC: Timing Obfuscated Integrated Circuits," in *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, 2019, pp. 105–110.

[99] J. Sweeney, V. Zackriya, V S. Pagliarini, and L. Pileggi, Lawrence, "Latch-Based Logic Locking," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2020, pp. 132–141.

[100] K. Z. Azar, H.M. Kamali, S. Roshanisefat, H. Homayoun, C. Sotiriou, and A. Sasan, "Data Flow Obfuscation: A New Paradigm for Obfuscating Circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. v, no. n, pp. 1–14, 2021.

[101] M. S. Rahman, R. Guo, H. M. Kamali, F. Rahman, F. Farahmandi, M. Abdel-Moneum, and M. Tehranipoor, "O'Clock: Lock the Clock via Clock-gating for SoC IP Protection," in *Design Automation Conference (DAC)*, 2022, pp. 1–6.

[102] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "SMT Attack: Next Generation Attack on Obfuscated Circuits with Capabilities and Performance Beyond the SAT Attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, pp. 97–122, 2019.

[103] J. Knechtel, "Hardware Security for and beyond CMOS Technology: An Overview on Fundamentals, Applications, and Challenges," in *International Symposium on Physical Design (ISPD)*, 2020, pp. 75–86.

[104] X. Fong, Y. Kim, K. Yogendra, D. Fan, A. Sengupta, A. Raghunathan, and K. Roy, "Spin-Transfer Torque Devices for Logic and Memory: Prospects and Perspectives," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 1–22, 2015.

[105] A. Makarov, T. Windbacher, V. Sverdlov, S. Selberherr, "CMOS-Compatible Spintronic Devices: a Review," *Semiconductor Science and Technology*, vol. 31, no. 11, p. 113006, 2016.

[106] S. Baek, K. Park, D. Kil, Y. Jang, J. Park, K. Lee, and B. Park, "Complementary Logic Operation based on Electric-Field Controlled Spin-Orbit Torques," *Nature Electronics*, vol. 1, no. 7, pp. 398–403, 2018.

[107] Q. Alasad, J. Yuan, and D. Fan, , "Leveraging All-Spin Logic to Improve Hardware Security," in *Great Lakes Symposium on VLSI (GLSVLSI)*, 2017, pp. 491–494.

[108] T. Winograd, H. Salmani, H. Mahmoodi, K. Gaj, and H. Homayoun, "Hybrid STT-CMOS designs for reverse-engineering prevention," in *Design Automation Conference (DAC)*, 2016, pp. 1–6.

[109] J. Yang, X. Wang, Q. Zhou, Z. Wang, H. Li, Y. Chen, W. Zhao, "Exploiting Spin-orbit Torque Devices as Reconfigurable Logic for Circuit Obfuscation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 1, pp. 57–69, 2018.

[110] G. Kolhe, S. M. PD, S. Rafatirad, H. Mahmoodi, A. Sasan, and H. Homayoun, "On custom lut-based obfuscation," in *Great Lakes Symposium on VLSI (GLSVLSI)*, 2019, pp. 477–482.

[111] S. Patnaik, N. Rangarajan, J. Knechtel, O. Sinanoglu, and S. Rakheja, "Advancing Hardware Security using Polymorphic and Stochastic Spin-Hall Effect Devices," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 97–102.

[112] N. Rangarajan, S. Patnaik, J. Knechtel, R. Karri, O. Sinanoglu, S. Rakheja, "Opening the Doors to Dynamic Camouflaging: Harnessing the Power of Polymorphic Devices," *IEEE Transactions on Emerging Topics in Computing*, 2020.

[113] S. Kvatinsky, G. Satat, N. Wald, E. Friedman, A. Kolodny, and U. Weiser, "Memristor-based Material Implication (IMPLY) Logic: Design Principles and Methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 10, pp. 2054–2066, 2013.

[114] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. Friedman, A. Kolodny, and U. Weiser, "MAGIC—Memristor-Aided Logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[115] A. Rezaei, J. Gu, H. Zhou, "Hybrid Memristor-CMOS Obfuscation against Untrusted Foundries," in *EEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2019, pp. 535–540.

[116] Z. Chen, D. Farmer, S. Xu, R. Gordon, P. Avouris, J. Appenzeller, "Externally Assembled Gate-all-around Carbon Nanotube Field-effect Transistor," *IEEE electron device letters*, vol. 29, no. 2, pp. 183–185, 2008.

[117] A. Franklin, M. Luisier, S. Han, G. Tulevski, C. Breslin, L. Gignac, M. Lundstrom, and W. Haensch, "Sub-10 nM Carbon Nanotube Transistor," *Nano letters*, vol. 12, no. 2, pp. 758–762, 2012.

[118] A. Todri-Sanial, J. Dijon, and A. Maffucci, *Carbon Nanotubes for Interconnects*. Springer, 2017.

[119] T. Mikolajick, and A. Heinzig, J. Trommer, T. Baldauf, and W. Weber, "The RFET—A Reconfigurable Nanowire Transistor and its Application to Novel Electronic Circuits and Systems," *Semiconductor Science and Technology*, vol. 32, no. 4, p. 043001, 2017.

[120] J. Colinge, A. Kranti, R. Yan, C. Lee, I. Ferain, R. Yu, N. Akhavan, and P. Razavi, "Junctionless Nanowire Transistor (JNT): Properties and Design Guidelines," *Solid-State Electronics*, vol. 65, pp. 33–37, 2011.

[121] Y. Bi, K. Shamsi, J. Yuan, P. Gaillardon, G. Micheli, X. Yin, X. Hu, M. Niemier, and Y. Jin, "Emerging Technology-based Design of Primitives for Hardware Security," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 1, pp. 1–19, 2016.

[122] Q. Alasad, J. Yuan, and Y. Bi, "Logic locking using hybrid CMOS and emerging SiNW FETs," *Electronics*, vol. 6, no. 3, p. 69, 2017.

[123] Q. Alasad and J. Yuan, "Logic Obfuscation against IC Reverse Engineering attacks using PLGs," in *2017 IEEE International Conference on Computer Design (ICCD)*, 2017, pp. 341–344.

[124] F. Rahman, B. Shakya, X. Xu, D. Forte, and M. Tehranipoor, "Security beyond CMOS: Fundamentals, Applications, and Roadmap," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 12, pp. 3420–3433, 2017.

[125] N. Jayasankaran, A. Borbon, E. Sanchez-Sinencio, J. Hu, and J. Rajendran, "Towards Provably-Secure Analog and Mixed-signal Locking against Overproduction," in *International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8.

[126] J. Leonhard, M. Yasin, S. Turk, M. Nabeel, M. Louërat, R. Chotin-Avot, H. Aboushady, O. Sinanoglu, and H. Stratigopoulos, "MixLock: Securing mixed-signal circuits via logic locking," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 84–89.

[127] K. Juretus, R. Venugopal, and I. Savidis, "Securing Analog Mixed-signal Integrated Circuits through Shared Dependencies," in *Great Lakes Symposium on VLSI (GLSVLSI)*, 2019, pp. 483–488.

[128] M. Elshamy, A. Sayed, M. Louërat, A. Rhouni, H. Aboushady, and H. Stratigopoulos, "Securing programmable analog ICs against piracy," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, pp. 61–66.

[129] J. Leonhard, A. Sayed, M. Louërat, MH. Aboushady, H. Stratigopoulos, "Analog and mixed-signal IC security via sizing camouflaging," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 5, pp. 822–835, 2020.

[130] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "ChaoLock: Yet another SAT-hard logic locking using chaos computing," in *International Symposium on Quality Electronic Design (ISQED)*, 2021, pp. 387–394.

[131] J. Leonhard, N. Limaye, S. Turk, A. Sayed, A. Rizo, H. Aboushady, O. Sinanoglu, and H. Stratigopoulos, "Digitally-assisted mixed-signal circuit security," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[132] W. Ditto, A. Miliotis, K. Murali, S. Sinha, and M. Spano, "Chaogates: Morphing Logic Gates that Exploit Dynamical Patterns," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 20, no. 3, p. 037107, 2010.

[133] C. Pilato, F. Regazzoni, R. Karri, and S. Garg, "TAO: Techniques for algorithm-level obfuscation during high-level synthesis," in *Design Automation Conference (DAC)*, 2018, pp. 1–6.

[134] C. Pilato, A. Chowdhury, D. Sciuto, S. Garg, and R. Karri, "ASSURE: RTL locking against an Untrusted Foundry," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 7, pp. 1306–1318, 2021.

[135] M. Zuzak, Y. Liu, and A. Srivastava, "A Resource Binding Approach to Logic Obfuscation," in *Design Automation Conference (DAC)*, 2021, pp. 235–240.

[136] R. Muttaki, R. Mohammadivojdan, M. Tehranipoor, and F. Farahmandi, "HLock: Locking IPs at the High-Level Language," in *Design Automation Conference (DAC)*, 2021, pp. 79–84.

[137] N. Limaye, A. Chowdhury, C. Pilato, M. Nabeel, O. Sinanoglu, S. Garg, and R. Karri, "Fortifying RTL Locking Against Oracle-Less (Untrusted Foundry) and Oracle-Guided Attacks," in *Design Automation Conference (DAC)*, 2021, pp. 91–96.

[138] C. Karfa, T. Khader, Y. Nigam, R. Chouksey, and R. Karri, "HOST: HLS Obfuscations against SMT ATtack," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 32–37.

[139] L. Collini and C. Pilato, "A Composable Design Space Exploration Framework to Optimize Behavioral Locking," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 1–6.

[140] G. Takhar, R. Karri, C. Pilato, and S. Roy, "HOLL: Program Synthesis for Higher Order Logic Locking," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2022.

[141] B. Hu, J. Tian, M. Shihab, G. Reddy, W. Swartz, Y. Makris, B. C. Schaefer, and C. Sechen, "Functional Obfuscation of Hardware Accelerators through Selective Partial Design Extraction onto an Embedded FPGA," in *Great Lakes Symposium on VLSI (GLSVLSI)*, 2019, pp. 171–176.

[142] P. Mohan, O. Atli, J. Sweeney, O. Kibar, L. Pileggi, and K. Mai, "Hardware Redaction via Designer-Directed Fine-Grained eFPGA Insertion," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 1186–1191.

[143] J. Bhandari, A. Moosa, B. Tan, C. Pilato, G. Gore, X. Tang, S. Temple, P. Gaillardon, R. Karri, "Exploring eFPGA-based Redaction for IP Protection," in *International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[144] J. Bhandari, A. Moosa, B. Tan, C. Pilato, G. Gore, X. Tang, S. Temple, P. Gaillardo, and R. Karri, "Not All Fabrics Are Created Equal: Exploring eFPGA Parameters For IP Redaction," *arXiv preprint arXiv:2111.04222*, 2021.

[145] M. Bushnell and V. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Springer Science & Business Media, 2004, vol. 17.

[146] S. M. Plaza and I. L. Markov, "Solving the Third-shift Problem in IC Piracy with Test-aware Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 6, pp. 961–971, 2015.

[147] 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 (HOST)*, 2017, pp. 95–100.

[148] Y. Shen and H. Zhou, "Double-Dip: Re-evaluating Security of Logic Encryption Algorithms," in *Proceedings of the on Great Lakes Symposium on VLSI (GLSVLSI)*, 2017, pp. 179–184.

[149] X. Xu, B. Shakya, M. Tehranipoor, D. Forte, "Novel Bypass Attack and BDD-based Tradeoff Analysis against all Known Logic Locking attacks," in *CHES*, 2017, pp. 189–210.

[150] Y. Shen, A. Rezaei, and H. Zhou, "SAT-based Bit-flipping Attack on Logic Encryptions," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 629–632.

[151] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal Attacks on Logic Locking and Camouflaging Techniques," *IEEE Transactions on Emerging Topics in Computing*, 2017.

[152] N. Limaye, S. Patnaik, and O. Sinanoglu, "Fa-SAT: Fault-aided SAT-based Attack on Compound Logic Locking Techniques," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 1166–1171.

[153] 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.

[154] Y. Shen, Y. Li, A. Rezaei, S. Kong, D. Dlott, and H. Zhou, "BeSAT: Behavioral SAT-based Attack on Cyclic Logic Encryption," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2019, pp. 657–662.

[155] K. Shamsi, D. Z. Pan, and Y. Jin, "IcySAT: Improved SAT-based Attacks on Cyclic Locked Circuits," in *Int'l Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–7.

[156] A. Chakraborty, Y. Liu, and A. Srivastava, "TimingSAT: Timing Profile Embedded SAT Attack," in *International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–6.

[157] N. G. Jayasankaran, A. S. Borbon, A. Abuellil, E. Sánchez-Sinencio, J. Hu, and J. Rajendran, "Breaking Analog Locking Techniques via Satisfiability Modulo Theories," in *IEEE International Test Conference (ITC)*, 2019, pp. 1–10.

[158] C. Karfa, R. Chouksey, C. Pilato, S. Garg, and R. Karri, "Is Register Transfer Level Locking Secure?" in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, pp. 550–555.

[159] 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 (ICCAD)*, 2020, pp. 1–9.

[160] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Security Analysis of Anti-SAT," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017, pp. 342–347.

[161] D. Sirone and P. Subramanyan, "Functional Analysis Attacks on Logic Locking," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2514–2527, 2020.

[162] A. Sengupta, N. Limaye, and O. Sinanoglu, "Breaking CAS-Lock and Its Variants by Exploiting Structural Traces," *Cryptology ePrint Archive*, 2021.

[163] Z. Han, M. Yasin, and J. Rajendran, "Does logic locking work with {EDA} Tools?" in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1055–1072.

[164] N. Limaye, S. Patnaik, and O. Sinanoglu, "Valkyrie: Vulnerability Assessment Tool and Attack for Provably-Secure Logic Locking Techniques," *IEEE Transactions on Information Forensics and Security*, 2022.

[165] M. El Massad, S. Garg, and M. Tripunitara, "Reverse Engineering Camouflaged Sequential Circuits without Scan Access," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2017, pp. 33–40.

[166] K. Shamsi, M. Li, D. Z. Pan, and Y. Jinr, "KC2: Key-condition Crunching for Sequential Circuit Deobfuscation," in *Design, Automation & Test in Europe Conference (DATE)*, 2019, pp. 534–539.

[167] S. Roshanisefat, H. M. Kamali, H. Homayoun, A. Sasan, "RANE: An Open-Source Formal De-obfuscation Attack for Reverse Engineering of Logic Encrypted Circuits," in *Great Lakes Symposium on VLSI (GLSVLSI)*, 2021, pp. 221–228.

[168] K. Z. Azar, H. M. Kamali, F. Farahmandi, M. Tehranipoor, "Warm Up before Circuit De-obfuscation? An Exploration through Bounded-Model-Checkers," in *International Symposium on Hardware Oriented Security and Trust (HOST)*, 2022, pp. 1–4.

[169] Y. Shi, C. W. Ting, B. H. Gwee, Y. Ren, "A Highly Efficient Method for Extracting FSMs from Flattened Gate-level Netlist," in *IEEE Int'l Symposium on Circuits and Systems (ISCAS)*, 2010, pp. 2610–2613.

[170] R. Tarjan, "Depth-First Search and Linear Graph Algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

[171] T. Meade *et al.*, "Netlist Reverse Engineering for High-Level Functionality Reconstruction," in *Asia and South Pacific Design Automation Conf. (ASP-DAC)*, 2016, pp. 655–660.

[172] Y. Hu, Y. Zhang, K. Yang, D. Chen, P. A. Beerel, and P. Nuzzo, "Fun-SAT: Functional Corruptibility-Guided SAT-Based Attack on Sequential Logic Encryption," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2021, pp. 1–11.

[173] A. Saha, H. Banerjee, R. S. Chakraborty, D. Mukhopadhyay, "ORACALL: An Oracle-based Attack on Cellular Automata guided Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 12, pp. 2445–2454, 2021.

[174] R. Karmakar, S. S. Jana, and S. Chattopadhyay, "A Cellular Automata guided Obfuscation Strategy for Finite-State-Machine Synthesis," in *Design Automation Conference (DAC)*, 2019, pp. 1–6.

[175] L. Alrahis, M. Yasin, N. Limaye, H. Saleh, B. Mohammad, M. Alqutayri, and O. Sinanoglu, "Scansat: Unlocking Static and Dynamic Scan Obfuscation," *IEEE Transactions on Emerging Topics in Computing*, 2019.

[176] N. Limaye and O. Sinanoglu, "DynUnlock: Unlocking Scan Chains Obfuscated using Dynamic Keys," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, pp. 270–273.

[177] A. Alaql, D. Forte, and S. Bhunia, "Sweep to the Secret: A Constant Propagation Attack on Logic Locking," in *Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, 2019, pp. 1–6.

[178] M. El Massad, J. Zhang, S. Garg, and M. V. Tripunitara, "Logic locking for Secure Outsourced Chip Fabrication: A New Attack and Provably Secure Defense Mechanism," *arXiv preprint arXiv:1703.10187*, 2017.

[179] L. Li and A. Orailoglu, "Piercing Logic Locking Keys through Redundancy Identification," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 540–545.

[180] Y. Zhang, P. Cui, Z. Zhou, and U. Guin, "TGA: An Oracle-less and Topology-guided Attack on Logic Locking," in *ACM Workshop on Attacks and Solutions in Hardware Security Workshop*, 2019, pp. 75–83.

[181] Y. Zhang, A. Jain, P. Cui, Z. Zhou, and U. Guin, "A Novel Topology-guided Attack and its Countermeasure towards Secure Logic Locking," *Journal of Cryptographic Engineering*, vol. 11, no. 3, pp. 213–226, 2021.

[182] A. Jain, T. Rahman, and U. Guin, "Atpg-guided Fault Injection Attacks on Logic Locking," *arXiv preprint arXiv:2007.10512*, 2020.

[183] D. Duvalsaint, X. Jin, B. Niewenhuis, and R. D. Blanton, "Characterization of Locked Combinational Circuits via ATPG," in *2019 IEEE International Test Conference (ITC)*, 2019, pp. 1–10.

[184] D. Duvalsaint, Z. Liu, A. Ravikumar, and R. D. Blanton, "Characterization of Locked Sequential Circuits via ATPG," in *2019 IEEE International Test Conference in Asia (ITC-Asia)*, 2019, pp. 97–102.

[185] P. Chakraborty, J. Cruz, and S. Bhunia, "SAIL: Machine Learning guided Structural Analysis Attack on Hardware Obfuscation," in *2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2018, pp. 56–61.

[186] P. Chakraborty, J. Cruz, and S. Bhunia, "SURF: Joint structural functional attack on logic locking," in *International Symposium on Hardware Oriented Security and Trust (HOST)*, 2019, pp. 181–190.

[187] L. Alrahis, S. Patnaik, F. Khalid, M. Hanif, H. Saleh, M. Shafique, and O. Sinanoglu, Ozgur, "GNNUnlock: Graph Neu-

ral Networks-based Oracle-less Unlocking Scheme for Provably Secure Logic Locking," *arXiv preprint arXiv:2012.05948*, 2020.

[188] Z. Chen, L. Zhang, G. Kolhe, H. M. Kamali, S. Rafatirad, S. M. P. Dinakarrao, H. Homayoun, C.-T. Lu, and L. Zhao, "Deep Graph Learning for Circuit Deobfuscation," *Frontiers in big Data*, vol. 4, 2021.

[189] L. Alrahis, S. Patnaik, M. Shafique, and O. Sinanoglu, "OMLA: An Oracle-less Machine Learning-based Attack on Logic Locking," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2021.

[190] L. Alrahis, S. Patnaik, M. A. Hanif, M. Shafique, and O. Sinanoglu, "UNTANGLE: Unlocking Routing and Logic Obfuscation Using Graph Neural Networks-based Link Prediction," in *International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[191] L. Alrahis, S. Patnaik, M. Shafique, and O. Sinanoglu, "MuxLink: Circumventing Learning-Resilient MUX-Locking Using Graph Neural Network-based Link Prediction," *arXiv preprint arXiv:2112.07178*, 2021.

[192] D. Sisejkovic, F. Merchant, L. Reimann, and R. Leupers, "Deceptive Logic Locking for Hardware Integrity Protection against Machine Learning Attacks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[193] A. Jain, Z. Zhou, and U. Guin, "TAAL: tampering attack on any key-based logic locked circuits," *arXiv preprint arXiv:1909.07426*, 2019.

[194] M. T. Rahman, M. S. Rahman, H. Wang, S. Tajik, W. Khalil, F. Farahmandi, D. Forte, N. Asadizanjani, and M. Tehranipoor, "Defense-in-depth: A Recipe for Logic Locking to Prevail," *Integration*, vol. 72, pp. 39–57, 2020.

[195] A. Alaql, M. M. Rahman, and S. Bhunia, "SCOPE: Synthesis-based Constant Propagation Attack on Logic Locking," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 8, pp. 1529–1542, 2021.

[196] L. Li and A. Orailoglu, "Shielding logic locking from redundancy attacks," in *IEEE VLSI Test Symposium (VTS)*, 2019, pp. 1–6.

[197] L. Alrahis, S. Patnaik, J. Knechtel, H. Saleh, B. Mohammad, M. Al-Qutayri, and O. Sinanoglu, "UNSAIL: Thwarting oracle-less machine learning attacks on logic locking," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2508–2523, 2021.

[198] H. Shen, N. Asadizanjani, M. Tehranipoor, and D. Forte, "Nanopyramid: An Optical Scrambler against Backside Probing Attacks," in *International Symposium for Testing and Failure Analysis (ISTFA)*, 2018, p. 280.

[199] S. Parvin, T. Krachenfels, S. Tajik, J.-P. Seifert, F. S. Torres, and R. Drechsler, "Toward Optical Probing Resistant Circuits: A Comparison of Logic Styles and Circuit Design Techniques," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2022, pp. 1–6.

[200] K. Shamsi, D. Z. Pan, and Y. Jin, "On the Impossibility of Approximation-Resilient Circuit Locking," in *International Symposium on Hardware Oriented Security and Trust (HOST)*, 2019, pp. 161–170.

[201] U. Feige, A. Fiat, and A. Shamir, "Zero-Knowledge Proofs of Identity," *Journal of Cryptology*, vol. 1, no. 2, pp. 77–94, 1988.

Hadi Mardani Kamali is a postdoctoral research associate at Florida Institute for Cybersecurity Research (FICS), the Department of Electrical and Computer Engineering at the University of Florida. He received his Ph.D. degree from the Department of Electrical and Computer Engineering at George Mason University, 2021. He received his B.S. and M.S. from the Department of Electrical and Computer Engineering at Sharif University of Technology, 2013, and K. N. T. University, 2011, respectively. His research delves into hardware security with a particular focus on exploiting IP protection techniques, design-for-trust for VLSI circuits, and CAD frameworks for security (design-for-security), in which he has numerous publications in top journals and conferences including IEEE TC/TVLSI/TCAD, IACR Transactions on CHES, DAC, ICCAD, etc., with awards including nominations for Best Paper Award in ISVLSI'20, ICCAD'19, ICCAD'20, and IEEE CAS 2020.

Kimia Zamiri Azar is a postdoctoral research associate in the Department of Electrical and Computer Engineering at the University of Florida. She received a Ph.D. degree from the Department of Electrical and Computer Engineering at George Mason University in 2021. She also received her B.S. and M.S. from the Department of Electrical and Computer Engineering at Shahid Beheshti University, 2015, and K. N. T. University, 2013, respectively. Her research interests span hardware security and trust, supply chain security, System-on-Chips security validation and verification, and IoT security. She has published over 20 journal articles and refereed conference papers in the area of VLSI design and test, with awards including nominations for Best Paper Award in IEEE Computer Society Annual Symposium on VLSI (ISVLSI)'20 and IEEE/ACM Conference on Computer-Aided-Design (ICCAD)'20.

Farimah Farahmandi is an assistant professor in the Department of Electrical and Computer Engineering at the University of Florida. She received her Ph.D. from the Department of Computer and Information Science and Engineering at the University of Florida, 2018. She received her B.S. and M.S. from the Department of Electrical and Computer Engineering at the University of Tehran, Iran in 2010 and 2013, respectively. Her research interests include design automation of System-on-Chips and energy-efficient systems, formal verification, hardware security validation, and post-silicon validation and debug. Her research has resulted in two books, seven book chapters, and several publications in premier ACM/IEEE journals and conferences including IEEE Transactions on Computers, IEEE Transactions on CAD, Design Automation Conference (DAC), and Design Automation and Test in Europe (DATE). Her research has been recognized by several awards including IEEE System Validation and Debug Technology Committee Student Research Award, Gartner Group Info-Tech Scholarship, a nomination for the Best Paper Award in ASPDAC 2017, and DAC Richard Newton Young Student Fellowship. She has actively collaborated with various research groups (IBM, Intel, and Cisco) that has led to several joint publications. She currently serves as an Associate Editor of IET Computers & Digital Techniques. She also has served on many technical program committees as well as organizing committees of premier ACM and IEEE conferences. Her research has been sponsored by SRC, AFRL, DARPA, and Cisco. She is a member of IEEE and ACM.

Mark Tehranipoor is currently the Intel Charles E. Young Preeminence Endowed Chair Professor in Cybersecurity at the University of Florida. His current research projects include: hardware security and trust, supply chain security, IoT security, VLSI design, test and reliability. He is a recipient of a dozen best paper awards and nominations, as well as the 2008 IEEE Computer Society (CS) Meritorious Service Award, the 2012 IEEE CS Outstanding Contribution, the 2009 NSF CAREER Award, and the 2014 AFOSR MURI award. He received the 2020 University of Florida Innovation of the year as well as teacher/scholar of the year awards. He co-founded the IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), IEEE International Conference on Physical Assurance and Inspection of Electronics (PAINE). He serves on the program committee of more than a dozen leading conferences and workshops. He has also served as Program and General Chair of a number of IEEE and ACM sponsored conferences and workshops (HOST, ITC, DFT, D3T, DBT, NATW, and more). He is currently serving as a founding EIC for Journal on Hardware and Systems Security (HaSS) and served as Associate Editor for TC, JETTA, JOLPE, TODAES, IEEE D&T, TVLSI. He is currently serving as a founding director for Florida Institute for Cybersecurity Research (FICS) and a number of other centers with focus on microelectronics security. Dr. Tehranipoor is a Fellow of the IEEE, a Golden Core Member of IEEE CS, and Member of ACM and ACM SIGDA.