Complexity Analysis of the SAT Attack on Logic Locking

Yadi Zhong, *Student Member IEEE* and Ujjwal Guin, *Member IEEE*

***Abstract*—Due to the adoption of horizontal business models following the globalization of semiconductor manufacturing, the overproduction of integrated circuits (ICs) and the piracy of intellectual properties (IPs) can lead to significant damage to the integrity of the semiconductor supply chain. Logic locking emerges as a primary design-for-security measure to counter these threats, where ICs become fully functional only when unlocked with a secret key. However, Boolean satisfiability-based attacks have rendered most locking schemes ineffective. This gives rise to numerous defenses and new locking methods to achieve SAT resiliency. This paper provides a unique perspective on the SAT attack efficiency based on conjunctive normal form (CNF) stored in SAT solver. First, we show how the attack learns new relations between keys in every iteration using distinguishing input patterns and the corresponding oracle responses. The input-output pairs result in new CNF clauses of unknown keys to be appended to the SAT solver, which leads to an exponential reduction in incorrect key values. Second, we demonstrate that the SAT attack can break any locking scheme within linear iteration complexity of key size. Moreover, we show how key constraints on point functions affect the SAT attack complexity. We explain why proper key constraint on AntiSAT reduces the complexity effectively to constant 1. The same constraint helps the breaking of CAS-Lock down to linear iteration complexity. Our analysis provides a new perspective on the capabilities of SAT attack against multiplier benchmark c6288, and we provide new directions to achieve SAT resiliency.**

***Index Terms*—Logic locking, Boolean satisfiability (SAT), conjunctive normal form (CNF), reverse engineering, IP piracy, IC overproduction.**

I. Introduction

The integrated circuits (ICs) are fundamental to virtually every technology in the Department of Defense (DoD), industrial and commercial spaces. Moore's Law has guided the microelectronics industry for decades to enhance the performance of ICs. The continuous addition of new functionalities in SoCs has forced design houses to adopt newer and lower technology nodes to increase operational speed, reduce power consumption, overall die area, and the resultant cost of a chip. This exponential growth becomes feasible due to the globalization of semiconductor design, manufacturing, and test processes. Building and maintaining a fabrication unit (foundry) requires a multi-billion dollar investment [1]. As a result, a system-on-a-chip (SoC) design house acquires intellectual properties (IPs) from many vendors and sends the design to a foundry for manufacturing, typically located offshore due to the horizontal integration in the semiconductor industry. At present, the majority of the SoC design houses no longer design the complete SoC and manufacture chips on their own. As a result, the trusted foundry model is no longer assumed to be valid for producing ICs, where the trustworthiness of microelectronic parts is often questioned.

Yadi Zhong and Ujjwal Guin are with the Department of Electrical and Computer Engineering, Auburn University, AL, USA (e-mail: {yadi and ujjwal.guin}@auburn.edu).

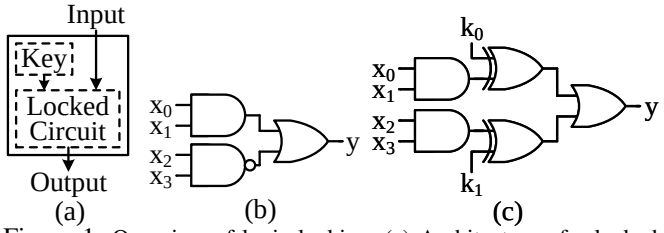

Figure 1: Overview of logic locking. (a) Architecture of a locked circuit. (b) Original design. (c) XOR-based locking, with secret key of $k_0k_1 = 01$.

Due to the outsourced IC design and fabrication, the underlying hardware in various information systems that were once trusted can no longer be so. The untrusted chip fabrication and test facilities represent security threats to the current horizontal integration. The security threats posed by these entities include: (i) overproduction of ICs, where an untrusted foundry fabricates more chips without the consent of the SoC design house to generate revenue by selling them in the market [2]–[8], and (ii) piracy of IPs, where an entity in the supply chain can use, modify and/or sell functional IPs illegally [9]–[12]. An untrusted foundry has access to all the mask information constructed from the GDSII or OASIS files and then reconstructs all the layers and the complete netlist with advanced tools [13]. In addition, reverse engineering (RE) of ICs becomes feasible even for advanced technology nodes due to the advancement of the tools for the decapsulation of the ICs and imaging. RE is commonly used in the semiconductor industry to perform failure analysis, defect identification, and verify intellectual property (IP) infringement [14], [15]. Unfortunately, the same RE can be exploited by an adversary to reconstruct the gate-level netlist from a chip [16].

One of the best ways to prevent an adversary from cloning a netlist (either by an untrusted foundry or a reverse engineer) is to hide or obfuscate the circuit. The attacker cannot decode the original functionality even after extracting the netlist from RE. Logic locking promises to hide the inner details of a circuit by inserting a set of key gates. The only way to recover the original functionality is by applying a secret key stored in a tamper-proof memory of the chip. Figure 1 shows an abstract representation of logic locking. In addition to logic locking, hardware watermarking [17]–[19] could identify and prevent copying a netlist to a certain extent; however, it does not offer

Table I: Summary of post-SAT logic locking techniques and corresponding attacks.

Locking Type	Techniques	Attacks
Point function	[23]–[32]	[71]–[85]
Cyclic	[33]–[38]	[86]–[88]
LUT	[39]–[46]	[46], [89], [90]
Scan	[47]–[52]	[91]–[93]
FSM	[53]–[58]	[94]–[99]
Timing	[59]–[64]	[100], [101]
HLS	[65]–[70]	[83], [84], [101]

a proactive protection mechanism. The initial efforts in logic locking [2], [7], [20], [21], and hardware watermarking [17]–[19] were broken by Boolean Satisfiability (SAT) attack [22]. The distinguishing input patterns, obtained from SAT solver, combined with their corresponding responses from the oracle, are crucial for SAT attack [22] to uniquely determine the secret key. A DIP with its oracle response is denoted as an input-output (IO) pair, and we will use this terminology throughout the paper. The effectiveness of SAT attack propels the research community for new locking schemes in the post-SAT era, which are summarized in Table I. These include point function-based lockings [23]–[32], cyclic-based [33]–[38], LUT/routing-based [39]–[46], scan and finite-state machine (FSM)-based lockings [47]–[58], timing-based [59]–[64], and high-level synthesis (HLS)-based [65]–[70]. Concurrently, multiple attacks [46], [71]–[101] against these logic locking techniques arise.

This paper presents two novel aspects to analyze the iteration complexity of the oracle-guided SAT attack. First, we show a detailed analysis of the SAT attack based on the conjunctive normal form (CNF) clauses stored in the SAT solver. The SAT attack iteratively finds DIPs to eliminate an equivalent class of incorrect keys. We explore what the attack learned after finding a DIP at each iteration. We show that the SAT tool creates a relationship between different key bits by applying the DIP to the oracle and observing the correct response. Note that the expected goal for any logic locking technique is to achieve an exponential iteration complexity of key size so that an adversary cannot determine the correct key value within given time constraints. However, our analysis points to the linear growth of the required patterns or iterations rather than the desired exponential increase with keys. Using examples, we show how the attack uses a DIP to eliminate a class of equivalent keys to make the complexity linear. We also show that the complexity gets even lower for circuits with multiple overlapping logic cones. Note that a logic cone can be described as a directed graph where the inputs nodes and gates point toward the sole output. Second, one interesting observation is that the complexity (*i.e.,* number of iterations/DIPs) of the SAT attack often reduces with increasing key size. We provide detailed explanations of why it takes fewer iterations to find the correct key when we lock a circuit with a larger key size. Finally, we analyze the SAT attack complexity for a circuit locked using point functions [27], [29], [102].

The contributions of this paper are summarized as follows:

- *New perspective on SAT attack efficiency:* Even though the SAT attack was presented in 2015, its complexity analysis was not performed to find out why a DIP eliminates a large number of keys. This paper uses examples to describe the step-by-step analysis of incorrect key elimination for each DIP. The inter-dependencies among key bits are clearly revealed with a DIP and the corresponding oracle's response. We further show that the attack requires less number of iterations when keys can be observed simultaneously at multiple primary outputs.
- *SAT attack complexity:* The majority of locking schemes focus on an exponential complexity close to the entire keyspace for ensuring hardness against SAT attack. However, SAT attack has shown an overall linear trend upon key size. Furthermore, increasing the number of key gates does not necessarily correlate to more iterations solving the correct key. Instead, it is common to observe a decrease in iteration complexity. *To the best of our knowledge, we are the first to show a reduction in attack complexity with a larger key size.* We believe that the findings of this paper provide researchers with the necessary information to develop an SAT resilient solution. To address the reduction in attack complexity with a larger key, we observe that the oracle's response of a DIP plays an important role in removing a large number of incorrect keys. For example, a response 0 at the OR gate effectively splits the logic cone into two subcones, where keys inside the subcone are dependent, but independent from the other subcone's. Such IO pairs exponentially reduce the attack complexity. Similarly, logic 1 at the AND gate has a similar effect in shrinking the keyspace.
- *SAT analysis on point functions-based locking:* Logic locking with point functions has demonstrated a strictly exponential iteration complexity against SAT attack. Unfortunately, those locking designs with complementary key blocks can be broken under the properly constraining of sub-keys with the SAT tool. We show how and why SAT attack needs one IO pair only for deriving the complete key for AntiSAT under certain key restrictions, but it would remain exponential complexity if the constraint is placed on the other key block instead. We provide a similar analysis on CAS-Lock, which can effectively reduce the exponential iteration complexity to linear. We present insights on how the same analysis can be applied to TTLock and various versions of SFLL.
- *SAT attack time complexity:* The SAT attack, or its variants, can be very effective in breaking secure logic locking that aims to achieve exponential iteration complexity. To build an SAT-resilient solution, we investigate the time complexity rather than the iteration count. We show that the attack spends most time for the c6288 multiplier benchmark on the last iteration of UNSAT so to confirm that no other DIPs exist. Note that the iteration count is still linear to key size (see Table IV).

The rest of the paper is organized as follows. We introduce the background of SAT attack and various locking methods in Section II. The inter-dependency between keys learned by SAT solver after each DIP is extensively explored in Section III. The SAT attack complexity is further analyzed and explained

in Section IV. Analysis of the point functions is shown in Section V. The future directions are described in Section VI. Finally, we conclude the paper in Section VII.

II. Background

A. SAT attack on Logic Locking

The entire series of attacks and the solutions thereafter originated from the SAT attack [22]. Subramanyan et al. [22] exploit the idea of combinational equivalence checking with miter circuit and Boolean Satisfiability [103] to attack logic locking schemes. This oracle-guided attack successfully derives the secret key of various logic locking techniques [2], [7], [17]–[21] within a short time frame. The SAT attack requires two circuits, the original circuit, $C_O(X, Y)$, and its locked version, $C(X, K, Y)$, where X, Y, and K are the inputs, outputs, and key, respectively. The correct key K_c restores the original circuit functionality so that its output response is always consistent with the original circuit (e.g., the oracle) under every possible input combination, $C(X, K_c, Y) = C_O(X, Y)$. An incorrect key programmed in the tamper-proof memory leads to output mismatch under one or more input vectors. The output discrepancy between an incorrect key and the correct one is shown on the miter circuit's output. The SAT attack derives the key through the following steps:

Algorithm 1: SAT attack on logic locking [22].

Input : Unlocked circuit, oracle ($C_O(X, Y)$) and locked circuit ($C(X, K, Y)$)
Output: Correct Key (K_c)

```
1  i ← 0 ;
2  F ← [];
3  while (true) do
4  |   i ← i + 1 ;
5  |   [X_i, K_i, r] = sat[F ∧ (Y_{A_i} ≠ Y_{B_i})];
6  |   if (r == false) then
7  |   |   break;
8  |   end
9  |   Y_i = sim_eval(X_i);
10 |   F ← F ∧ C(X_i, K, Y_i);
11 end
12 K_c ← K_i;
13 return K_c ;
```

• *Finding the distinguishing input pattern (DIP) from the miter circuit:* It first constructs a miter with two copies of the locked circuit A and B. Both circuits ($C(X, K_A, Y_A)$ and $C(X, K_B, Y_B)$ in CNF) share the same input X except for the keys, K_A, K_B. Any output mismatch between the two locked circuits can be easily identified at the miter's output. In each round (*i.e.,* i^{th}), the tool finds the hypothesis key K_i, and reports a Boolean indicator r depending on whether a satisfiable assignment for the miter exists or not, Algorithm 1, Line 5. If SAT is returned, the miter succeeded in amplifying the mismatched output, r is `true`, and the corresponding input pattern X_i is also recorded.

• *Deriving the correct key:* Upon obtaining a DIP X_i, SAT attack acquires the actual output Y_i from oracle simulation, $C_O(X_i, Y_i)$, Line 9. Input X_i and output response Y_i are used in updating the CNF formula F, Line 10. The clauses in F help narrow down the valid keyspace until it is left with only the correct key(s). If the UNSAT conclusion is generated, the differential output cannot be observed, r is assigned to `false`, and X_i is empty (Lines 6-8) and the program ends. Note that the last iteration of SAT attack returns UNSAT as all incorrect keys are pruned from the keyspace.

The SAT attack repeats the above two steps, where it iteratively checks for satisfiable assignment of the miter circuit. If r is `true` at the i^{th} iteration, we know that incorrect keys still exist in the search space. When the miter circuit becomes UNSAT with the clauses in F, the Boolean variable r becomes `false`, indicating no differential output exists. This means no more incorrect keys can be found as no discrepancy can be produced. If multiple keys remain in the search space, it must be true that multiple solutions are valid since they all give the same output response. This holds for a few locking designs [29], [102] and certain locking scenarios, *e.g.,* chained XOR key gates, where the correct key is not unique. Returning any one of them can restore the original circuit functionality. If only one key is left, it must be the right one. Then, the attack exits the `while` loop, Lines 6-8, and extracts the last round's hypothesis key as the correct one, Line 12. The attack finishes by reporting the correct key to the console, Line 13.

Note that we modify the original program [22] by disabling both preload vectors (all zeros and all ones) so that the number of IO pairs $|P|$ used to derive the correct key is one less than the number of total iterations TI, $|P| = TI - 1$. This is because the last iteration does not produce a DIP.

B. SAT Resistant Logic Locking Techniques and Attacks

As SAT attack [22] successfully breaks various logic locking techniques [2], [7], [17]–[21], it propels the research community to explore new locking schemes [23]–[32] that utilize point functions for achieving the minimal output corruptibility. SARLock [23] only perturbs one input pattern's output for each incorrect key. AntiSAT [31], [32], [102] and CAS-Lock [29] configure the point function with two complementary blocks g and $\overline{g}$. SFLL [27], [28], [30] flips the output for certain input patterns, where the correct key flips back the upset output and restores the original functionality. Although these techniques guarantee exponential iterations in SAT attack, various attacks [71]–[85] have been proposed to exploit the designs' vulnerabilities, *e.g.,* from structural and functional perspectives, and restore the original circuit. Nevertheless, SAT attack is still the backbone for the oracle-guided attacks [72]–[76], [80], [81].

III. SAT Attack Analysis: Pruning of Incorrect Key with CNF Update

This section presents a novel perspective of analyzing the SAT attack's effectiveness in breaking various locking schemes in deriving the secret key. We investigate the CNF clauses stored in the SAT solver and how it gets updated in every

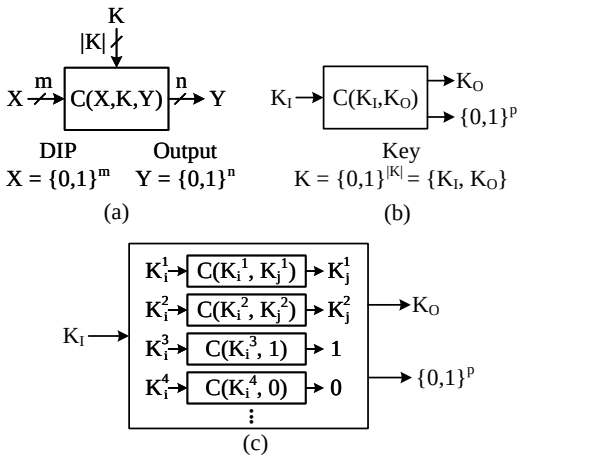

Figure 2: Abstract representation of functions of key bits derived from an IO pair. (a) Locked circuit with an IO pair, (b) function of keys, and (c) subfunctions.

iteration with a DIP and its output response. The CNF consists of multiple clauses connected with AND ($\wedge$). One or more literals are joined by OR ($\vee$) inside each clause. We use literals, variables, and nodes interchangeably.

The SAT attack requires an unlocked circuit, $C_O(X, Y)$, and its locked version, $C(X, K, Y)$, where X, Y, and K are m, n and $|K|$ bit wide. The correct key K_c restores the original function so that its output response is always consistent with the unlocked circuit for all input combinations, *i.e.,* $C(X, K_c, Y) = C_O(X, Y)$. The SAT solver iteratively finds satisfiable assignments of the miter circuit whose inputs are denoted as DIPs. DIPs and the corresponding oracle outputs are denoted as IO pairs. As logic values for an IO pair $\{X, Y\}$ are known, $C(X, K, Y)$, shown in Figure 2(a), is transformed into the functions of keys $C(K_I, K_O)$, shown in Figure 2(b), where K_O can be derived from K_I. Further, $C(K_I, K_O)$ can be expanded further and is shown in Figure 2(c). Any key in K_O, *e.g.,* K_j^t, is dependent upon key bits K_i^t, *i.e.,* $K_j^t = f(K_i^t)$, where $K_i^t \subseteq K_I$. In addition, the combination of some key bits, *e.g.,* K_i^s, $K_i^s \subseteq K_I$, produces a deterministic output, *i.e.,* either logic 0 or 1.

$$C(K_I, K_O) \iff$$
$$\begin{cases} C(K_i^t, K_j^t), \text{where } K_j^t = f(K_i^t), K_j^t \notin K_i^t; i, j, t = 1, 2, \ldots \\ C(K_i^s, \{0, 1\}), \text{where } \{0, 1\} = f(K_i^s); i, s = 1, 2, \ldots \end{cases}$$

This key-dependent function $C(K_I, K_O)$ reveals additional information on the interdependency between key bits, *e.g.,* $K_j^t = f(K_i^t)$ and $\{0, 1\} = f(K_i^s)$, crucial to the implicit removal of large incorrect key combinations.

The placement of the key gates inside a particular cone is crucial as overlapping cones may reduce the attack complexity. A logic cone can be described as a combinational logic unit that represents a Boolean function bounded by an output and all its inputs. An increased number of primary outputs usually leads to multiple key values propagating across different output bits simultaneously. We begin our analysis with an example circuit with a non-overlapping cone with a single output and show how the SAT attack decrypts the 3-bit key with 3 IO pairs. Then, we describe how SAT attack can use fewer patterns to determine the secret key under overlapping logic cones.

A. SAT Attack for a Locked Cone with One Output

This section examines how SAT attack implicitly removes the incorrect keys from the entire key search space. As described in Section II-A, SAT solver finds a valid assignment to the miter circuit, and the tool records the extracted input vector, along with its output response obtained from the oracle simulation. The following example shows how SAT attack learns additional information on the secret keys from each IO pair from the miter circuit and oracle simulation.

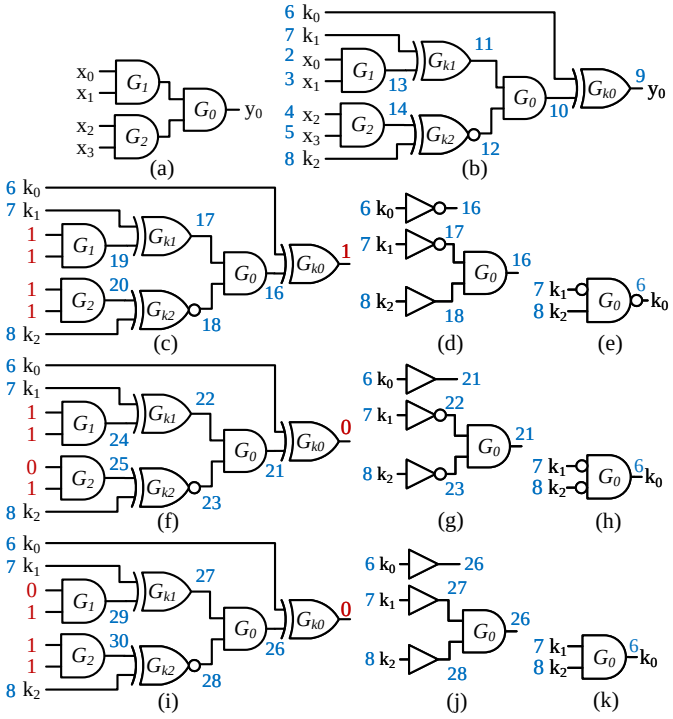

Figure 3: Step-by-step SAT attack analysis. (a) Original circuit. (b) Locked circuit with $K = \{001\}$. CNF update and key-pruning for (c-e) 1st IO pair $P_1 = \{1111; 1\}$; (f-h) 2nd IO pair $P_2 = \{1101; 0\}$, and (i-k) 3rd IO pair $P_3 = \{0111; 0\}$.

Let us consider an example circuit with 4 inputs $x_0, ..., x_3$ and 1 output y_0 of Figure 3(a). Figure 3(b) is the locked circuit with a 3-bit key, k_0, k_1, k_2, using strong logic locking (SLL) scheme. Each node is assigned with a unique literal (in blue) by SAT solver. Upon finding a valid assignment to the miter circuit in the first iteration, DIP X_1 is extracted, $\{X_1\} = \{x_0, x_1, ..., x_3\} = \{1111\}$. The output response $Y_1 = 1$ is obtained from oracle simulation with input X_1. SAT attack then records this IO pair $P_1 = \{X_1; Y_1\} = \{x_0, ..., x_3; y_0\} = \{1111; 1\}$. We show in detail how the locked circuit's CNF gets updated under P_1, where the search space is shrank in half (eliminated 4 incorrect keys). The literal assignment for the locked circuit's original CNF remains unchanged, but the internal nodes for IO pair P_1 are labeled with new variables {16-20} (Figure 3(c), consistent with the internal operations of SAT attack [22]. The CNF for $C(X_1, K, Y_1)$ (abbreviated as C_1) is:

$$C_1 = \overbrace{(\overline{17} \vee \overline{18} \vee 16) \wedge (17 \vee \overline{16}) \wedge (18 \vee \overline{16})}^{\text{AND gate } G_0} \wedge$$
$$\overbrace{(\overline{2} \vee \overline{3} \vee 19) \wedge (2 \vee \overline{19}) \wedge (3 \vee \overline{19})}^{\text{AND gate } G_1} \wedge$$
$$\overbrace{(\overline{4} \vee \overline{5} \vee 20) \wedge (4 \vee \overline{20}) \wedge (5 \vee \overline{20})}^{\text{AND gate } G_2} \wedge$$
$$\overbrace{(\overline{6} \vee \overline{16} \vee \overline{9}) \wedge (\overline{6} \vee 16 \vee 9) \wedge (6 \vee \overline{16} \vee 9) \wedge (6 \vee 16 \vee \overline{9})}^{\text{XOR gate } G_{k0}} \wedge$$
$$\overbrace{(\overline{7} \vee \overline{19} \vee \overline{17}) \wedge (\overline{7} \vee 19 \vee 17) \wedge (7 \vee \overline{19} \vee 17) \wedge (7 \vee 19 \vee \overline{17})}^{\text{XOR gate } G_{k1}} \wedge$$
$$\overbrace{(\overline{20} \vee \overline{8} \vee 18) \wedge (\overline{20} \vee 8 \vee \overline{18}) \wedge (20 \vee \overline{8} \vee \overline{18}) \wedge (20 \vee 8 \vee 18)}^{\text{XNOR gate } G_{k2}}$$

With IO pair P_1, we know the logic values for input/output, namely literals $2 = 1, 3 = 1, 4 = 1, 5 = 1, 9 = 1$, and the C_1 is updated as:

$$\begin{aligned} C_1 \;=\; & (\overline{17} \vee \overline{18} \vee 16) \wedge (17 \vee \overline{16}) \wedge (18 \vee \overline{16}) \wedge (19) \wedge (20) \\ & \wedge (\overline{6} \vee \overline{16}) \wedge (6 \vee 16) \wedge (\overline{7} \vee \overline{19} \vee \overline{17}) \wedge (\overline{7} \vee 19 \vee 17) \\ & \wedge (7 \vee \overline{19} \vee 17) \wedge (7 \vee 19 \vee \overline{17}) \wedge (\overline{20} \vee \overline{8} \vee 18) \\ & \wedge (\overline{20} \vee 8 \vee \overline{18}) \wedge (20 \vee \overline{8} \vee \overline{18}) \wedge (20 \vee 8 \vee 18) \end{aligned}$$

Both nodes 19, 20 are in logic 1, and the CNF is adjusted:

$$\begin{aligned} C_1 \;=\; & (\overline{17} \vee \overline{18} \vee 16) \wedge (17 \vee \overline{16}) \wedge (18 \vee \overline{16}) \wedge (\overline{6} \vee \overline{16}) \wedge \\ & (6 \vee 16) \wedge (\overline{7} \vee \overline{17}) \wedge (7 \vee 17) \wedge (\overline{8} \vee 18) \wedge (8 \vee \overline{18}) \end{aligned}$$

This equation reveals that literals 6 and 16 have the opposite logic values, so are 7 and 17, while 8 and 18 are identical. The circuit representation of C_1 is shown in Figure 3(d), still a function of k_0, k_1, k_2. These are the clauses appended in the formula F (Algorithm 1 Line 10). Equivalently, what SAT attack learned from the 1st IO pair P_1 is essentially a relation between 3 key bits, where $\overline{k_0} = \texttt{AND}(\overline{k_1}, k_2)$, as in Figure 3(e). The constraint shrinks the possible keyspace in half.

On the second iteration, SAT attack returns the 2nd IO pair $P_2 = \{X_2; Y_2\} = \{1101; 0\}$, as in Figure 3(d). Using the derivation we performed for the first iteration, the circuit representation of the added CNF clauses is shown in Figure 3(g), which again is a function between the 3 key bits, $k_0 = \texttt{AND}(\overline{k_1}, \overline{k_2})$ (Figure 3(h)). It further shrinks the remaining keyspace in half, with only two keys left valid. Figure 3(i) shows the 3^{rd} IO pair $P_3 = \{0111; 0\}$, whose CNF $C(X_3, K, Y_3)$ and its equivalent relation $k_0 = \texttt{AND}(k_1, k_2)$ are illustrated in Figure 3(j), (k), respectively. The combined effect of these three IO pairs, P_1, P_2, P_3, Figure 3(e, h, k), uniquely determine key $K = \{k_0, k_1, k_2\} = \{001\}$. On the 4th iteration, no more distinguishing input can be found for the miter circuit, where r is `false`, and the SAT attack is complete.

In short, each IO pair provides additional information on the unknown key bits, where $C(X_i, K, Y_i)$ essentially becomes an equation for the unknown keys. A new equation for key is obtained in every iteration from the corresponding IO pair, which is independent of the findings derived from the previous rounds. SAT attack derives the secret key once the accumulated system of equations can uniquely determine all key bits.

B. SAT Attack against Multiple Overlapping Logic Cones

It is common for a circuit to have multiple outputs or fanouts. In other words, that circuit has multiple logic cones. With more fanouts, incorrect key responses are more likely to be observed than a single output. As the logic values for multiple keys can reach several outputs simultaneously, it accelerates and facilitates the removal of incorrect combinations to get the final key than the single logic cone where every key has to be observed from the same output pin. This is demonstrated by the example below. The following example shows that SAT attack needs fewer iterations to derive the secret key under multiple intersecting logic cones.

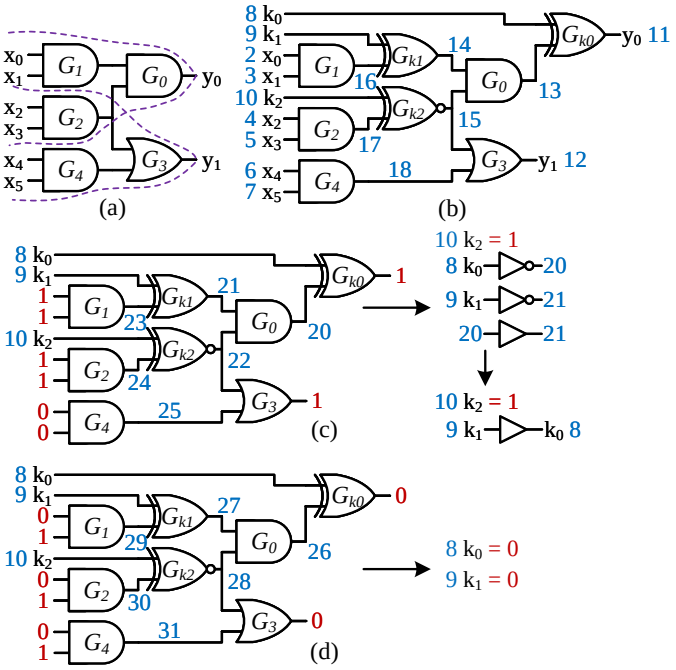

Figure 4: SAT attack on 2 intersecting cones with $K = \{001\}$. (a) Original circuit. (b) Locked circuit. CNF update and key-pruning for (c) 1st pair $P_1 = \{111100; 11\}$, (d) 2nd IO pair $P_2 = \{010101; 00\}$.

Let us consider a circuit with 2 outputs, y_0 and y_1, as shown in Figure 4(a). The locked circuit, as shown in Figure 4(b), has 3 key bits, k_0, k_1, k_2, with the same locations as in Figure 3(b). It differs from the locked circuit in Figure 3(b) with additional gates G_3, G_4 and output y_1. This circuit has two logic cones; one with output y_0, inputs x_0, x_1, x_2, x_3, keys k_0, k_1, k_2, and gates $G_0, G_1, G_2, G_{k0}, G_{k1}, G_{k2}$; the other with output y_1, inputs x_2, x_3, x_4, x_5, key k_2, and gates G_2, G_3, G_4, G_{k2}. The effect of k_2 can be observed from both outputs, y_0 and y_1. SAT attack only needs 2 IO pairs to solve the keys, as opposed to 3 IO observations for the locked cone with a single output y_0 in Figure 3(b). Figure 4(c) illustrates the 1st IO pair $P_1 = \{X_1; Y_1\} = \{x_0, ..., x_5; y_0, y_1\} = \{011001; 00\}$, Its equivalent CNF expression of $C(X_1, K, Y_1)$ (abbreviated as C_1) is expressed in:

$$\begin{aligned} C_1 = & (\overline{21} \vee \overline{22} \vee 20) \wedge (21 \vee \overline{20}) \wedge (22 \vee \overline{20}) \wedge (\overline{2} \vee \overline{3} \vee 23) \\ & \wedge (2 \vee \overline{23}) \wedge (3 \vee \overline{23}) \wedge (\overline{4} \vee \overline{5} \vee 24) \wedge (4 \vee \overline{24}) \wedge (5 \vee \overline{24}) \\ & \wedge (22 \vee 25 \vee \overline{12}) \wedge (\overline{22} \vee 12) \wedge (\overline{25} \vee 12) \wedge (\overline{6} \vee \overline{7} \vee 25) \end{aligned}$$

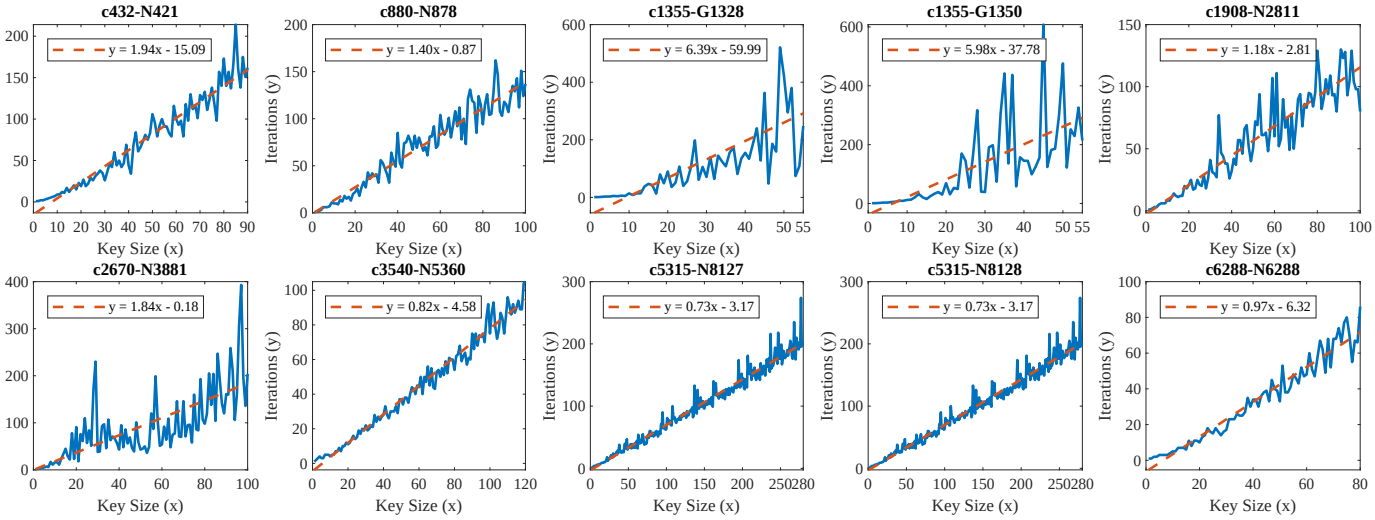

Figure 5: SAT attack total iterations for ISCAS'85 benchmarks.

$$\wedge\ (6 \vee \overline{25}) \wedge (7 \vee \overline{25}) \wedge (\overline{8} \vee \overline{20} \vee \overline{11}) \wedge (\overline{8} \vee 20 \vee 11)$$
$$\wedge\ (8 \vee \overline{20} \vee 11) \wedge (8 \vee 20 \vee \overline{11}) \wedge (\overline{9} \vee \overline{23} \vee \overline{21})$$
$$\wedge\ (\overline{9} \vee 23 \vee 21) \wedge (9 \vee \overline{23} \vee 21) \wedge (9 \vee 23 \vee \overline{21})$$
$$\wedge\ (\overline{10} \vee \overline{24} \vee 22) \wedge (\overline{10} \vee 24 \vee \overline{22}) \wedge (10 \vee \overline{24} \vee \overline{22})$$
$$\wedge\ (10 \vee 24 \vee 22)$$

The 1st IO pair P_1 gives $2 = 1, 3 = 1, 4 = 1, 5 = 1, 6 = 0, 7 = 0, 15 = 1, 17 = 1$. The CNF for the locked circuit with P_1 is adjusted analogously to the previous example (Figure 3) by plugging in the logic value of these known literals. It is straightforward that node 23, the output of AND gate G_1, has logic 1, as its inputs are literals $2 = 1$ (x_0) and $3 = 1$ (x_1). Similarly, nodes $24 = 1$, and $25 = 0$, based on literals $4 - 7$ ($x_2, ..., x_5$). With an output 1 for OR gate G_5, its remaining input of node 22 must be 1, as $25 = 0$. Therefore, the CNF clauses added to SAT solver after the first iteration is:

$$C_1{=}(\overline{21}{\vee}20){\wedge}(21{\vee}\overline{20}){\wedge}(\overline{8}{\vee}\overline{20}){\wedge}(8{\vee}20){\wedge}(\overline{9}{\vee}\overline{21}){\wedge}(9{\vee}21){\wedge}(10)$$

With C_1, SAT attack determines key bit $k_2 = 1$, along with key-dependent equation $k_0 = k_1$, as shown in Figure 4(c). With the 2nd IO pair $P_2 = \{X_2; Y_2\} = \{010101; 00\}$, as in Figure 4(d), SAT atttack uniquely determines both key bits k_0 and k_1 as logic 0. In the third round, SAT attack returns UNSAT (as all key bits are solved), and the program finishes.

Table II: SAT attack total iterations (TI) comparison between multiple primary outputs ($|PO|$) and single cone.

Benchmark	Locked Circuit (SLL) [22]			Locked Cone (Sec. IV)		
	$\|PO\|$	$\|K\|$	TI	$\|PO\|$	$\|K\|$	TI
c432	7	80	24	1	21	27
c880	32	192	76	1	50	80
c1355	32	137	29	1	17	33
c1908	25	220	110	1	92	123
c3540	22	167	40	1	58	40
c5315	123	231	55	1	78	60

In addition to the example described above, we perform experiments to show a weaker attack resiliency for overlapping cones, as summarized in Table II with locked ISCAS'85 benchmarks. Table II compares the attack complexity with key sizes between the complete benchmark circuit, where multiple overlapping cones exist, and the extracted single cone from the same benchmark. Columns 2-4 and 5-7 list the number of primary outputs ($|PO|$), key size ($|K|$), and the total SAT attack iterations (TI) for breaking the SLL-based locked benchmarks and the corresponding largest cone, respectively. For example, SAT attack takes 76 iterations to determine a 192-bit key for the c880 benchmark, whereas it takes 80 iterations to break the largest cone of c880 locked with a merely 50-bit key. We observe the same behavior for all other benchmarks as well. The SAT attack can only break fewer keys (or smaller key size) for a single-output logic cone than for the keys of the same benchmark circuit having multiple PO. This confirms a lower complexity for overlapping cones, which is due to the effect of incorrect keys manifested through multiple outputs where the interdependency between key bits is broken. Therefore, having multiple overlapping logic cones will reduce the iteration counts for SAT attack, making it easier to derive the final key when key bits can be observed at the outputs simultaneously. Since we are examining and analyzing the effectiveness of SAT attack, we henceforth focus on the analysis with a single logic cone only, as it is more complex than multiple cones and offers an upper bound to the iteration complexity. If we show the linear iteration complexity for a non-overlapping cone, then automatically, the same linear-complexity will be preserved for overlapping cones.

IV. SAT Attack Analysis: Iteration Complexity

In this section, we focus on the total iterations required for the SAT attack as the SAT attack complexity. We observe the linear iteration complexity for all ISCAS'85 benchmarks that agrees with the previously reported results. We, however, also observe that the decrease in iteration complexity with the increased key size for a large number of cases. To explain this phenomenon, we analyze how the output response from oracle, under certain DIPs, can trim more incorrect keys than other IO pairs. The complexity drop is caused by the multiple effective IO pairs selected by the tool. This explanation can

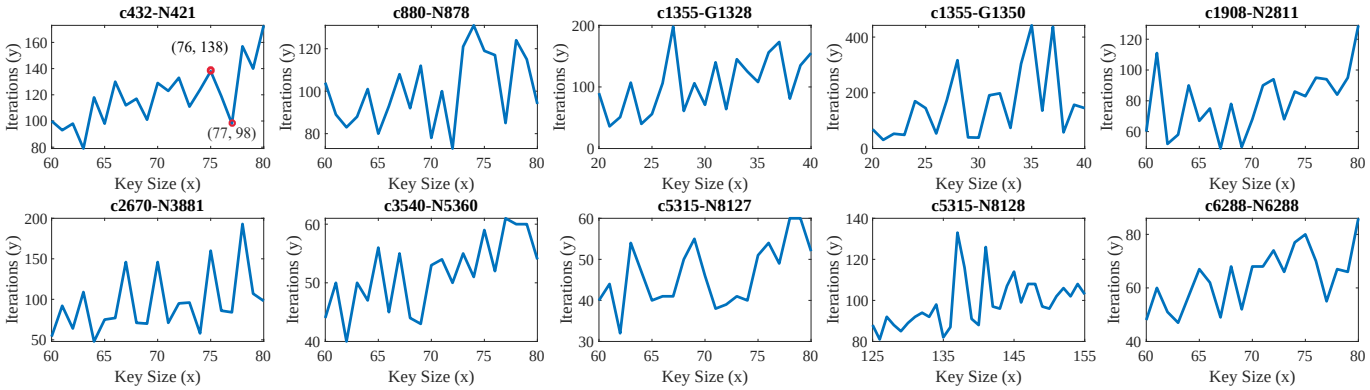

Figure 6: The zoomed-in view of SAT attack Iterations for ISCAS'85 benchmarks.

also clarify the local peaks in iteration complexity due to the SAT attack selecting multiple less-effective IO pairs. We focus on the complexity trend for the iteratively increase in key sizes for any XOR-based locked circuits. The iterative insertion of keys (and key gates) ensures that the addition of one more key bit does not alter the locations of the already inserted key bits (and key gates). *To the best of our knowledge, this is the first study to report the reduction of iteration count with an increased key size.*

The overview of SAT attack complexity analysis on the same logic cone is summarized in the following steps: (i) benchmark synthesis, (ii) cone analysis and the largest cone extraction, (iii) iterative insertion of key bits, and (iv) SAT attack iteration complexity aggregation. Synthesis is performed under 32nm technology libraries in Synopsys Design Compiler [104]. Figure 5 shows an overall linear trend in total attack iterations under increased key sizes for non-overlapping cones. The best fit lines are drawn in dashed lines with equations. To avoid the complexity reduction under multiple logic cones, the largest cone from each synthesized ISCAS'85 benchmark is extracted so that the response of any incorrect key combinations is observed through the sole output only. Each circuit is mapped to a directed graph with inputs pointing toward gates' output and, ultimately, the primary output. Logic cones are extracted by reversal of edge directions [105] and breadth-search [106] from each primary output. The ordered node list obtained in breadth-first search is used for determining (i) the largest cone (or cones if a tie) by node count and (ii) key gate insertion sequence as breadth-first search traverses all gates (nodes) within the same layer (same distance from output nodes) first before reaching gates at further layers. Following the same node order as in breadth-first search, we successively add one more XOR/XNOR key gate at a time, starting from gates closest to the primary output with increasing proximity. The original cone and its locked designs are all converted to the *bench* format. SAT attack runs through all key sizes for every locked cone, and the total iterations are recorded. Figure 5 shows the SAT attack iteration complexity on 9 benchmark cones with increasing key sizes. For example, c432-N421 is the logic cone from c432 benchmark with output N421. Cone c5315-N8127 and c5315-N8128 both contain the same gates count, but a significant overlap of gates exists. We also include cones with reconvergent fanouts [107], such as c432-N421, c1355-G1350, c5315-N8127 and c5315-N8128.

There are two observations from Figure 5. First, the overall complexity increase for SAT attack is not exponential, but linear. This means that, on average, the attack removes an exponential (or sub-exponential) number of incorrect key combinations per iteration. Second, all 9 benchmark cones exhibit the non-monotonically complexity increase when additional keys are inserted.

Figure 6 shows the zoomed-in view of SAT attack complexity for benchmark cones. For example, for cone c432-N421, it takes 138 iterations to break the key size of 76, but only needs 98 iterations when one more key bit is added. A non-monotonically increase in complexity is also observed in all the other benchmark cones. It is evident that an insertion of more key gates does not always lead to an increase in attack complexity. The non-monotonically increasing behavior in iteration complexity is observed in all cones. The question is, *what causes the SAT attack to have such complexity drops when more keys are present in a locked design?*

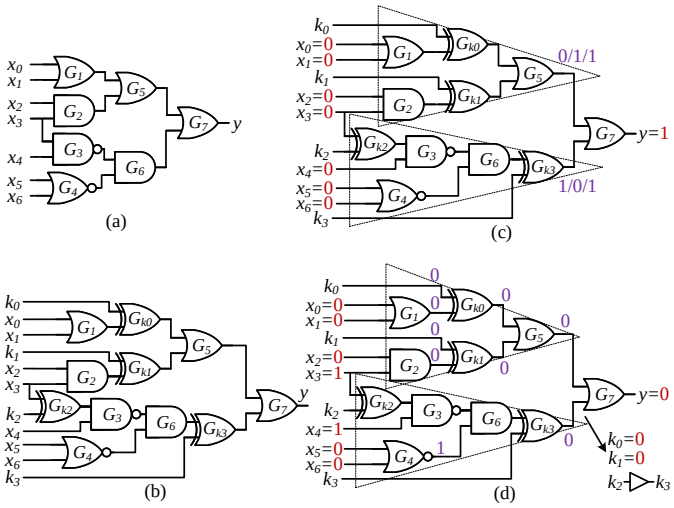

Figure 7: Key elimination. (a) original circuit, (b) locked circuit with 4 keys, (c) the 1^{st} IO pair $P_1 = \{0000000;1\}$ from SAT attack (d) the second IO pair $P_2 = \{0001100;0\}$ and equivalent relation of k_0, k_1, k_2, k_3 under P_2 only, where k_0 and k_1 are determined.

To describe the non-monotonically increasing behavior, we consider another example shown in Figure 7 where the effectiveness of individual IO pair in eliminating incorrect

Table III: SAT attack uses 2 patterns to eliminate all 15 incorrect keys from keypace. If output differs from the oracle's, ✗ is placed, else ✓. The correct key is highlighted.

4-bit key $\{k_0, ..., k_3\}$	IO Pair 1 (P_1) {0000000;1}	IO Pair 2 (P_2) {0001100;0}
0000	✓	✓
0001	✗	✗
0010	✓	✗
0011	✗	✓
0100	✓	✗
0101	✓	✗
0110	✓	✗
0111	✓	✗
1000	✓	✗
1001	✓	✗
1010	✓	✗
1011	✓	✗
1100	✓	✗
1101	✓	✗
1110	✓	✗
1111	✓	✗

keys are explored. The purpose here is to demonstrate that all the IO pairs are not equally effective in eliminating incorrect keys, some are better than others. As the SAT tool finds a DIP, which typically depends on the circuit topology, it is possible that the tool selects a more efficient DIP in earlier iterations for a locked circuit with a larger key that eliminates a large number of incorrect keys that results in a reduction in iteration. Figure 7(a) shows the circuit, where four keys ($k_0 - k_3$) are added (Figure 7(b)). Note that an OR gate (G_7) is located at the cone output. The 1^{st} IO pair $P_1 = \{X_1; Y_1\} = \{x_0, ..., x_6; y\} = \{0000000;1\}$ returned by SAT attack has $y = 1$ as the output. As logic 1 is the output of G_7, its two inputs could be any of the 3 combinations $\{01/10/11\}$. As the correct key cannot be determined uniquely, we focus on finding the incorrect ones, which are unique and result from $\{00\}$. One can simply find these incorrect ones using logic propagation, and it can be shown that there exist only 2 incorrect keys, Table III Column 2. Any key combinations that cause an output mismatch with the oracle's are marked with ✗, indicating an incorrect key value implicitly removed from keyspace; key value(s) which produces the same output as the oracle's is noted with ✓. From the 1^{st} iteration, we observe fewer incorrect keys (i.e., $2 \ll \frac{2^4}{2}$) are removed than the 2^{nd} iteration (i.e., $14 \gg \frac{2^4}{2}$) due to the properties of OR gate, where no unique conclusion can be made regarding its inputs (i.e., 10, 01 or 11) if the output is 1.

On the second iteration, the tool obtains another IO pair, $P_2 = \{0001100;0\}$, with 0 at the output of the OR gate. The rest 13 incorrect key combinations are identified from P_2, as listed in Table III, Column 3. Here, we are interested in how P_2 trims more than half of the keys in the search space. The locked circuit with the IO pair P_2 is shown in Figure 7(d). With the same derivation for CNF $C(X_2, K, Y_2)$, we know the outputs of gates G_1, G_2, G_4 are 0, 0, 1, once we have the input X_2. These gates' outputs can be similarly decided with X_1. With output $y = 0$ at OR gate G_7, both inputs from this OR gate must be 0. This means both outputs of OR gate G_5 and XOR gate G_{k3} are 0; and subsequently, the inputs of OR gate G_5 must be 0 as well, which are the output of both XOR gates G_{k0} and G_{k1}. This results in the unique solution for 2 key bits k_0 and k_1 with $k_0 = 0, k_1 = 0$.

A similar analysis can be performed on AND gates, whose inputs are uniquely defined under a logic 1 output. In summary, having a response of 0 at OR gates, or 1 at AND gates effectively splits the cone to two halves, where keys in one half are independent of the keys on the other half. This is equivalent to splitting the logic cone into two subcones based on inputs ports of OR/AND gates, where keys in both subcones can be evaluated and trimmed simultaneously. Therefore, the efficiency of removing incorrect keys depends on IO pairs, where the selection of an IO pair depends on the locked circuit topology that changes when adding more key bits. The effective IO pairs help remove more incorrect keys than the others. If a few effective IO pairs are selected in earlier iterations of the SAT attack, the iteration count can go down significantly. This leads to a non-monotonically increasing iteration complexity with the key size.

V. Case Study: Locking with Point Functions

As the efficiency of SAT attack is indisputable, the subsequent logic locking proposals shift the focus toward building an exponential complexity in total iterations against SAT attack. One of the common approaches is to embed a point function right before the output of a logic cone, where the circuit's output response is perturbed based on the designer's chosen input combinations. This section presents a theoretical analysis of point functions of AntiSAT [102], CAS-Lock [29], TTLock [26], and SFLL [27], and explains why they can also be breakable with SAT-based attacks.

A. Deterministic Property of SAT Attack with Constraints

This section analyzes the SAT attack complexity on point function-based locking schemes with complementary blocks, g and $\overline{g}$, where K_g and $K_{\overline{g}}$ are inside g and $\overline{g}$, respectively. Sengupta et al. [81] have shown an effective approach to reduce AntiSAT and CAS-Lock to polynomial complexity with key-bit mapping (KBM) & SAT, where KBM separates K_g and $K_{\overline{g}}$ and SAT attack is applied with a fixed K_g. The following analysis explains how and why SAT attack is still effective in breaking AntiSAT and CAS-Lock under proper key constraints. Our proposed SAT attack analysis also explains the same attack complexity of linear in iterations. In addition, we provide explanations to show (i) AntiSAT with fixed K_g requires only one IO pair to determine the secret key, and (ii) linear complexity for CAS-Lock under the same constraint. However, constraining $K_{\overline{g}}$ would not give any extra benefits on the complexity reduction to an adversary on breaking AntiSAT.

1) SAT attack analysis on AntiSAT under key constraints: The two sets of keys, K_g and $K_{\overline{g}}$, in AntiSAT offer two choices for the attacker, fixing one or the other. Using our key pruning analysis of Section III, we explain how an adversary can determine the key with single IO pair only when setting K_g constant. Yet, he/she will be less fortunate in breaking the secret key if $K_{\overline{g}}$ is kept constant instead.

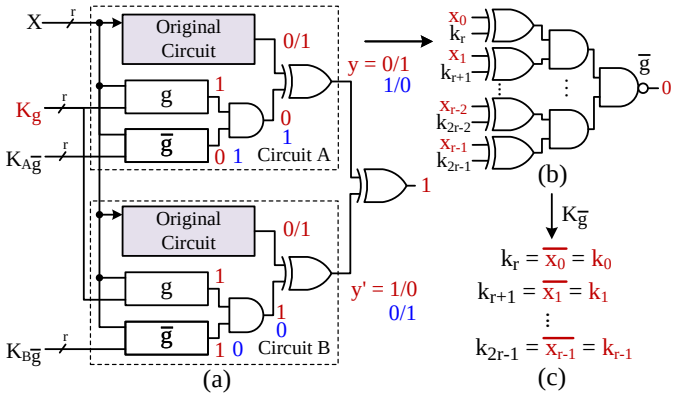

Figure 8: SAT attack on AntiSAT with fixed key K_g. (a) Miter construction. (b) CNF update. (c)$K_{\overline{g}} = \{k_r - k_{2r-1}\}$ is determined.

• Key constraint on K_g: Let us consider a circuit with r-bit input $X = \{x_0, ..., x_{r-1}\}$, 1-bit output y, locked with $2r$-bit keys of $K_g = \{k_0, ..., k_{r-1}\}$ and $K_{\overline{g}} = \{k_r, ..., k_{2r-1}\}$ of r-bit each. We assume the attacker already knows the bit locations for K_g using the KBM of [81]. Furthermore, the r-bit $K_g = \{k_0, ..., k_{r-1}\}$ is set to a constant vector. SAT attack is able to find an IO pair and uniquely determines all bits of $K_{\overline{g}}$. Figure 8(a) shows the miter construction, where K_g are highlighted in red to indicate a fixed value. As miter creates differential output between two copies of the locked circuit A and B, without loss of generality, suppose the point function of circuit A has output 0 and circuit B's has output 1, shown in red (vice versa in blue). Since both circuits have the same original cone, its output is identical to both A and B as they share the same input X. Without loss of generality, we assume the output of original cone under the DIP found by miter is logic 0. One can also assume with logic 1 instead. Hence, the output of AntiSAT block in circuit A is 0 while 1 for B's. As AntiSAT block has AND at the output, both inputs of this AND gate in B are 1, where $g = 1, \overline{g} = 1$. Then, we know that the DIP X obtained from the miter must be complementary to the fixed K_g, $X = \overline{K_g}$ to ensure all ones for g's AND tree of B. Following the analysis in Section III, the solver updates its CNF clauses with X and the oracle's output (logic 0 from the assumption). When this IO pair is applied to the locked circuit, the AntiSAT block gets a logic 0 output. Since DIP X gives $g = 1$ for circuit B, we still have $g = 1$ during CNF update. Then, $\overline{g} = 0$, as shown in Figure 8(b). As $\overline{g}$ is the NAND gate's output, all its inputs have logic 1. This uniquely determines all r-bit key $K_{\overline{g}}$, which is the complement of DIP X, $K_{\overline{g}} = X$, and identical to K_g, $K_{\overline{g}} = X = K_g$. SAT attack completes on the 2nd iteration since key $K_{\overline{g}}$ is already resolved. Therefore, the constraint on K_g helps SAT attack finish within one IO pair.

• Key constraint on $K_{\overline{g}}$: If the adversary decides to set key $K_{\overline{g}}$ constant instead, he/she will not get the same efficiency for key derivation as in fixing K_g. Suppose we constrain $K_{\overline{g}} = \{k_r, k_{r+1}, ..., k_{2r-1}\}$ to a constant r-bit vector. When SAT solver tries to find a satisfiable assignment to the miter circuit, following the same assumptions as before, we can derive that both A and B have the same logic 1 for $\overline{g}$ blocks. Having an output 1 at the NAND gate is equivalent to putting logic 0 to an AND gate. There are $2^r - 1$ possible solutions for the r-bit input to produce a logic 1 at $\overline{g}$'s output. Equivalently, there are $2^r - 1$ choices of DIP, satisfying the criterion of miter construction. When the tool updates SAT solver's CNF with DIP and output response, we get $\overline{g} = 1$ for NAND gate and $g = 0$ for AND gate. Since unknown key bits are in K_g of block g, a specific IO pair can prune only 1 incorrect key combination that results in $g = 1 (\neq 0)$. The total IO pairs required to remove all incorrect keys of the r-bit keyspace for K_g is $2^r - 1$. The total iterations required for SAT attack is 2^r. Therefore, by constraining $K_{\overline{g}}$, the adversary removes only one incorrect key and the overall SAT attack complexity remains exponential.

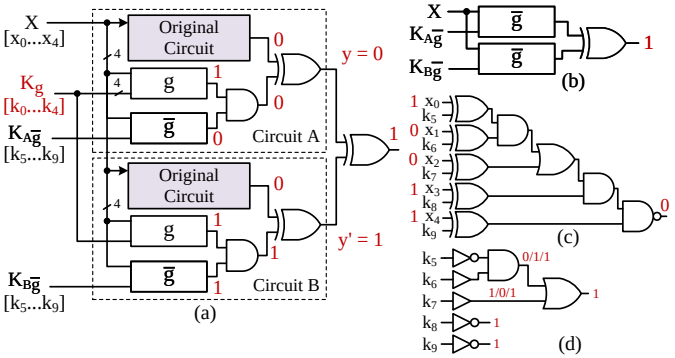

Figure 9: SAT attack on CAS-Lock with fixed K_g. (a) Miter construction and (b) Equivalent representation. (c) CNF update at SAT attack iteration 1. (d) Key pruning after iteration 1.

2) SAT attack analysis on CAS-Lock under key constraints: The same analysis on key constraints in AntiSAT can be applied to CAS-Lock, where the constraining of K_g or $K_{\overline{g}}$ leads to linear complexity in solving $K_{\overline{g}}$ or K_g, respectively. We illustrate with a $(2r = 10)$-bit CAS-Lock example, where block g and $\overline{g}$ have one OR gate each, as shown in Figure 9. Our analysis can be generalized and applied to any OR gate replacement inside the cascaded AND chain of g and $\overline{g}$. When SAT attack searches for a DIP X for miter, as shown in Figure 9(a), the CAS-Lock block of one copy (*i.e.,* A) has logic 0 while the other (*i.e.,* B) has logic 1. As K_g is fixed, the miter is essentially solving a differential output for $K_{\overline{g}}$ (Figure 9(b)). Suppose B's CAS-Lock block is 1, then it has $g = 1$ and $\overline{g} = 1$. The DIP X obtained by SAT solver must satisfy $g = 1$ as K_g is constant. The oracle response, identical to the analysis for AntiSAT, helps to determine a logic 0 for the CAS-Lock block, as no alteration of output logic occurred. The CNF update implicitly eliminates the wrong keys in $\overline{g}$ with $\overline{g} = 0$ under $g = 1$ and output 0 for the combined blocks. After the 1st iteration, k_8, k_9 are uniquely determined, which reduces the key space from 2^5 to 2^3 and is shown in Figure 9(d). Note that some of the keys k_5, k_6, k_7 will be determined in the same way in the 2nd iteration of the SAT attack. The attack will continue iterating until all key bits are uniquely determined.

B. Extending the point function analysis to TTLock and SFLL

TTLock [26] and SFLL [27], [28] do not have two sets of keys like AntiSAT and CAS-Lock. Both perturb unit (*PU*) and restore unit (*RU*) are serially XORed with the original circuit,

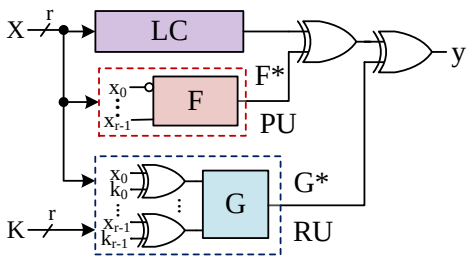

Figure 10: Generalized architecture of stripped functionality logic locking (SFLL). The functions F and G can be configured to implement TTLock [26] and SFLL-HD^h [27].

e.g., a logic cone (*LC*) of interest, as shown in Figure 10. Keys are in the restore unit (*RU*) only, where the perturb function (F^*) is key-free. The same analysis can be performed as PU and RU with the correct key implementing the same function even though different versions of SFLL have different output corruptibility. The output of PU, F^*, is logic 1 for only one input combination, where it alters the circuit behavior. The correct key helps flip back the perturbed logic and restores the original functionality as LC. Therefore, it must be true that the functional behavior for PU and RU are identical under the correct key so that LC's output is preserved. In other words, PU is the oracle for RU. If we can extract both PU and RU, we can then apply SAT attack on both circuits only, without requiring an oracle LC. As TTLock and SFLL perform logic synthesis after insertions of PU and RU, the adversary needs an accurate identification of PU and RU under logic optimization. The extraction of RU during post-synthesis is straightforward because commercial CAD tools cannot merge it inside LC or PU when the key is unknown, however, the challenging part is to retrieve PU since CAD tools may partially merge PU inside LC. Using the directed acyclic graph analysis [76], there are multiple candidates for PU with full input X. One only needs to apply SAT attack to all possible PUs with the extracted RU and perform key validation in the end. *Note that we do not need an unlocked chip (serves as the oracle for traditional SAT attack) as the oracle is already present in the synthesized LC & PU circuit.* Our future work is to find an efficient way to determine the valid oracle and identify the wrong ones from all extracted PUs.

VI. Future Directions

Even though point functions have demonstrated exponential iteration complexity, the adversary can formulate the attack with structural and functional analysis so that the complexity drops significantly. Although SAT attack has demonstrated linear trends in solving the secret key, we believe it is still possible for a logic design to achieve SAT resiliency. Here, we discuss how future locking schemes should consider a drastic increase in the hardness of their design against SAT attack from the example of c6288 benchmark. Besides targeting exponential iterations required for SAT attack, it may be feasible to significantly increase the overall time for SAT solver to find each satisfiable assignment for the miter circuit, which result in a longer computation time within each iteration.

As described in SAT attack [22], Subramanyan et al. stated that the multiplier benchmark c6288 is inherently challenging to SAT solvers, and was excluded from analysis. We locked its largest cone, N6288, in the same way as we did for other benchmarks in ISCAS'85, as described in Section IV. From the perspective of SAT attack iteration complexity, it remains linear with key size $|K|$, as shown on the bottom-right plot in Figure 5 and Column 2 of Table IV. This suggests that c6288 behaves identically to other ISCAS'85 benchmarks. In addition, one can also observe that the complexity can decrease when more keys are inserted, as shown in Figure 6 and Column 2 of Table IV. The question is, *what makes the circuit structure of a multiplier challenging to SAT solver?* To better analyze the SAT complexity in breaking c6288_N6288, we record the CPU time spent for each iteration, including the very last UNSAT round. We exclude the pre-processing time, *i.e.,* setting up arrays of literals, initializing solver, *etc*. The post-processing time is also excluded from the CPU time, *i.e.,* displaying the correct keys and the overall status, *etc*. Table IV lists the time duration for SAT solver to derive the correct keys. The 1st and 2nd columns list key sizes $|K|$ and IO pair count $|P|$. Column 3 is the total time the SAT solver spent, which consists of two parts, (i) time used for generating all IO pairs, Column 4, and (ii) time checking that no more DIP exists (UNSAT), Column 6, where the averaged time it takes to find each IO pair is in Column 5. Column 7 reports the time ratio of the UNSAT decision over the total time spent on SAT solver. The interesting observation is that the major time spent was not on finding DIPs to prune off keyspace, but was on the last iteration, where SAT solver tries various backtracking before getting the UNSAT decision. The total time devoted to generating the IO pairs is negligible compared to the time

Table IV: Anatomy of SAT attack time on c6288_N6288.

$\|K\|$	$\|P\|$	CPU time (s)				$\frac{\text{UNSAT}}{\text{Total}}$ (%)
		Total	IO Pairs	Average	UNSAT	
1	1	86.351	0.09108	0.09108	86.25948	99.894
2	2	84.439	0.10289	0.05145	84.33634	99.878
3	3	86.551	0.11019	0.03673	86.44092	99.872
4	4	88.804	0.11963	0.02991	88.68458	99.865
5	4	79.614	0.12705	0.03176	79.48717	99.840
6	4	62.048	0.11630	0.02908	61.93153	99.812
7	4	88.088	0.11822	0.02955	87.97006	99.865
8	4	66.762	0.11330	0.02832	66.64887	99.830
9	5	78.434	0.12385	0.02477	78.31049	99.842
10	7	62.018	0.14788	0.02113	61.87004	99.761
11	8	72.615	0.15925	0.01991	72.45534	99.780
12	6	66.560	0.19532	0.03255	66.36468	99.706
13	9	74.612	0.22130	0.02459	74.39026	99.703
14	8	78.492	0.14760	0.01845	78.34455	99.811
15	10	77.133	0.17205	0.01721	76.96051	99.776
16	11	83.077	0.23765	0.02161	82.83926	99.713
17	11	85.083	5.70418	0.51856	79.37841	93.295
18	15	72.317	0.30082	0.02006	72.01650	99.584
19	15	89.654	0.34619	0.02308	89.30831	99.613
20	14	92.588	0.32586	0.02328	92.26268	99.648
21	15	67.431	0.45250	0.03017	66.97835	99.328
22	12	80.299	0.26642	0.02220	80.03259	99.668
23	19	88.228	0.63912	0.03364	87.58904	99.275
24	15	76.825	0.42104	0.02807	76.40369	99.451
25	20	88.295	2.48402	0.12420	85.81125	97.186
30	16	73.065	0.84954	0.05310	72.21507	98.837
35	29	86.737	14.53748	0.50129	72.19920	83.239
40	27	149.097	13.34636	0.49431	135.7502	91.048
45	41	1130.466	18.31241	0.44664	1112.154	98.380
50	37	84.404	6.16717	0.16668	78.23738	92.693
55	45	1188.844	57.14645	1.26992	1131.698	95.193

spent in the very last iteration (UNSAT). In particular, the time duration for UNSAT in solving c6288 benchmark cones with respect to the total time span is generally over 90%.

For future SAT-resilient locking schemes, one of the design-goals could be on achieving the same degree of time complexity for each SAT-solving iteration as the very last iteration of c6288 benchmark. One of the possible considerations for logic locking designs is to increase the total backtracks and logic reassignment required for SAT solvers to find a DIP in every iteration so that a longer time duration can be achieved.

VII. Conclusion

In this paper, we provide a new perspective to analyze the efficiency of the SAT attack based on the CNF clause updates inside SAT solver. In each iteration, SAT attack records the interdependencies between key bits from a distinguishing input pattern and its output response. Any locked circuit with multiple logic cones facilitates the incorrect key removal as the effect of keys is propagated to multiple outputs. We further investigate the SAT attack complexity with the same cone of increasing key sizes. A non-monotonically increase in SAT complexity under increased key sizes is reported for the first time, where the insertion of additional key bits does not guarantee a strict linear growth in the SAT attack iteration complexity. Instead, this phenomenon of complexity drop happens to all ISCAS'85 benchmark cones. We subsequently provided an explanation of this observation from the oracle's response and logic gate types. It explains why more incorrect keys are eliminated from the keyspace with a particular IO pair. In addition, we give analytical reasoning to show how the constraining of key bits for post-SAT solutions like AntiSAT and CAS-Lock would aggressively reduce the key search down to constant or linear complexity. Finally, we furnish our discussions on SAT attack complexity analysis with novel observations on breaking the multiplier benchmark c6288, along with future directions.

ACKNOWLEDGEMENT

This work was supported by the National Science Foundation under Grant Number CNS-1755733. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] W. Shih, 2022. Intel's $88 billion European expansion is part of a new phase in the globalization of the semiconductor industry, Forbes.

[2] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending Piracy of Integrated Circuits," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 1069–1074, 2008.

[3] Y. Alkabani and F. Koushanfar, "Active Hardware Metering for Intellectual Property Protection and Security," in *USENIX security symposium*, pp. 291–306, 2007.

[4] R. S. Chakraborty and S. Bhunia, "Hardware Protection and Authentication Through Netlist Level Obfuscation," in *Proc. of IEEE/ACM International Conf. on Computer-Aided Design*, pp. 674–677, 2008.

[5] Y. Alkabani, F. Koushanfar, and M. Potkonjak, "Remote activation of ICs for piracy prevention and digital right management," in *Proc. of IEEE/ACM Int. Conf. on Computer-aided design*, pp. 674–677, 2007.

[6] J. Huang and J. Lach, "IC Activation and User Authentication for Security-Sensitive Systems," in *IEEE International Workshop on Hardware-Oriented Security and Trust*, pp. 76–80, 2008.

[7] 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.

[8] U. Guin, Q. Shi, D. Forte, and M. M. Tehranipoor, "FORTIS: a comprehensive solution for establishing forward trust for protecting IPs and ICs," *Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 21, no. 4, p. 63, 2016.

[9] E. Castillo, U. Meyer-Baese, A. García, L. Parrilla, and A. Lloris, "IPP@HDL: Efficient Intellectual Property Protection Scheme for IP Cores," *IEEE Trans. on VLSI (TVLSI)*, pp. 578–591, 2007.

[10] M. Tehranipoor and C. Wang, *Introduction to Hardware Security and Trust*. Springer Science & Business Media, 2011.

[11] M. Tehranipoor, U. Guin, and D. Forte, *Counterfeit Integrated Circuits: Detection and Avoidance*. Springer International Publishing, 2015.

[12] S. Bhunia and M. Tehranipoor, *Hardware Security: A Hands-on Learning Approach*. Morgan Kaufmann, 2018.

[13] R. Torrance and D. James, "The State-of-the-Art in IC Reverse Engineering," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 363–381, 2009.

[14] R. Torrance and D. James, "Reverse Engineering in the Semiconductor Industry," in *2007 IEEE Custom Integrated Circuits Conf.*, pp. 429–436, IEEE, 2007.

[15] R. Torrance and D. James, "The state-of-the-art in semiconductor reverse engineering," in *Proc. of the Design Automation Conference*, pp. 333–338, 2011.

[16] S. E. Quadir, J. Chen, D. Forte, N. Asadizanjani, S. Shahbazmohamadi, L. Wang, J. Chandy, and M. Tehranipoor, "A survey on chip to system reverse engineering," *ACM journal on emerging technologies in computing systems (JETC)*, vol. 13, no. 1, pp. 1–34, 2016.

[17] E. Charbon, "Hierarchical watermarking in IC design," in *Proc. of the IEEE Custom Integrated Circuits Conference*, pp. 295–298, 1998.

[18] A. B. Kahng, J. Lach, W. H. 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 CAD of Integrated Circuits and Systems*, pp. 1236–1252, 2001.

[19] G. Qu and M. Potkonjak, *Intellectual Property Protection in VLSI Designs: Theory and Practice*. Springer Sc. & Business Media, 2007.

[20] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proc. of Annual Design Automation Conference*, pp. 83–89, 2012.

[21] 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*, pp. 410–424, 2015.

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

[23] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 236–241, 2016.

[24] Y. Xie and A. Srivastava, "Anti-SAT: Mitigating SAT Attack on Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 2, pp. 199–207, 2019.

[25] 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.

[26] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu, "Tt-lock: Tenacious and traceless logic locking," in *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 166–166, IEEE Computer Society, 2017.

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

[28] A. Sengupta, M. Nabeel, N. Limaye, M. Ashraf, and O. Sinanoglu, "Truly stripping functionality for logic locking: A fault-based perspective," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[29] B. Shakya, X. Xu, M. Tehranipoor, and D. Forte, "Cas-lock: A security-corruptibility trade-off resilient logic locking scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 175–202, 2020.

[30] M. Yasin, A. Sengupta, B. C. Schafer, Y. Makris, O. Sinanoglu, and J. J. Rajendran, "What to Lock?: Functional and Parametric Locking," in *Proc. of Great Lakes Symposium on VLSI*, pp. 351–356, 2017.

[31] J. Zhou and X. Zhang, "Generalized sat-attack-resistant logic locking," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2581–2592, 2021.

[32] Y. Liu, M. Zuzak, Y. Xie, A. Chakraborty, and A. Srivastava, "Strong anti-sat: Secure and effective logic locking," in *2020 21st Int. Symposium on Quality Electronic Design (ISQED)*, pp. 199–205, IEEE, 2020.

[33] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "Cyclic obfuscation for creating sat-unresolvable circuits," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pp. 173–178, 2017.

[34] S. Roshanisefat, H. Mardani Kamali, and A. Sasan, "Srclock: Sat-resistant cyclic logic locking for protecting the hardware," in *Proceedings of 2018 Great Lakes Symposium on VLSI*, pp. 153–158, 2018.

[35] A. Rezaei, Y. Li, Y. Shen, S. Kong, and H. Zhou, "Cycsat-unresolvable cyclic logic encryption using unreachable states," in *Proc. of the 24th Asia and South Pacific Design Automation Conf.*, pp. 358–363, 2019.

[36] S. Roshanisefat, H. M. Kamali, H. Homayoun, and A. Sasan, "Sathard 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.

[37] A. Rezaei, Y. Shen, S. Kong, J. Gu, and H. Zhou, "Cyclic locking and memristor-based obfuscation against cycsat and inside foundry attacks," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 85–90, IEEE, 2018.

[38] X.-M. Yang, P.-P. Chen, H.-Y. Chiang, C.-C. Lin, Y.-C. Chen, and C.-Y. Wang, "Looplock 2.0: An enhanced cyclic logic locking approach," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 1, pp. 29–34, 2021.

[39] 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 *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 405–410, IEEE, 2018.

[40] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Full-lock: Hard distributions of sat instances for obfuscating circuits using fully configurable logic and routing blocks," in *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

[41] G. Kolhe, H. M. Kamali, M. Naicker, T. D. Sheaves, H. Mahmoodi, P. S. Manoj, H. Homayoun, S. Rafatirad, and A. Sasan, "Security and complexity analysis of lut-based obfuscation: From blueprint to reality," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2019.

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

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

[44] 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.

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

[46] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Interlock: An intercorrelated logic and routing locking," in *2020 IEEE/ACM Int. Conf. On Computer Aided Design (ICCAD)*, pp. 1–9, IEEE, 2020.

[47] D. Zhang, M. He, X. Wang, and M. Tehranipoor, "Dynamically obfuscated scan for protecting ips against scan-based attacks throughout supply chain," in *2017 IEEE 35th VLSI Test Symposium (VTS)*, pp. 1–6, IEEE, 2017.

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

[49] 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*, vol. 9, no. 4, pp. 2109–2124, 2019.

[50] S. Potluri, A. Aysu, and A. Kumar, "SeqL: Secure Scan-Locking for IP Protection," *arXiv preprint arXiv:2005.13032*, 2020.

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

[52] 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.

[53] F. Koushanfar, "Active hardware metering by finite state machine obfuscation," in *Hardware Protection through Obfuscation*, pp. 161–187, Springer, 2017.

[54] J. Dofe and Q. Yu, "Novel dynamic state-deflection method for gate-level design obfuscation," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 273–285, 2017.

[55] T. Meade, Z. Zhao, S. Zhang, D. Pan, and Y. Jin, "Revisit sequential logic obfuscation: Attacks and defenses," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, IEEE, 2017.

[56] 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 *2020 IEEE 38th VLSI Test Symposium (VTS)*, pp. 1–6, IEEE, 2020.

[57] L. Li, S. Ni, and A. Orailoglu, "Janus: Boosting logic obfuscation scope through reconfigurable fsm synthesis," in *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 292–303, IEEE, 2021.

[58] 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 *2022 Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, pp. 1323–1328, IEEE, 2022.

[59] Y. Xie and A. Srivastava, "Delay locking: Security enhancement of logic locking against ic counterfeiting and overproduction," in *Proceedings of the 54th Annual Design Automation Conf.*, pp. 1–6, 2017.

[60] G. L. Zhang, B. Li, B. Yu, D. Z. Pan, and U. Schlichtmann, "Timingcamouflage: Improving circuit security against counterfeiting by unconventional timing," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 91–96, IEEE, 2018.

[61] M. Alam, S. Ghosh, and S. S. Hosur, "Toic: timing obfuscated integrated circuits," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, pp. 105–110, 2019.

[62] J. Sweeney, M. Zackriya V, S. Pagliarini, and L. Pileggi, "Latch-Based Logic Locking," *arXiv preprint arXiv:2005.10649*, 2020.

[63] K. Z. Azar, H. M. Kamali, S. Roshanisefat, H. Homayoun, C. P. Sotiriou, and A. Sasan, "Data flow obfuscation: A new paradigm for obfuscating circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 4, pp. 643–656, 2021.

[64] M. S. Rahman, R. Guo, H. M. Kamali, F. Rahman, F. Farahmandi, and M. Abdel-Moneum, "O'clock: Lock the clock via clock-gating for soc ip protection," in *Design Automation Conf. (DAC)*, pp. 1–6, 2022.

[65] C. Pilato, F. Regazzoni, R. Karri, and S. Garg, "Tao: Techniques for algorithm-level obfuscation during high-level synthesis," in *Proceedings of the 55th Annual Design Automation Conference*, pp. 1–6, 2018.

[66] C. Pilato, A. B. Chowdhury, D. Sciuto, S. Garg, and R. Karri, "Assure: Rtl locking against an untrusted foundry," *IEEE Transactions on VLSI Systems*, vol. 29, no. 7, pp. 1306–1318, 2021.

[67] M. Zuzak, Y. Liu, and A. Srivastava, "A resource binding approach to logic obfuscation," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 235–240, IEEE, 2021.

[68] M. R. Muttaki, R. Mohammadivojdan, M. Tehranipoor, and F. Farahmandi, "Hlock: Locking ips at the high-level language," in *2021 58th ACM/IEEE Design Automation Conf. (DAC)*, pp. 79–84, IEEE, 2021.

[69] N. Limaye, A. B. Chowdhury, C. Pilato, M. T. Nabeel, O. Sinanoglu, S. Garg, and R. Karri, "Fortifying rtl locking against oracle-less (untrusted foundry) and oracle-guided attacks," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 91–96, IEEE, 2021.

[70] C. Karfa, T. A. Khader, Y. Nigam, R. Chouksey, and R. Karri, "Host: Hls obfuscations against smt attack," in *2021 Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, pp. 32–37, IEEE, 2021.

[71] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal attacks on logic locking and camouflaging techniques," *Transactions on Emerging Topics in Computing*, 2017.

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

[73] X. Xu, B. Shakya, M. M. Tehranipoor, and D. Forte, "Novel Bypass Attack and BDD-based Tradeoff Analysis Against All Known Logic Locking Attacks," in *International Conference on Cryptographic Hardware and Embedded Systems*, pp. 189–210, 2017.

[74] Y. Shen and H. Zhou, "Double DIP: Re-Evaluating Security of Logic Encryption Algorithms," in *Proceedings of the Great Lakes Symposium on VLSI*, pp. 179–184, 2017.
[75] Y. Shen, A. Rezaei, and H. Zhou, "Sat-based bit-flipping attack on logic encryptions," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 629–632, IEEE, 2018.
[76] D. Sirone and P. Subramanyan, "Functional analysis attacks on logic locking," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 936–939, 2019.
[77] 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*, pp. 75–83, 2019.
[78] A. Jain, T. Rahman, and U. Guin, "ATPG-Guided Fault Injection Attacks on Logic Locking," in *IEEE Physical Assurance and Inspection of Electronics (PAINE)*, pp. 1–6, 2020.
[79] A. Jain, Z. Zhou, and U. Guin, "TAAL: tampering attack on any key-based logic locked circuits," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 26, no. 4, pp. 1–22, 2021.
[80] N. Limaye, S. Patnaik, and O. Sinanoglu, "Fa-SAT: Fault-aided SAT-based Attack on Compound Logic Locking Techniques," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1166–1171, IEEE, 2021.
[81] A. Sengupta, N. Limaye, and O. Sinanoglu, "Breaking cas-lock and its variants by exploiting structural traces," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 418–440, 2021.
[82] L. Alrahis, S. Patnaik, F. Khalid, M. A. Hanif, H. Saleh, M. Shafique, and O. Sinanoglu, "GNNUnlock: Graph Neural Networks-based Oracle-less Unlocking Scheme for Provably Secure Logic Locking," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 780–785, IEEE, 2021.
[83] Z. Han, M. Yasin, and J. J. Rajendran, "Does logic locking work with {EDA} tools?," in *30th USENIX Security Symposium (USENIX Security 21)*, pp. 1055–1072, 2021.
[84] 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*, vol. 17, pp. 744–759, 2022.
[85] D. Duvalsaint, X. Jin, B. Niewenhuis, and R. Blanton, "Characterization of Locked Combinational Circuits via ATPG," in *IEEE International Test Conference (ITC)*, pp. 1–10, 2019.
[86] 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)*, pp. 49–56, 2017.
[87] Y. Shen, Y. Li, A. Rezaei, S. Kong, D. Dlott, and H. Zhou, "Besat: Behavioral sat-based attack on cyclic logic encryption," in *Proc. of the 24th Asia and South Pacific Design Automation Conf.*, pp. 657–662, 2019.
[88] K. Shamsi, D. Z. Pan, and Y. Jin, "Icysat: Improved sat-based attacks on cyclic locked circuits," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–7, IEEE, 2019.
[89] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "Nngsat: Neural network guided sat attack on logic locked complex structures," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, IEEE, 2020.
[90] 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 *2021 IEEE/ACM International Conf. On Computer Aided Design (ICCAD)*, pp. 1–9, IEEE, 2021.
[91] 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.
[92] 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," *arXiv preprint:1906.07806*, 2019.
[93] N. Limaye and O. Sinanoglu, "Dynunlock: Unlocking scan chains obfuscated using dynamic keys," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 270–273, IEEE, 2020.
[94] M. El Massad, S. Garg, and M. Tripunitara, "Reverse engineering camouflaged sequential circuits without scan access," in *2017 IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, pp. 33–40, IEEE, 2017.
[95] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, "Kc2: Key-condition crunching for fast sequential circuit deobfuscation," in *2019 Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, pp. 534–539, IEEE, 2019.
[96] S. Roshanisefat, H. Mardani Kamali, H. Homayoun, and A. Sasan, "Rane: An open-source formal de-obfuscation attack for reverse engineering of logic encrypted circuits," in *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, pp. 221–228, 2021.
[97] K. Azar, H. Kamali, F. Farahmandi, and M. Tehranipoor, "Warm up before circuit de-obfuscation? an exploration through bounded-model-checkers," in *International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 1–4, 2022.
[98] Y. Hu, Y. Zhang, K. Yang, D. Chen, P. A. Beerel, and P. Nuzzo, "Funsat: Functional corruptibility-guided sat-based attack on sequential logic encryption," in *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 281–291, IEEE, 2021.
[99] A. Saha, H. Banerjee, R. S. Chakraborty, and 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.
[100] A. Chakraborty, Y. Liu, and A. Srivastava, "Timingsat: Timing profile embedded sat attack," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–6, ACM, 2018.
[101] 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*, pp. 97–122, 2019.
[102] Y. Xie and A. Srivastava, "Anti-sat: Mitigating sat attack on logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 2, pp. 199–207, 2018.
[103] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 12, pp. 1377–1394, 2002.
[104] Synopsys Design Compiler. Synopsys, Inc., 2017.
[105] https://www.mathworks.com/help/matlab/ref/digraph.flipedge.html.
[106] https://www.mathworks.com/help/matlab/ref/graph.bfsearch.html.
[107] K. Juretus and I. Savidis, "Increasing the SAT Attack Resiliency of In-Cone Logic Locking," in *International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2019.

Yadi Zhong (S'20) is currently pursuing her Ph.D. in Computer Engineering from the Department of Electrical and Computer Engineering, Auburn University, AL, USA. She received her B.E. degree from the same university in 2020. Her research interests are logic locking, fault injection and hardware security, and post-quantum cryptography. She led a student team that received several awards including 1st place in HeLLO: CTF'21 and Hack@CHES 2021 and 2nd place in Hack@SEC 2021. She was the recipient of the Auburn University Presidential Graduate Research Fellowships in 2020. She is a student member of the IEEE.

Ujjwal Guin (S'10–M'16) received his PhD degree from the Electrical and Computer Engineering Department, University of Connecticut, in 2016. He is currently an Assistant Professor in the Electrical and Computer Engineering (ECE) Department of Auburn University, Auburn, AL, USA. He received his B.E. degree from the Department of Electronics and Telecommunication Engineering, Bengal Engineering and Science University, India, in 2004 and his M.S. degree from the ECE Department, Temple University, Philadelphia, PA, USA, in 2010. Dr. Guin's current research interests include hardware security, blockchain, and VLSI design & test. He has authored several journals and refereed conference papers. He serves on organizing committees of HOST, VTS, ITC-India, and PAINE. He also serves on technical program committees in several reputed conferences, such as DAC, HOST, ITC, VTS, PAINE, VLSID, GLSVLSI, ISVLSI, and Blockchain. He is a member of both the IEEE and ACM.