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 (SAT)-based attacks have rendered most locking schemes ineffective. This gives rise to numerous defenses and new locking methods to achieve SAT resiliency. This article 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—Boolean satisfiability (SAT), conjunctive normal form (CNF), integrated circuit (IC) overproduction, intellectual property (IP) piracy, logic locking, reverse engineering (RE).

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 system-on-a-chip (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 multibillion dollar investment [1]. As a result, an 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.

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: 1) 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], [3], [4], [5], [6], [7], [8] and 2) piracy of IPs, where an entity in the supply chain can use, modify and/or sell functional IPs illegally [9], [10], [11], [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 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. Fig. 1 shows an abstract representation of logic locking. In addition to logic locking, hardware watermarking [17], [18], [19] could identify and prevent copying a netlist to a certain extent; however, it does not offer a proactive protection mechanism. The initial efforts in logic locking [2], [7], [20], [21] and hardware watermarking [17], [18], [19] were broken by Boolean Satisfiability (SAT) attack [22]. The distinguishing
input patterns (DIPs), 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 this article. 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], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], cyclic-based [34], [35], [36], [37], [38], [39], LUT/routing-based [40], [41], [42], [43], [44], [45], [46], [47], scan and finite-state machine (FSM)-based lockings [48], [49], [50], [51], [52], [53], [54], [55], [56], [57], [58], [59], timing-based [60], [61], [62], [63], [64], [65], and high-level synthesis (HLS)-based [66], [67], [68], [69], [70], [71]. Concurrently, multiple attacks [47], [72], [73], [74], [75], [76], [77], [78], [79], [80], [81], [82], [83], [84], [85], [86], [87], [88], [89], [90], [91], [92], [93], [94], [95], [96], [97], [98], [99], [100], [101], [102], [103], [104], [105] against these logic locking techniques arise. In addition, various machine learning-based attacks [106], [107], [108], [109], which are structural in nature and do not require the oracle, target the identification and recovery of keys that are obfuscated after synthesis process by commercial CAD tools.

This article 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 input 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 [24], [27], [29].

The contributions of this article are summarized as follows.

1) **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 article uses examples to describe the step-by-step analysis of incorrect key elimination for each DIP. The interdependencies 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.

2) **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 of solving the correct key. Instead, it is common to observe a local decrease in iteration complexity. To the best of our knowledge, we are the first to show the local reduction in attack complexity with a larger key size. We believe that the findings of this article 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.

3) **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 proper constraining of subkeys 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

**TABLE I**

| Locking Type   | Techniques          | Attacks |
|---------------|---------------------|---------|
| Point function| [23]–[33]           | [72]–[89] |
| Cyclic        | [34]–[39]           | [90]–[95] |
| LUT           | [40]–[47]           | [47], [93], [94] |
| Scan          | [48]–[53]           | [95]–[97] |
| FSM           | [54]–[59]           | [98], [103] |
| Timing        | [60]–[65]           | [104], [105] |
| HLS           | [66]–[71]           | [84], [85], [104] |

![Fig. 1. Overview of logic locking. (a) Architecture of a locked circuit. (b) Original design. (c) XOR-based locking, with secret key of k0k1 = 01.](image-url)
effectively reduce the exponential iteration complexity to linear. We present insights on how the same analysis can be applied to TTTLock and various versions of stripped functionality logic locking (SFLL).

4) 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 a 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 this article is organized as follows. We introduce the background of SAT attack and various locking methods in Section II. The interdependency 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 this article 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] exploited the idea of combinational equivalence checking with miter circuit and SAT [110] to attack logic locking schemes. This oracle-guided attack successfully derives the secret key of various logic locking techniques [2], [7], [17], [18], [19], [20], [21] within a short time frame. The SAT attack requires two circuits, the original circuit, \( CO(X, Y) \), 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, \( C(X, K_c, Y) = CO(X, Y) \). 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.

1) Finding the 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.

2) Deriving the Correct Key: Upon obtaining a DIP \( X_i \), SAT attack acquires the actual output \( Y_i \) from oracle simulation, \( CO(X_i, Y_i) \), 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, wherever 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 [24], [29] 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 the original SAT attack program [22] includes two preload vectors (all zeros and all ones) at the initial setup, before invoking SAT solver with the miter circuit. The number of IO pairs \( |P| \) used to derive the correct key is one more than the number of total iterations \( TI \), \( |P| = 2 + (TI - 1) = TI + 1 \). This is because the last iteration does not produce a DIP. It is clear that the IO pair count for determining the secret key of a locked circuit is in the same order as the iteration count, which only differs by a constant of 1. For better analyzing the iteration complexity of SAT attack on c6288 benchmark (see Section VI-A, we modify the original program [22] by disabling both preload vectors, resulting in \( |P| = TI - 1 \).

B. SAT Resistant Logic Locking Techniques and Attacks

As SAT attack [22] successfully breaks various logic locking techniques [2], [7], [17], [18], [19], [20], [21], it propels
the research community to explore new locking schemes [23], [24], [25], [26], [27], [28], [29], [30], [31], [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 [24], [31], [32] and CAS-Lock [29] configure the point function with two complementary blocks $g$ and $\bar{g}$. 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 [72], [73], [74], [75], [76], [77], [78], [79], [80], [81], [82], [83], [84], [85], [86] 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 [73], [74], [75], [76], [77], [81], [82], [87], [88].

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 iteration with a DIP and its output response. The CNF consists of multiple clauses connected with AND ($\land$) and OR ($\lor$). One or more literals are joined by OR ($\lor$) 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 Fig. 2(a), is transformed into the functions of keys $(K_f, K_O)$, shown in Fig. 2(b), where $K_O$ can be derived from $K_f$. Further, $(K_f, K_O)$ can be expanded further and is shown in Fig. 2(c). Any key in $K_O$, e.g., $K'_i$, is dependent upon key bits $K_j^i$; i.e., $K_j^i = f(K'_i)$, where $K'_i \subseteq K_f$. In addition, the combination of some key bits, e.g., $K'_1, K'_2 \subseteq K_f$, produces a deterministic output, i.e., either logic 0 or 1

$C(K_f, K_O) \iff \begin{cases} C(K'_i, K'_j), \text{ where } K'_j = f(K'_i), K'_j \notin K'_i; i, j, t = 1, 2, \ldots \\ C(K'_1, 0, 1), \text{ where } 0, 1 = f(K'_1); i, s = 1, 2, \ldots \end{cases}$

This key-dependent function $C(K_f, K_O)$ reveals additional information on the interdependency between key bits, e.g., $K'_1 = f(K'_i)$ and $0, 1 = f(K'_i)$, 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 nonoverlapping 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 when key gates are placed under overlapping logic cones.

A. SAT Attack for Locked Cone With One Output

In this section, we examine 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.

Let us consider an example circuit with four inputs $x_0, \ldots, x_3$ and 1 output $y_0$ of Fig. 3(a). Fig. 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 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, $(X_1) = \{x_0, x_1, \ldots, x_3\} = \{1111\}$. The output response $Y_1 = 1$ is obtained from oracle simulation with input $X_1$. SAT attack then records this IO pair $P_1 = \{X_1; Y_1\} = \{x_0, \ldots, x_3; y_0\} = \{1111; 1\}$. We show in detail how the locked circuit’s CNF gets updated under $P_1$, where the search space is shrunk 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] [Fig. 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

$$C_1 = (17 \lor 18 \lor 16) \land (17 \lor 16) \land (18 \lor 16)$$

$$\land (2 \lor 3 \lor 19) \land (2 \lor 19) \land (3 \lor 19)$$

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
On the 2nd iteration, SAT attack returns the second IO pair \( P_2 = \{X_2; Y_2\} = \{1101; 0\} \), as in Fig. 3(d). Using the derivation we performed for the first iteration, the circuit representation of the added CNF clauses is shown in Fig. 3(g), which again is a function between the 3 key bits, \( k_0 = \text{AND}(k_1, k_2) \) [Fig. 3(h)]. It further shrinks the remaining keyspace in half, with only two keys left valid. Fig. 3(i) shows the 3rd IO pair \( P_3 = \{0111; 0\} \), whose CNF \( C(X_3, K, Y_3) \) and its equivalent relation \( k_0 = \text{AND}(k_1, k_2) \) are illustrated in Fig. 3(j) and (k), respectively. The combined effect of these three IO pairs, \( P_1, P_2, P_3 \), Fig. 3(e), (h), and (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 \( C(X_1, K, Y_1) \) 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.

Let us consider a circuit with two outputs, \( y_0 \) and \( y_1 \), as shown in Fig. 4(a). The locked circuit, as shown in Fig. 4(b), has three key bits, \( k_0, k_1, k_2 \), with the same locations as in Fig. 3(b). It differs from the locked circuit in Fig. 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 \( G_0, G_1, G_2, G_6, G_8, G_{10}, G_{12} \); the other with output \( y_1 \), inputs \( x_2, x_3, x_4, x_5 \), key \( k_2 \), and gates \( G_2, G_3, G_4, G_{22} \). 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 Fig. 3(b). Fig. 4(c) illustrates the 1st IO pair \( P_1 = \{X_1; Y_1\} = \{x_0, \ldots, x_3; y_0, y_1\} = \{011001; 00\} \), its equivalent CNF expression of \( C(X_1, K, Y_1) \) (abbreviated as \( C_1 \)) is expressed in

\[
C_1 = (21 \lor 22 \lor 20) \land (21 \lor 20) \land (22 \lor 20) \land (2 \lor 3 \lor 23) \\
\land (22 \lor 25 \lor 12) \land (22 \lor 12) \land (25 \lor 12) \land (6 \lor 7 \lor 25) \\
\land (6 \lor 25) \land (7 \lor 23) \land (8 \lor 20 \lor 11) \land (8 \lor 20 \lor 11) \\
\land (8 \lor 20 \lor 11) \land (9 \lor 23 \lor 21) \land (9 \lor 23 \lor 21) \\
\land (9 \lor 23 \lor 21) \land (9 \lor 23 \lor 21) \\
\land (9 \lor 23 \lor 21)
\]

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
With straightforward that node 23, the output of AND gate $G$.

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

\[
\begin{align*}
\land (10 \lor 24 \lor 22) \land (10 \lor 24 \lor 22) & \land (10 \lor 24 \lor 22).
\end{align*}
\]

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 (Fig. 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 $4 - 7$ ($x_2, \ldots, x_5$). 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

\[
C_1 = (21 \lor 20) \land (21 \lor 20) \land (8 \lor 20) \land (8 \lor 20) \land (9 \lor 21) \land (10).
\]

With $C_1$, SAT attack determines key bit $k_2 = 1$, along with key-dependent equation $k_0 = k_1$, as shown in Fig. 4(c). With the 2nd IO pair $P_2 = \{X_2; Y_2\} = \{010101; 00\}$, as in Fig. 4(d), SAT attack 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.

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 nonoverlapping 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 the decrease in iteration complexity with increased key sizes 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 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 increased key size.

The overview of SAT attack complexity analysis on the same logic cone is summarized in the following steps: 1) benchmark synthesis; 2) cone analysis and the largest cone extraction; 3) iterative insertion of key bits; and 4) SAT attack iteration complexity aggregation. Synthesis is performed under 32-nm technology libraries in Synopsys Design Compiler [111]. Fig. 5 shows an overall linear trend in total
attack iterations under increased key sizes for nonoverlapping 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 and breadth-search from each primary output. The ordered node list obtained in breadth-first search is used for determining: 1) the largest cone (or cones if a tie) by node count and 2) 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 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. Fig. 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 gate count, but a significant overlap of gates exists. Please note that these logic cones all have reconvergent fanouts.

There are two observations from Fig. 5. First, the overall complexity increase is not exponential, but linear. This means that, on average, the attack removes an exponential (or subexponential) number of incorrect keys per iteration. Second, all nine benchmark cones exhibit the local nonmonotonically complexity increase when additional keys are inserted. Note that a monotonic function \( f \) is either an entirely nonincreasing or nondecreasing function, where its first derivative does not change sign. Now, \( f \) is called monotonically increasing if \( \forall x, y, f(x) \leq f(y) \) for \( x \leq y \). We denote a function as nonmonotonically increasing if it increases globally (on average), but not monotonic.

Fig. 6 shows the zoomed-in view of SAT attack iteration 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 nonmonotonically increase in complexity is also observed in all the other benchmark cones. Note that the same nonmonotonic behavior for the iteration complexity can still be observed under a different initialization seed setup. In addition, a nonmonotonic linear increase can be observed in the averaged linear iteration complexity with 100 different seeds, as shown in Fig. 7 for c1355-G1350 and c1908-N2811. However, it does not suggest or infer that the minimum DIP count for solving each locked
plexity is observed in all cones. The question is, what causes the SAT attack to have such complexity drops when more keys are used? Another example shown in Fig. 8 where the effectiveness of individual IO pairs in eliminating incorrect keys is 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 which results in a reduction in iteration. Fig. 8(a) shows the circuit, where four keys \((k_0, k_3)\) are added. Note that an OR gate \((G_k)\) is located at the cone output. The 1st IO pair \(P_1 = \{0000000:1\}\) from SAT attack, and the 2nd 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.

Fig. 7. Average iteration complexity of the SAT attack on locked benchmarks c1355-G1350 and c1908-N2811 with 100 different seed setups for SAT solver Lingeling.

Fig. 8. Key elimination. (a) Original circuit, (b) locked circuit with 4 keys, (c) 1st IO pair \(P_1 = \{0000000:1\}\) from SAT attack, and (d) 2nd 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.

using logic propagation, and it can be shown that there exist only two incorrect keys, Table III, column 2. Any key combinations that cause an output mismatch with the oracle’s are marked with \(\times\), indicating an incorrect key value implicitly removed from keyspace; key value(s) which produces the same output as the oracle’s is noted with \(\checkmark\). From the 1st iteration, we observe fewer incorrect keys (i.e., \(2 \ll [2^4/2]\) are removed than the 2nd iteration (i.e., \(14 \gg [2^4/2]\)) 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 Fig. 8(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 \(G_3\) 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 into two halves, where keys in one half are independent of the keys in the other half. This is equivalent to splitting the logic cone into two subcones based on input 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.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
of the SAT attack, the iteration count can go down significantly. This leads to a nonmonotonically 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 [24], CAS-Lock [29], TTLock [26], and SFLL [27], and explains why they can also be broken by SAT-based attacks. In this section, we present a case study on how our proposed SAT attack analysis (see Sections III and IV) can be used to analyze the attack complexity of key-bit mapping (KBM) & SAT [82], a modified version of SAT attack with key constraints. The following analysis clarifies how and why SAT attack can still be effective in breaking AntiSAT and CAS-Lock under proper key constraints. Note that we are not proposing any new attacks but rather providing explanations to demonstrate that: 1) AntiSAT with fixed \( K_g \) requires only a single IO pair to determine the secret key and 2) the linear complexity for CAS-Lock under the same constraint. We also show that adding additional key constraints on \( K_g \) would not yield any extra benefits on the complexity reduction to an adversary for breaking AntiSAT-based locking designs. Note that the adversarial model for logic locking follows the same Kerckhoffs’s principle as in modern cryptography, where the security of the system is based on the secret key and not on the obscurity of the algorithm used [83], [107], [109]. For point function-based locking techniques such as AntiSAT and CAS-Lock, we assume that the attacker has full knowledge of the locking scheme and the existence of a comparator logic inside the locked netlist.

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 \( g' \), where \( K_g \) and \( K_g' \) are inside \( g \) and \( g' \), respectively. Sengupta et al. [82] have shown an effective approach to reducing AntiSAT and CAS-Lock to polynomial complexity with KBM & SAT, where KBM separates \( K_g \) and \( K_g' \) 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: 1) AntiSAT with fixed \( K_g \) requires only one IO pair to determine the secret key and 2) linear complexity for CAS-Lock under the same constraint. However, constraining \( K_g' \) 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 \( K_g' \), 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 \( K_{g'} \) is kept constant instead.

a) Key constraint on \( K_g \): Let us consider a circuit with \( r \)-bit input \( X = \{x_0, x_1, \ldots, x_{r-1}\} \), 1-bit output \( y \), locked with \( 2r \)-bit keys of \( K_g = \{k_0, \ldots, k_{r-1}\} \) and \( K_{g'} = \{k_r, \ldots, k_{2r-1}\} \) of \( r \)-bit each. We assume the attacker already knows the bit locations for \( K_g \) using the KBM of [82]. Furthermore, the \( r \)-bit \( K_g = \{k_0, \ldots, k_{r-1}\} \) is set to a constant vector. SAT attack is able to find an IO pair and uniquely determines all bits of \( K_g \). Fig. 9(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 (and vice versa in blue). Since both circuits have the same original cone, their output is identical to both A and B as they share the same input \( X \). Without loss of generality, we assume the output of the original cone under the DIP found by the 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 \( g = 1 \) and \( g' = 1 \). 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, \( g' = 0 \), as shown in Fig. 9(b). As \( g' \) is the NAND gate’s output, all its inputs have logic 1. This uniquely determines all \( r \)-bit key \( K_{g'} \), which is the complement of DIP \( X \), \( K_{g'} = X \), and identical to \( K_g \). SAT attack completes on the 2nd iteration since key \( K_{g'} \) is already resolved. Therefore, the constraint on \( K_g \) helps SAT attack finish within one IO pair.

b) Key constraint on \( K_{g'} \): If the adversary decides to set key \( K_{g'} \) constant instead, he/she will not get the same efficiency for key derivation as in fixing \( K_g \). Suppose we constrain \( K_{g'} = \overline{K_g} \) and \( K_{g'} = \overline{K_g} \) on the SAT attack.
\[ k_r, k_{r+1}, \ldots, k_{2r-1} \] 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 \( g \) 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 the 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 \( K_g \), the adversary removes only one incorrect key and the overall SAT attack complexity remains exponential.

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 \( K_r \) leads to linear complexity in solving \( K_g \) or \( K_r \), 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 Fig. 10. 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 Fig. 10(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 \( K_g \) [Fig. 10(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 \( g = 0 \) 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 Fig. 10(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, e.g., a logic cone (LC) of interest, as shown in Fig. 11. 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 [77], 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

A. Achieving Higher Time Complexity Against SAT Solvers

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
TABLE IV
ANATOMY OF SAT ATTACK TIME ON C6288_N6288

| | | CPU time (s) | UNSAT (%) |
|---|---|---|---|
| | | 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 | 86.40492 | 99.872 |
| 4 | 4 | 88.804 | 0.11963 | 0.02991 | 88.68458 | 99.865 |
| 5 | 5 | 79.614 | 0.12705 | 0.03176 | 79.48717 | 99.840 |
| 6 | 6 | 62.048 | 0.11630 | 0.02908 | 61.93133 | 99.812 |
| 7 | 7 | 88.995 | 0.11822 | 0.02955 | 88.79706 | 99.865 |
| 8 | 8 | 66.762 | 0.11330 | 0.02832 | 66.64487 | 99.830 |
| 9 | 9 | 78.434 | 0.12385 | 0.02477 | 78.31049 | 99.842 |
| 10 | 10 | 62.018 | 0.14788 | 0.02113 | 61.87094 | 99.761 |
| 11 | 11 | 72.615 | 0.15925 | 0.01991 | 72.53534 | 99.780 |
| 12 | 12 | 66.560 | 0.19532 | 0.03255 | 66.36468 | 99.706 |
| 13 | 13 | 74.612 | 0.22130 | 0.02459 | 74.39026 | 99.703 |
| 14 | 14 | 78.492 | 0.18760 | 0.01845 | 78.34455 | 99.811 |
| 15 | 15 | 77.133 | 0.17205 | 0.01721 | 76.96051 | 99.775 |
| 16 | 16 | 83.077 | 0.23765 | 0.03161 | 82.83926 | 99.713 |
| 17 | 17 | 85.083 | 0.74014 | 0.51856 | 79.37841 | 93.295 |
| 18 | 18 | 72.317 | 0.30082 | 0.02006 | 72.01650 | 99.584 |
| 19 | 19 | 89.654 | 0.34619 | 0.02308 | 89.30831 | 99.613 |
| 20 | 20 | 92.588 | 0.32586 | 0.02328 | 92.26268 | 99.645 |
| 21 | 21 | 67.431 | 0.42529 | 0.03017 | 66.97835 | 99.325 |
| 22 | 22 | 80.299 | 0.26642 | 0.02220 | 80.03259 | 99.668 |
| 23 | 23 | 88.226 | 0.43014 | 0.03254 | 87.83904 | 99.275 |
| 24 | 24 | 76.825 | 0.42104 | 0.03024 | 76.40369 | 99.481 |
| 25 | 25 | 88.295 | 2.44402 | 0.12420 | 85.81125 | 97.186 |
| 26 | 26 | 73.065 | 0.45945 | 0.03510 | 72.21507 | 98.837 |
| 27 | 27 | 86.737 | 14.53748 | 0.50129 | 72.19920 | 83.239 |
| 28 | 28 | 149.097 | 13.34636 | 0.49431 | 135.7502 | 91.048 |
| 29 | 29 | 113.466 | 18.31241 | 0.45646 | 112.154 | 98.380 |
| 30 | 30 | 84.404 | 1.66177 | 0.16668 | 83.23738 | 92.693 |
| 31 | 31 | 118.844 | 57.14645 | 1.26992 | 113.61698 | 95.193 |

Fig. 12. SAT attack key evaluation of circuit with an AND gate at the output. Oracle response of (a) logic 1 and (b) logic 0.
number of incorrect keys (or finding a DIP). The goal is to keep the complexity in the order of $O(2^{K_1 + K_2})$ [Fig. 12(b)], not max($O(2^{K_1}), O(2^{K_2})$), equivalent to a logic 0 output at the AND gate in Fig. 12(a). We envision, with controllability analysis, nodes with different output probabilities for logic 0 and 1 under different gate types could be a good indicator for the adaptive key insertion strategy.

C. Extension of the Proposed Complexity Analysis to the SMT Attacks

Our complexity analysis can be extended to the Satisfiability modulo theory (SMT) attacks proposed in [104] due to the similarities between SMT and SAT. SMTs, which consider the satisfiability of formulas under nonbinary variables, offer more flexibility in the input space than the binary space for SAT. SMT attacks expand the capability of the SAT attack to target nonfunctional-based attacks such as delay and timing-based logic locking [60]. Azar et al. [104] proposed four approaches: 1) reduced SAT attack; 2) eager SMT attack; 3) lazy SMT attack; and 4) accelerated lazy SMT attack. Our future work will explore and extend the proposed complexity analysis approach to these SMT attacks.

VII. CONCLUSION

In this article, we provide a new perspective to analyze the efficiency of the SAT attack based on the CNF clause updates inside the SAT solver. In each iteration, SAT attack records the interdependencies between key bits from a DIP and its output response. Any locked circuit with multiple logic cones facilitates 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 nonmonotonically 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.

REFERENCES

[1] W. Shih, "Intell’s S88 Billion European Expansion Is Part of a New Phase in the Globalization of the Semiconductor Industry," Forbes, Jersey City, NJ, USA, 2022.
[2] J. A. Roy, F. Koushanfar, and L. L. Markov, “EPIC: Ending piracy of integrated circuits,” in Proc. Conf. Design Autom. Test Europe, 2008, pp. 1069–1074.
[3] Y. Alkabani and F. Koushanfar, “Active hardware metering for intellectual property protection and security,” in Proc. USENIX Security Symp., 2007, pp. 291–306.
[4] R. S. Chakraborty and S. Bhunia, “Hardware protection and authentication through netlist level obfuscation,” in Proc. IEEE/ACM Int. Conf. Compu.-Aided Design, 2008, pp. 674–677.
[5] Y. Alkabani, F. Koushanfar, and M. Potkonjak, “Remote activation of ICs for piracy prevention and digital right management,” in Proc. IEEE/ACM Int. Conf. Compu.-Aided Design, 2007, pp. 674–677.
[6] J. Huang and J. Lach, “IC activation and user authentication for security-sensitive systems,” in Proc. IEEE Int. Workshop Hardw. Orient. Security Trust, 2008, pp. 76–80.
[7] A. Baungarten, A. Tyagi, and J. Zambreno, “Preventing IC piracy using reconfigurable logic barriers,” IEEE Des. Test Comput., vol. 27, no. 1, pp. 66–75, Jan./Feb. 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,” Trans. Design Autom. Electron. Syst., vol. 21, no. 4, pp. 63, 2016.
[9] E. Capello, U. Meyer-Baese, A. Garcia, L. Parrilla, and A. Lloris, “IFP@HDL: Efficient intellectual property protection scheme for IP cores,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 15, no. 5, pp. 578–591, May 2007.
[10] M. Tehranipoor and C. Wang, Introduction to Hardware Security and Trust. New York, NY, USA: Springer, 2011.
[11] M. Tehranipoor, U. Guin, and D. Forte, Counterfeit Integrated Circuits: Detection and Avoidance. Cham, Switzerland: Springer Int., 2015.
[12] S. Bhunia and M. Tehranipoor, Hardware Security: A Hands-On Learning Approach. New York, NY, USA: Morgan Kaufmann, 2018.
[13] R. Torrance and D. James, “The state-of-the-art in IC reverse engineering,” in Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst., 2009, pp. 363–381.
[14] R. Torrance and D. James, “Reverse engineering in the semiconductor industry,” in Proc. IEEE Custom ICs Conf., 2007, pp. 429–436.
[15] R. Torrance and D. James, “The state-of-the-art in semiconductor reverse engineering,” in Proc. Design Autom. Conf., 2011, pp. 333–338.
[16] S. E. Quadir et al., “A survey on chip to system reverse engineering,” ACM J. Emerg. Technol. Comput. Syst., vol. 13, no. 1, pp. 1–34, 2016.
[17] E. Charbon, “Hierarchical watermarking in IC design,” in Proc. IEEE Custom Integr. Circuits Conf., 1998, pp. 295–298.
[18] A. B. Kahng et al., “Constraint-based watermarking techniques for design IP protection,” IEEE Trans. Comput.-Aided Design Integ. Circuits Syst., vol. 20, no. 10, pp. 1236–1252, Oct. 2021.
[19] G. Qu and M. Potkonjak, Intellectual Property Protection in VLSI Designs: Theory and Practice. New York, NY, USA: Springer, 2007.
[20] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, “Security analysis of logic obfuscation,” in Proc. Annu. Design Autom. Conf., 2012, pp. 83–89.
[21] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, “Fault analysis-based logic encryption,” IEEE Trans. Comput., vol. 64, no. 2, pp. 410–424, Feb. 2013.
[22] P. Subramanian, S. Ray, and S. Malik, “Evaluating the security of logic encryption algorithms,” in Proc. IEEE Int. Symp. Hardw. Orient. Security Trust (HOST), 2015, pp. 137–143.
[23] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu, “SARLock: SAT attack resistant logic locking,” in Proc. IEEE Int. Symp. Hardw. Orient. Security Trust (HOST), 2016, pp. 236–241.
[24] Y. Xie and A. Srivastava, “Anti-SAT: Mitigating SAT attack on logic locking,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 38, no. 2, pp. 199–207, Feb. 2019.
[25] K. Shamsi, T. Meade, M. Li, D. Z. Pan, and Y. Jin, “On the approxima
tion resiliency of logic locking and IC camouflaging schemes,” IEEE Trans. Inf. Forensics Security, vol. 14, no. 2, pp. 347–359, Feb. 2018.
[26] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu, “TTLock: Tenacious and traceless logic locking,” in Proc. IEEE Int. Symp. Hardw. Orient. Security Trust (HOST), 2017, p. 166.
[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 Proc. ACM SIGSAC Conf. Comput. Commun. Security, 2017, pp. 1601–1618.
[28] A. Sengupta, M. Nabeel, N. Limaye, M. Ashraf, and O. Sinanoglu, “Truly stripping functionality for logic locking: A fault-based perspective,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 39, no. 12, pp. 4439–4452, Dec. 2020.
[29] B. Shakya, X. Xu, M. Tehranipoor, and D. Forte, “CAS-lock: A security-correctability trade-off resilient logic locking scheme,” in Proc. IFIP Int. Conf. Trans. Cryptograph. Hardw. Embedded Syst., 2020, pp. 175–202.
[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. Great Lakes Symp. VLSI, 2017, pp. 351–356.
[31] J. Zhou and X. Zhang, “Generalized SAT-attack-resistant logic locking,” IEEE Trans. Inf. Forensics Security, vol. 16, pp. 2581–2592, 2021.
[32] Y. Liu, M. Zueck, Y. Xie, A. Charakhotry, and A. Srivastava, “Strong anti-SAT: Secure and effective logic locking,” in Proc. 21st Int. Symp. Qual. Electron. Design (ISQED), 2020, pp. 199–205.
ZHONG AND GUIN: COMPLEXITY ANALYSIS OF THE SAT ATTACK ON LOGIC LOCKING

[33] H. Zhou, “A humble theory and application for logic encryption,” in Proc. Cryptol. ePrint Arch., 2017, p. 19.

[34] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, “Cyclic obfuscation for creating SAT-unresolvable circuits,” in Proc. Great Lakes Symp. VLSI, 2017, pp. 173–178.

[35] S. Roshanisefat, H. M. Kamali, and A. Sasan, “SRLock: SAT-resistant cyclic logic locking for protecting the hardware,” in Proc. Great Lakes Symp. VLSI, 2018, pp. 153–158.

[36] A. Rezaei, Y. Li, Y. Shen, S. Kong, and H. Zhou, “CycSAT-unresolvable cyclic logic encryption using unreachable states,” in Proc. 24th Asia South Pac. Design Autom. Conf., 2019, pp. 358–363.

[37] 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 Trans. Very Large Scale Integr. (VLSI) Syst., vol. 28, no. 4, pp. 954–967, Apr. 2020.

[38] A. Rezaei, Y. Shen, S. Kong, J. Gu, and H. Zhou, “Cyclic locking and memristor-based obfuscation against CycSAT and inside foundry attacks,” in Proc. Design Autom. Test Europe Conf. Exhibit. (DATE), 2018, pp. 85–90.

[39] 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 Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 41, no. 1, pp. 29–34, Jan. 2022.

[40] 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 Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI), 2018, pp. 405–410.

[41] 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 Proc. 56th Annu. Design Autom. Conf., 2019, pp. 1–6.

[42] G. Kolhe et al., “Security and complexity analysis of LUT-based design obfuscation: From blueprint to reality,” in Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD), 2019, pp. 1–8.

[43] S. D. Chowdhury, G. Zhang, Y. Hu, and P. Nuzzo, “Enhancing SAT-attack resiliency and cost-effectiveness of reconfigurable-bitstream-based circuit obfuscation,” in Proc. Int. Symp. Circuits Syst. (ISCAS), 2021, pp. 1–5.

[44] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, “Cross-lock: Dense layout-level interconnect locking using cross-bar architectures,” in Proc. Great Lakes Symp. VLSI, 2018, pp. 147–152.

[45] S. Patnaik, M. Ashraf, O. Sinanoglu, and J. Knechtel, “Obfuscating the interconnects: Low-cost and resilient full-chip layout camouflaging,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 39, no. 12, pp. 4486–4481, Dec. 2020.

[46] J. Sweeney, M. J. Heule, and L. Pileggi, “Modeling techniques for logic locking,” in Proc. Int. Conf. Comput.-Aided Design (ICCAD), 2020, pp. 1–9.

[47] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, “InterLock: An intercorrelated logic and routing locking,” in Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD), 2020, pp. 1–9.

[48] D. Zhang, M. He, X. Wang, and M. Tehranipoor, “Dynamically obfuscated scan for protecting IPs against scan-based attacks throughout supply chain,” in Proc. IEEE 35th VLSI Test Symp. (VTS), 2017, pp. 1–6.

[49] R. Karmakar, S. Chatopadhyay, and R. Kapur, “Encrypt flip-flop: A novel logic encryption technique for sequential circuits,” 2018, arXiv:1801.04961.

[50] R. Karmakar, H. Kumar, and S. Chatopadhyay, “Efficient key-gate placement and dynamic scan obfuscation towards robust logic encryption,” IEEE Trans. Emerg. Topics Comput., vol. 9, no. 4, pp. 2109–2124, Oct.–Dec. 2019.

[51] S. Potturi, A. Ayu, and A. Kumar, “SeqL: Secure scan-locking for IP protection,” 2020, arXiv:2005.13662.

[52] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, “Scramble: The state, connectivity and routing augmentation model for building logic encryption,” in Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI), 2020, pp. 153–159.

[53] M. S. Rahman et al., “Security assessment of dynamically obfuscated scan chain against oracle-guided attacks,” ACM Trans. Design Autom. Electron. Syst., vol. 26, no. 4, pp. 1–27, 2021.

[54] F. Koushanfar, “Active hardware metering by finite state machine obfuscation,” in Hardware Protection Through Obfuscation. Cham, Switzerland: Springer, 2017, pp. 161–187. [Online]. Available: https://link.springer.com/book/10.1007/978-3-319-49019-9.

[55] J. Dofe and Q. Yu, “Novel dynamic state-delegation method for gate-level design obfuscation,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 37, no. 2, pp. 273–285, Feb. 2018.
