Complexity Analysis of the SAT Attack on Logic Locking

Yadi Zhong and Ujjwal Guin

Dept. of Electrical and Computer Engineering, Auburn University, City, Country,
yadi@auburn.edu,ujjwal.guin@auburn.edu

Abstract. Due to the adoption of the horizontal business model with the globalization of semiconductor manufacturing, the overproduction of integrated circuits (ICs) and the piracy of intellectual properties (IPs) have become a significant threat to the semiconductor supply chain. Logic locking has emerged as a primary design-for-security measure to counter these threats. In logic locking, ICs become fully functional after fabrication only when unlocked with the correct key. However, Boolean satisfiability-based attacks have rendered most locking schemes ineffective. This gives rise to the 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 the SAT solver. First, we show that the attack learns a new relation between key bits upon every distinguishing pattern. After each iteration, these additional clauses appended to the solver could significantly decrease the key search complexity. Second, we demonstrate that the SAT attack can break the locking scheme within the linear complexity of key size. The deviation away from linear search can be explained by the oracle's output and different logic gate types. This helps to answer how different distinguishing input eliminates fewer or more incorrect keys. Moreover, we show how key constraints on point functions affect the complexity of SAT attack. The proper key constraint on AntiSAT locking can effectively reduce the SAT attack complexity to constant 1. The same constraint minimizes the complexity of breaking CAS-Lock down to the linear range. Our analysis provides fresh perspectives on the capabilities of SAT attack, and we offer new directions to achieve SAT resiliency.

Keywords: Logic locking, Boolean satisfiability (SAT), distinguishing input patterns, conjunctive normal form (CNF), IP piracy

1 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 [Shi22]. 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

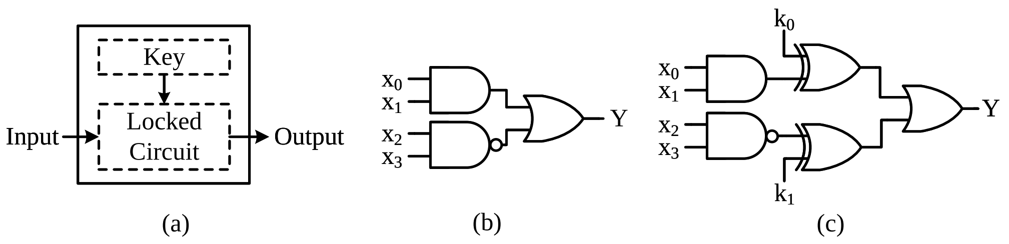

Figure 1: Overview of logic locking, where a circuit functions correctly when the right key is programmed in a tamper-proof memory. (a) Abstract view of logic locking. (b) Original circuit. (c) XOR-based locking, where XOR gates are added to lock the functionality.

chips on their own. As a result, the trusted foundry model is no longer assumed for the production of ICs, where the trustworthiness of microelectronic products is often questioned.

The underlying hardware in various information systems that were once trusted can no longer be so due to the outsourced IC design and fabrication. 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 [RKM08, AK07, CB08, AKP07, HL08, BTZ10, GSFT16], and (*ii*) piracy of IPs, where an entity in the supply chain can use, modify and/or sell functional IPs illegally [CMBG+07, TW11, TGF15, BT18]. An untrusted foundry can have access to all the mask information constructed from the GDSII or OASIS files and then reconstruct all the layers and the complete netlist with advanced tools [TJ09]. 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 [TJ07, TJ11]. Unfortunately, the same RE can be exploited by an adversary to reconstruct the gate-level netlist from a chip [QCF+16].

One of the best ways to prevent an adversary from copying 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 [Cha98, KLMS+01, QP07] 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 [RKM08, BTZ10, RPSK12], and hardware watermarking [Cha98, KLMS+01, QP07] was broken by Boolean Satisfiability (SAT) attack [SRM15]. Subramanyan et al. [SRM15] applies SAT solver for deriving the correct key value with distinguishing input patterns. Subsequently, novel logic locking techniques have been proposed to drastically increase the required number of input patterns in the SAT attack to break the key. These includes Anti-SAT [XS18], SARLock [YMRS16], SFLL [YSN+17], and CAS-Lock [SXTF20]. Concurrently, multiple attacks [YMSR17, SLM+17, XSTF17, SZ17, SS19, ZCZG19, JRG20, LPS21, SLS21, APK+21] against these logic locking arise.

This paper presents two novel aspects to analyze SAT attack complexity. 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 distinguishing input patterns (DIP) 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. Principally, the expected goal for any logic locking technique is to achieve an exponential search complexity for the secret key size so that an adversary cannot determine the correct key value within the polynomial attack complexity. However, our analysis points to the linear growth of the required patterns or iterations rather than the desired exponential increase with keys. Using an example, 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 logic cones. This motivates us to analyze the effectiveness of the SAT attack under the single logic cone of one output. Note that a logic cone can be described as a directed graph where the inputs nodes and gates are pointing toward the sole output. Second, one interesting observation is that the complexity of the SAT attack often reduces with the increase of the key size. We provide detailed explanations of why it takes fewer iterations to find the correct key when we lock the circuit with a larger key size. Finally, we analyze the SAT attack complexity for a circuit that is locked using point functions [XS18, SXTF20].

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 presents the step-by-step analysis of incorrect key elimination for each DIP using examples. 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 search 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 the 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 response of the oracle after applying a DIP plays an important role in eliminating a large number of incorrect keys. A response 0 at the OR gate effectively splits the logic cone into two subcones, where the keys inside the subcone are dependent, but independent from the keys of the other subcone. Such an IO pair reduces the attack complexity exponentially when keys are evenly distributed among the two subcones. Similarly, logic 1 at the AND output has a similar effect in shrinking the search space.
- *Insufficiency in the construction of point functions:* The point function insertion in logic locking has shown strictly exponential iterations in complexity against SAT attack. Unfortunately, they can be broken while properly constraining the SAT tool. We show why SAT attack needs only 1 IO pair to derive the key value for AntiSAT if the keys in the block g are held constant. We also demonstrate why CAS-Lock with fixed key K_g reduces the exponential search complexity to linear with our contributions to the analysis of SAT attack and its complexity.

The rest of the paper is organized as follows. We briefly introduce the background of SAT attack and various locking methods in Section 2. The inter-dependency between keys learned by the SAT solver after each DIP is extensively explored in Section 3. The SAT attack complexity is further analyzed and explained in Section 4. Analysis of the point functions is shown in Section 5. The future directions are described in Section 6. Finally, we conclude the paper in Section 7.

2 Background

Logic locking emerges as a promising solution against IC overproduction and IP piracy. However, the initial efforts [RKM08, BTZ10, RPSK12, RZZ+15] were challenged by SAT attack [SRM15], a tool that breaks all these locking techniques efficiently. Subsequent logic locking schemes [XS18, YMRS16, SXTF20, YSS+17, YSN+17, SNL+20] work towards thwarting this attack. In this section, we revisit the SAT attack algorithm and provide an overview of the various locking efforts toward SAT resiliency.

2.1 SAT attack on Logic Locking

The entire series of attacks and the solutions thereafter originated from the SAT attack [SRM15]. Subramanyan et al. [SRM15] exploits the idea of combinational equivalence checking with miter circuit and Boolean Satisfiability [KPKG02] to attack logic locking schemes. It successfully decrypts the secret key from various logic locking techniques [RKM08, BTZ10, RPSK12, RZZ+15] 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.

Algorithm 1: SAT attack on logic locking [SRM15].

```
   Input  : Unlocked circuit, oracle ($C_O(X,Y)$) and locked circuit ($C(X,K,Y)$)
   Output : Correct Key ($K_c$)
1  $i \leftarrow 0$ ;
2  $F \leftarrow []$;
3  while (true) do
4      $i \leftarrow i + 1$ ;
5      $[X_i, K_i, r] = \texttt{sat}[F \wedge (Y_{A_i} \neq Y_{B_i})]$;
6      if ($r == \texttt{false}$) then
7          break;
8      end
9      $Y_i = \texttt{sim\_eval}(X_i)$;
10     $F \leftarrow F \wedge C(X_i, K, Y_i)$;
11 end
12 $K_c \leftarrow K_i$;
13 return $K_c$ ;
```

The SAT attack derives the correct key value through the following steps:

- **Finding the distinguishing input pattern (DIP) from the miter circuit:** It first constructs two locked circuit instances (e.g., $C(X, K_A, Y_A)$ and $C(X, K_B, Y_B)$) to form a miter construction. Both circuits share the same input except 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. If the UNSAT conclusion is generated, the differential output cannot be observed, r is assigned to `false`, and X_i is empty.

- **Deriving the correct key:** Upon obtaining a DIP X_i, the SAT attack acquires the actual output Y_i from oracle simulation, $C_O(X_i, Y_i)$, Line 9. The input X_i and the correct output Y_i are used in updating the solver assumptions F, Line 10. The constraints in F help narrow down the valid keyspace until it is left with only the correct key(s).

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 assumptions F, the Boolean variable r becomes `false` and it indicates that differential output does not exist. 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 as the correct key since they all give the same output response. This holds for a few locking designs [XS18, SXTF20] and certain locking scenarios, *e.g.,* chained XOR key gates. 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 [SRM15] by disabling both preload vectors (all zeros and all ones) so that the number of distinguishing patterns $|P|$ used to derive the correct key by the SAT attack is one less than the number of iterations T, $|P| = T - 1$. This is because the last iteration does not produce any DIP.

2.2 SAT Resistant Logic Locking Techniques and Attacks

As SAT attack [SRM15] successfully breaks various logic locking techniques [RKM08, BTZ10, RPSK12], it propels the research community to explore new locking schemes [YMRS16, XS18, SXTF20, YSS$^+$17, YSN$^+$17, SNL$^+$20] that utilize point functions for achieving the minimal output corruptibility. SARLock [YMRS16] perturbs only the output of one input pattern for each incorrect key. AntiSAT [XS18] and CAS-Lock [SXTF20] configure the point function with two complementary blocks g and $\overline{g}$. SFLL [YSS$^+$17, YSN$^+$17] flips the output for certain input patterns, where the correct key flip back the upset output and restores the original functionality. Although these techniques guarantee exponential iterations in SAT attack, various attacks [YMSR17, SLM$^+$17, XSTF17, SZ17, SS19, ZCZG19, JRG20, LPS21, SLS21, APK$^+$21] have been proposed to exploit the designs' vulnerabilities, *e.g.,* from structural and functional perspectives, and restore the original circuit. These attacks can be divided into several categories, SAT-based [SLM$^+$17, XSTF17, SLS21, SZ17], ATPG-based [JRG20, LPS21], structural-based [YMSR17, SS20, ZCZG19, APK$^+$21]. For example, SARLock [YMRS16] can be broken by AppSAT [SLM$^+$17] and Bypass attack [XSTF17] while AntiSAT [XS18] is attacked by signal probability skew (SPS) [YMSR17], AppSAT [SLM$^+$17], and Bypass attack [XSTF17].

3 SAT Attack Analysis: Pruning of Incorrect Key with CNF Update

Even though SAT attack has been proposed, the reasons for its efficiency and effectiveness in breaking various locking schemes and deriving the correct keys have not been thoroughly analyzed. In this section, we investigate the CNF clauses stored in the SAT solver and how it gets updated in every iteration with the distinguishing input pattern and 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. Generally, a circuit can have multiple outputs $y_0, y_1, ..., y_{t-1}$. We define a logic cone based on outputs', where all the inputs and wires (gate's output) which could reach an output y_i belong to the logic cone of y_i. An increased number of primary outputs usually leads to the propagation of multiple key values reaching different output bits simultaneously. We begin our analysis with an example circuit 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 an increased number of logic cones. This offers new insights on the property of SAT attack.

3.1 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 2.1, 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.

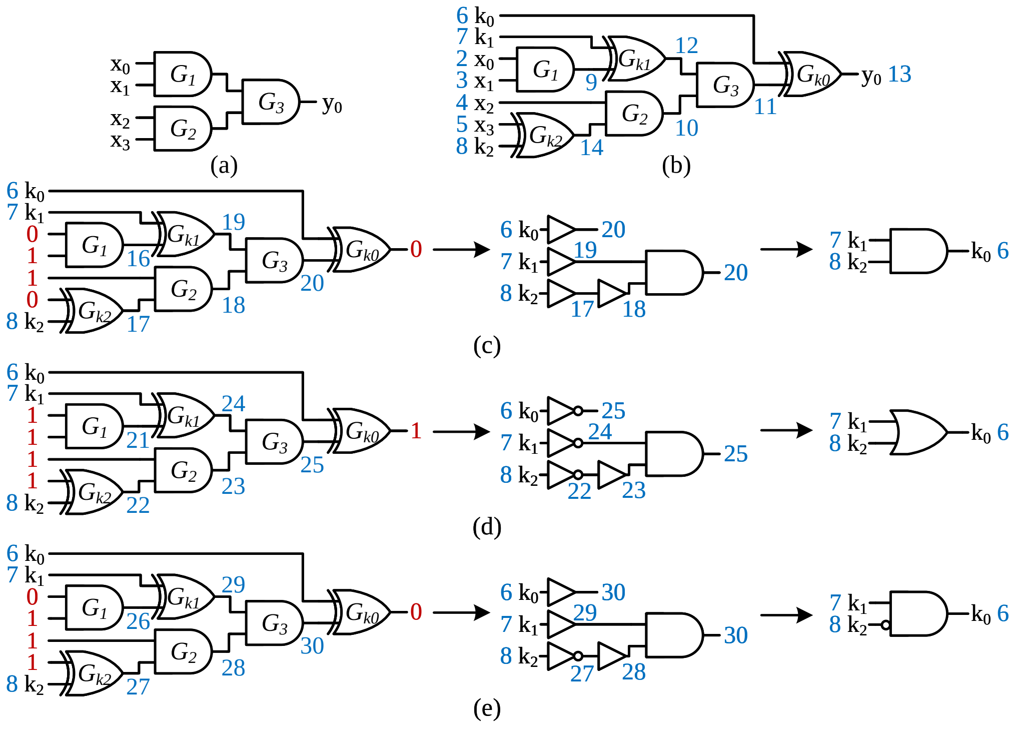

Figure 2: Step-by-step SAT attack analysis with different DIPs. Each DIP and the response observed from the oracle create a relationship among different key bits. (a) Original circuit. (b) Locked circuit. CNF update with (c) 1st IO pair $P_1 = \{X_1; Y_1\} = \{x_0, ..., x_3; y_0\} = \{0110; 0\}$, (d) 2nd IO pair $P_2 = \{1111; 1\}$, and (e) 3rd IO pair $P_3 = \{0111; 0\}$.

Let us consider a simple circuit with 4 inputs $x_0, x_1, ..., x_3$ and 1 output y_0 of Figure 2(a). The locked version with 3 key bits, k_0, k_1, k_2, are shown in Figure 2(b) and each logic gate is mapped to the corresponding CNF clauses. Upon finding of valid assignment to the miter circuit in the first iteration, the DIP X is extracted, $\{X_1\} = \{x_0, x_1, ..., x_3\} = \{0110\}$.

The correct output response $Y_1 = 0$ is gathered from simulating the oracle with the input patter X. This IO pair $P_0 = \{X_1; Y_1\} = \{x_0, x_1, ..., x_3; y_0\} = \{0110; 0\}$ is then recorded. We show in detail how the CNF for the locked circuit is updated so that the SAT attack prunes out half of the search space (4 incorrect keys) with this IO pair. To avoid altering the locked circuit's original CNF, the internal nodes are labeled with new variables 17-20, as shown in Figure 2(c), and its equivalent CNF for $C(X_1, K, Y_1)$ (abbreviated as C_1) is:

$$\begin{aligned}
C_1 \quad = \quad & \overbrace{(\overline{2}\vee\overline{3}\vee 16)\wedge(2\vee\overline{16})\wedge(3\vee\overline{16})}^{\text{AND gate } G_1}\wedge\overbrace{(\overline{4}\vee\overline{17}\vee 18)\wedge(4\vee\overline{18})\wedge(17\vee\overline{18})}^{\text{AND gate } G_2}\wedge \\
& \overbrace{(\overline{18}\vee\overline{19}\vee 20)\wedge(18\vee\overline{20})\wedge(19\vee\overline{20})}^{\text{AND gate } G_3}\wedge \\
& \overbrace{(\overline{20}\vee\overline{6}\vee\overline{13})\wedge\ (\overline{20}\vee 6\vee 13)\wedge(20\vee 6\vee\overline{13})\wedge(20\vee\overline{6}\vee 13)}^{\text{XOR gate } G_{k0}}\wedge \\
& \overbrace{(\overline{16}\vee\overline{7}\vee\overline{19})\wedge(\overline{16}\vee 7\vee 19)\wedge(16\vee 7\vee\overline{19})\wedge(16\vee\overline{7}\vee 19)}^{\text{XOR gate } G_{k1}}\wedge \\
& \overbrace{(\overline{17}\vee 5\vee 8)\wedge(\overline{17}\vee\overline{5}\vee\overline{8})\wedge(17\vee 5\vee\overline{8})\wedge(17\vee\overline{5}\vee 8)}^{\text{XOR gate } G_{k2}}
\end{aligned}$$

With IO pair P_1, we know the logic values for input/output, namely literals $2 = 0, 3 = 1, 4 = 1, 5 = 0, 13 = 0$, and the C_1 is updated as:

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

This means node 16 has to be logic 0, and F is adjusted to:

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

This equation indicates that variables 7 and 19 have the same logic value, so are 6 and 20, 8 and 17, 17 and 18. The circuit representation of the above equation is described in Figure 2(d), which is a function of k_0, k_1, k_2, as described in $C(X_i, K, Y_i)$. These are the clauses appended in the formula F. Equivalently, the SAT attack learned from IO pair P_1 is essentially the relation between k_0, k_1, k_2, where $k_0 = k_1 \cdot k_2$, as shown in Figure 2(e). With this constraint, the possible key space shrank into half.

In the second iteration, SAT solution of the miter, together with oracle simulation, returns the IO observation $P_2 = \{X_2; Y_2\} = \{1111; 1\}$ as the distinguishing IO pair, as drawn in Figure 2(d). Similar to the derivation we performed for the first iteration, the circuit representation of the added CNF clauses is shown in Figure 2(g), which again is a function between the 3 key bits, as simplified in Figure 2(h). This constraint shrinks the remaining keyspace in half, which, equivalently, removes another two incorrect key combinations. Likewise, the tool returns another IO pair $P_3 = \{0111; 0\}$ on the third iteration, and the appended CNF with its equivalent relation to the key is shown in Figure 2(e). The relations obtained from these three patterns, P_1, P_2, P_3, Figure 2(c,d,e), uniquely determines the 3 key bits $\{k_0, k_1, k_2\} = \{000\}$. This is the reason that, on the fourth iteration, no more distinguishing input can be found for the miter circuit, where r is `false`, and the SAT attack is complete.

To summarize, the IO pair provides the attack with additional information on the unknown key bits, as the locked circuit equation $C(X_i, K, Y_i)$ with a fixed IO offers an equation for the unknown keys. In each iteration, a new equation for the key bits is obtained from the corresponding IO pair, which is independent of the findings derived from the previous rounds. SAT attack derives the correct key value once the accumulated system of equations can uniquely determine the unknown key bits.

3.2 SAT Attack against Multiple Overlapping Logic Cones

It is common for a circuit to have multiple primary outputs. In other words, we know the circuit has multiple logic cones. These outputs are more likely to observe incorrect key responses than a single output. The reason is that the effect of multiple keys can reach several outputs, accelerating the removal of incorrect combinations and facilitating the final decision of the correct key in earlier iterations than the single logic cone where every key has to pass through the same output port. Keys that reside exclusively in the same logic cone may or may not be interdependent with each other when trying to propagate just one of them to the output. This is demonstrated by the example below. The following example shows that SAT attack needs fewer iterations to derive the correct key under multiple intersecting logic cones.

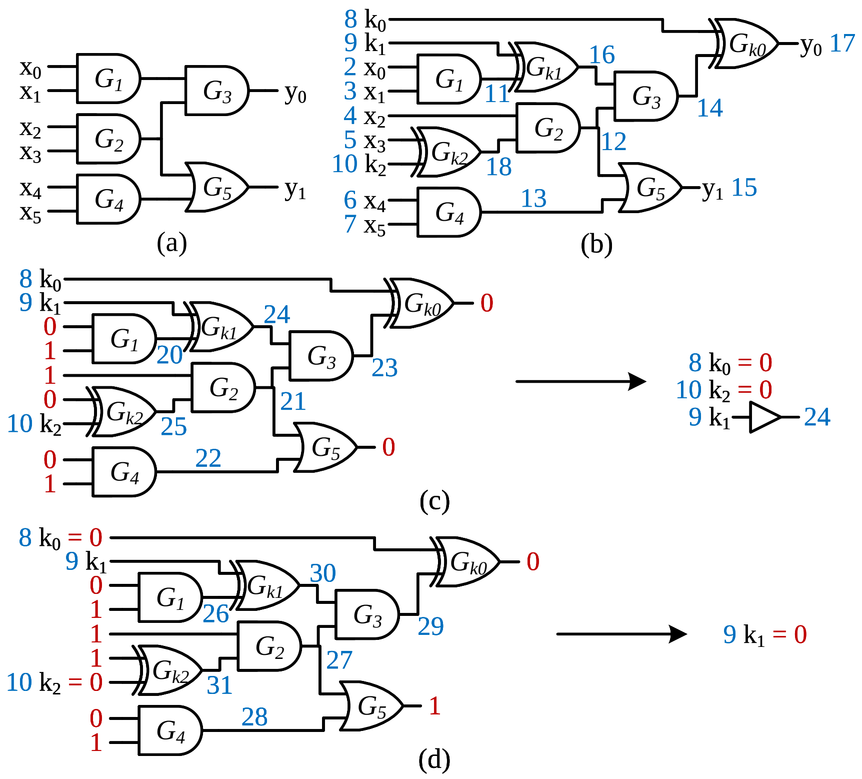

Figure 3: SAT attack on 2 intersecting logic cones. (a) original circuit (b) locked circuit (c) 1st pair $P_1 = \{X_1; Y_1\} = \{x_0, ..., x_5; y_0, y_1\} = \{011001; 00\}$ (d) equivalent relation of k_0, k_1, k_2 with P_1, k_0 and k_2 are solved (e) 2nd IO pair $P_2 = \{011101; 01\}$ (f) k_1 is obtained.

Let us consider a circuit with 2 outputs, y_0 and y_1, as shown in Figure 3(a). The locked circuit, as shown in Figure 3(b), has 3 key bits, k_0, k_1, k_2, with the same locations as in Figure 2(b). It differs from the locked circuit in Figure 2(b) with additional gates G_4, G_5 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 $G_1, G_2, G_3, G_{k0}, G_{k1}$; the other with output y_0, inputs x_2, x_3, x_4, x_5, key k_2 and gates G_2, 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 circuit with a single output y_0 in Figure 2(b). The first pattern $P_1 = \{X_1; Y_1\} = \{x_0, ..., x_5; y_0, y_1\} = \{011001; 00\}$ is shown in Figure 3(c), which is equivalent to the CNF expression of $C(X_1, K, Y_1)$ (abbreviated as C_1) below.

$$\begin{aligned} C_1 \quad = \quad & (\overline{2} \vee \overline{3} \vee 20) \wedge (2 \vee \overline{20}) \wedge (\overline{20} \vee 3) \wedge (\overline{20} \vee \overline{9} \vee \overline{24}) \wedge (\overline{20} \vee 9 \vee 24) \wedge (20 \vee 9 \vee \overline{24}) \wedge \\ & (20 \vee \overline{9} \vee 24) \wedge (\overline{4} \vee \overline{25} \vee 21) \wedge (4 \vee \overline{21}) \wedge (\overline{21} \vee 25) \wedge (\overline{21} \vee \overline{24} \vee 23) \wedge (21 \vee \overline{23}) \wedge \\ & (24 \vee \overline{23}) \wedge (21 \vee 22 \vee \overline{15}) \wedge (\overline{21} \vee 15) \wedge (\overline{22} \vee 15) \wedge (\overline{25} \vee 5 \vee 10) \wedge (\overline{25} \vee \overline{5} \vee \overline{10}) \wedge \\ & (25 \vee 5 \vee \overline{10}) \wedge (25 \vee \overline{5} \vee 10) \wedge (\overline{6} \vee \overline{7} \vee 22) \wedge (6 \vee \overline{22}) \wedge (\overline{22} \vee 7) \wedge (\overline{23} \vee \overline{8} \vee \overline{17}) \wedge \\ & (\overline{23} \vee 8 \vee 17) \wedge (23 \vee 8 \vee \overline{17}) \wedge (23 \vee \overline{8} \vee 17) \end{aligned}$$

IO pair P_1 determines the logic value for inputs and outputs, where $2 = 0, 3 = 1, 4 = 1, 5 = 0, 6 = 0, 7 = 1, 15 = 0, 17 = 0$. The CNF for the locked circuit with P_1 is adjusted analogously to the previous example's (Figure 2) by plugging in the logic value of these known literals. It is straightforward that node 20, the output of AND gate G_1, has logic 0, as its inputs are literals $2 = 0$ (x_0) and $3 = 1$ (x_1). Similarly, both node 21 and 22 are in logic 0, since they are the input of OR gate G_5, whose output is logic 0 ($15 = 0$). As literal 21 is the output of AND gate G_2, its unknown input, node 25 can be uniquely determined to be logic 0, since the other input x_2 (4) has logic 1 with AND output in logic 0. Literal 23, the output of AND gate G_3 must be logic 0 since one of its input, node 21, is 0. Hence, the CNF clauses added to SAT solver after the first iteration is:

$$C_1 \quad = \quad (9 \vee \overline{24}) \wedge (\overline{9} \vee 24) \wedge (\overline{10}) \wedge (\overline{8})$$

With C_1, SAT attack determines both key k_0, that is literal 8, has logic value 0 and k_2 of node 10 is logic 0 with 1 IO pair. Key k_1 is the only unknown key left in this iteration, and it has the same logic value as node 24, $(9 \vee \overline{24}) \wedge (\overline{9} \vee 24)$, which can be represented as a buffer. The findings of k_0, k_1 and k_2 are summarized in Figure 3(d). The IO vector for the second iteration is $P_2 = \{X_2; Y_2\} = \{011101; 01\}$, as shown in Figure 3(e) and it uniquely determines the key bit k_1 as logic 0. At the third round, SAT attack returns UNSAT as no more incorrect key needs to be pruned and the program finishes.

In essence, SAT attack learns new relations between the unknown key bits at each iteration, expressed in CNF clauses. If a key bit can be uniquely determined, its value is appended as a unit clause. The combined relations help SAT solver decrypt all key bits, deriving the correct key. Compared with a circuit with a single output, the circuit with more fanouts could facilitate the breaking of inter-dependency between key bits at earlier rounds. Having multiple outputs (or logic cones) will reduce the number of iterations for the SAT attack, making it easier to derive the correct key when multiple key bits can be observed at the outputs simultaneously. Therefore, in the following discussions, we focus on the analysis of the SAT attack with a single logic cone.

4 SAT Attack Analysis: Search Complexity

Logic locking perturbs the original functionality of the circuit with additional key gates so that, unless the correct key is programmed in the chip, the locked circuit behaves differently from the oracle. The goal of logic locking is to prevent the adversary from deriving the correct key value either under oracle-less attacks or oracle-guided attacks. In this section, we focus on the total iterations required for the SAT attack as the SAT attack complexity. *We provide analysis on how the output response of the oracle under of certain DIPs can trim more incorrect keys than other IO pairs.*

4.1 Linear Trend in Total SAT Attack Iterations

As observed from the original SAT attack [SRM15], MUX-based logic locking [RZZ+13] normally needs fewer IO vectors than its XOR-based counterpart. For LUT-based locking [BTZ10], SAT attack is essentially performing the same procedure as the automatic test pattern generation (ATPG) tool, as the keys inside one LUT can only be observed one at a time. The differential output behavior for the miter is guaranteed when the key bit value, or its complement, reaches the primary output, which is equivalent to having either a stuck-at 0 or stuck-at 1 placed at the same key line and observing the faulty response at the output [BA04]. That is to say, the total iterations required to break LUT-based locking are in the same order of the key size, and could be fewer when multiple logic cones are present in the circuit (see analysis in Section 3.2). Therefore, the complexity of LUT-based locking is linear with respect to key size. We instead select XOR-based locking to show the performance of SAT attack.

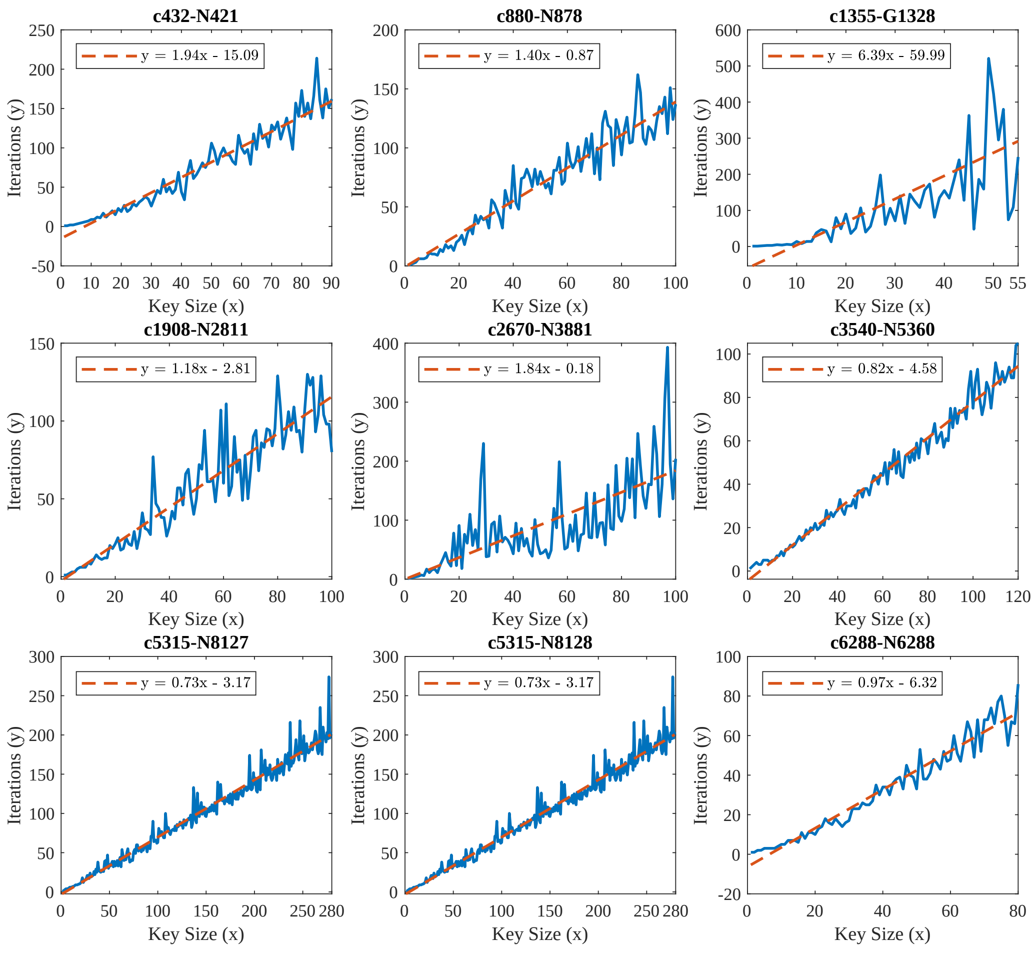

Figure 4: SAT attack Iterations for ISCAS'85 benchmarks.

SAT attack previously analyzed total observations needed to decrypt various locked benchmarks with the same overhead. To quantitatively show the overall trend of the complexity of this attack, we iteratively increase the key size of a circuit so that we could get a comprehensive view of the relations between key size and complexity within each locked circuit. The locking is performed with the insertion of XOR/XNOR key gates before any gate types except XOR/XNOR, inverter (INV), or buffer (BUF) to ensure a unique solution for the correct key. To remove the potential complexity reduction in the presence of multiple logic cones, we extract single cones from the ISCAS'85 benchmarks so that all the locked circuit has a single output, where the response of any incorrect key combinations is observed through the sole output. In particular, we pick the logic cone with the largest gate number, where more keys can be inserted than smaller cones. We exclude benchmark c499 as its largest cone supports only a maximum of 21 keys but includes the multiplier benchmark c6288. All benchmarks are first synthesized with 32nm technology libraries in Synopsys Design Compiler [Syn]. Then, each synthesized circuit is converted to a directed graph with inputs pointing toward the gates' output and, ultimately, the primary output. Logic cones are extracted by reversing all the edge direction [MATa] and performing breadth-search [MATb] from each primary output. The ordered list returned from the breadth-first search has two uses. First, by comparing the number of nodes in the list, the largest cone (or cones if a tie) is determined and extracted.. Second, the breadth-first search traverses all the gates (nodes) within the same layer (the same distance from the output nodes) before exploring the gates at the subsequent layers. Following the order of the same nodes as the breadth-first search, we successively add one more XOR key gate at a time, starting from gates that are closest to the primary output with increasing proximity, while skipping any XOR/XNOR, INV, and BUF. The original cone and its locked designs are all converted to the *bench* format. SAT attack runs through each and every locked cone, and the total iterations are recorded. Figure 4 shows the SAT attack complexity (with respect to total iterations) on 9 benchmark cones with increasing key size. For example, c432-N421 is the logic cone from the c432 benchmark with output N421. Cone c5315-N8127 and c5315-N8128 both contain the same number of gates, but there is a significant overlap between them.

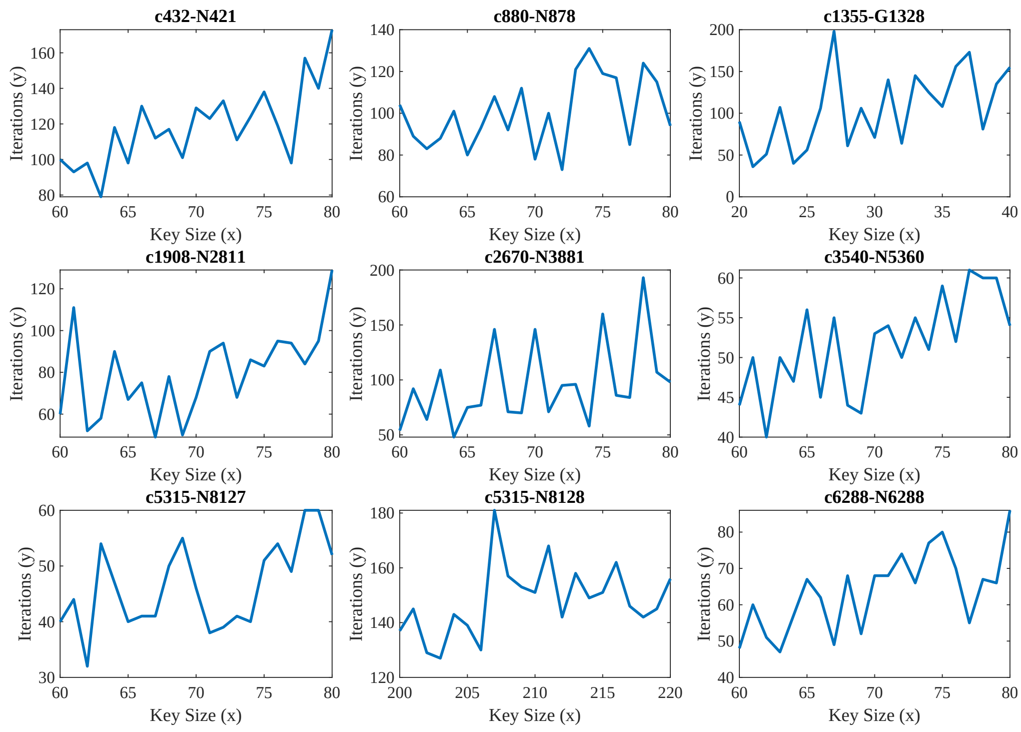

Figure 5: The zoomed-in view of SAT attack Iterations for ISCAS'85 benchmarks.

It is clear that there is an overall linear trend in total iterations with increased key size with single primary output. The best fit line for each cone is shown in a dashed line with its equation recorded in the legend. It indicates that locking with more key gates does not ensure an exponential complexity increase for the SAT attack. Second, all 9 benchmark cones do not have a monotonous increase in complexity when more keys are inserted. The zoomed-in view of the SAT attack complexity for the benchmarks in Figure 4 is shown in Figure 5. It is evident that the addition of key gates does not lead to an increment in SAT attack complexity. The zig-zag behavior in SAT attack complexity is observed in all benchmark cones. The question is, *what causes the SAT attack to have such complexity drops when more keys are present in a locked design?* The next section will provide a qualitative answer to this behavior.

4.2 Exponential Elimination of Keys within an Iteration

From Section 3, we know that, at each round (i^{th}), SAT attack finds DIP (X_i) for the miter circuit, the oracle output (Y_i) and updates CNF with new relations between the keys $C(X_i, K, Y_i)$. As described in Section 4.1, the reduction in SAT attack complexity can happen with a larger key size. Sometimes, one may notice that $C(X_i, K, Y_i)$ clauses added from an IO pair $\{X_i, Y_i\}$ may put more constraints on keys, which is equivalent to eliminating more incorrect keys, than other distinguishing IO vectors found by the SAT attack. We explore how the oracle's output response could explain the non-monotonous trend for the SAT attack complexity. We also demonstrate how SAT attack is able to trim the search space exponentially based on the circuit structure and the IO pair returned. This answers why SAT attack could use fewer IO observations than the key size to derive the correct key.

Two possible locking scenarios can happen for a locked circuit when we categorize it based on its output. One is that the output of the locked cone is directly connected to a key gate while the other does not. We first explore the locked circuit when one of the key gates is directly placed at the primary output, and shown in Figure 6. We show that only one pattern is sufficient to remove half of the keyspace.

Theorem 1. *The first IO pair P_1 removes 2^{n-1} incorrect key combinations for any locked circuit with key gate (XOR) at the primary output.*

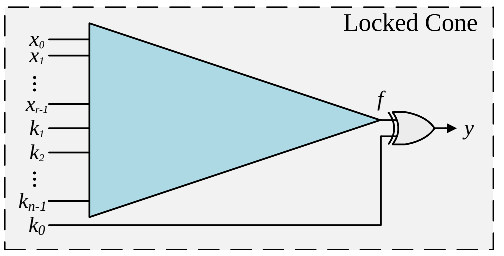

Figure 6: The abstract view of a locked circuit with key gate (XOR with key k_0 and f as its input) at the primary output.

Proof. The first IO pair $P_1 = \{X_1; Y_1\} = \{x_0, ..., x_{r-1}; y\}$ gives a relation between the keys of $C(X_1, K, Y_1)$. Specifically, the CNF for the output XOR gates is updated from $(\overline{k_0} \vee \overline{f} \vee \overline{y}) \wedge (k_0 \vee f \vee \overline{y}) \wedge (k_0 \vee \overline{f} \vee y) \wedge (\overline{k_0} \vee f \vee y)$ to either $(k_0 \vee \overline{f}) \wedge (\overline{k_0} \vee f)$ when $y = 0$ or $(\overline{k_0} \vee \overline{f}) \wedge (k_0 \vee f)$ when $y = 1$. This means that either f and k_0 has the same logic value $f = k_0$ if $y = 0$, or complementary of each other $f = \overline{k_0}$ if $y = 1$. With the known input vector X_1 from P_1, we know that the 1-bit f is essentially a function of the n-1 bit keys $\{k_1, ..., k_{n-1}\}$. It is equivalent to a mapping from (n-1)-bit to 1-bit, $M := \{0,1\}^{n-1} \longrightarrow \{0,1\}$. Depending on the oracle's output value y, either k_0 or its complementary logic value is the output of mapping M. In other words, this mapping M restricts the value of k_0 such that it cannot be both 0 and 1 under every input combination from the (n-1)-bit input, but a definite value. Namely, there are 2^{n-1} possible key combinations for the n-bit key K as key bit k_0 depends on the rest of the (n-1)-bit keys $\{k_1, ..., k_{n-1}\}$. Therefore, the mapping M, derived from IO pair P_1, removes $2^n - 2^{n-1} = 2^{n-1}$ possible key combinations from the search space.

This statement remains true if k_0 is the only key bit in the locked circuit, where $n = 1$. The construction of the miter circuit ensures that the two copies of the locked circuit have the opposite logic value for the key bit k_0, as they form the entire keyspace. The first IO pair is sufficient to derive the correct key and prune out the incorrect one, which is the same as slashing the search space in half. $\square$

This proves that the first pattern trims an exponential number of 2^{n-1} keys from the entire key search space if the key gate is placed at the output no matter the logic value of the IO pair. Thus, the security level of the unknown keys is reduced from n-bit to $(n-1)$-bit as the keyspace shrinks from 2^n down to 2^{n-1} after the first IO pair. As a result, the first IO pair practically removes the security gains of k_0 from the locked circuit. After the first iteration, the complexity of breaking the locked circuit drops to the same order as when the output gate is not locked - that is, the second locking scenario. However, for an IO pair, the reduction in complexity might change when all key gates are placed inside the cone. When no key gates are at the output, we explore how keyspace is pruned and show that this pruning depends on the IO pair. Note that similar or more incorrect keys can be pruned under certain IO pairs, but not for all.

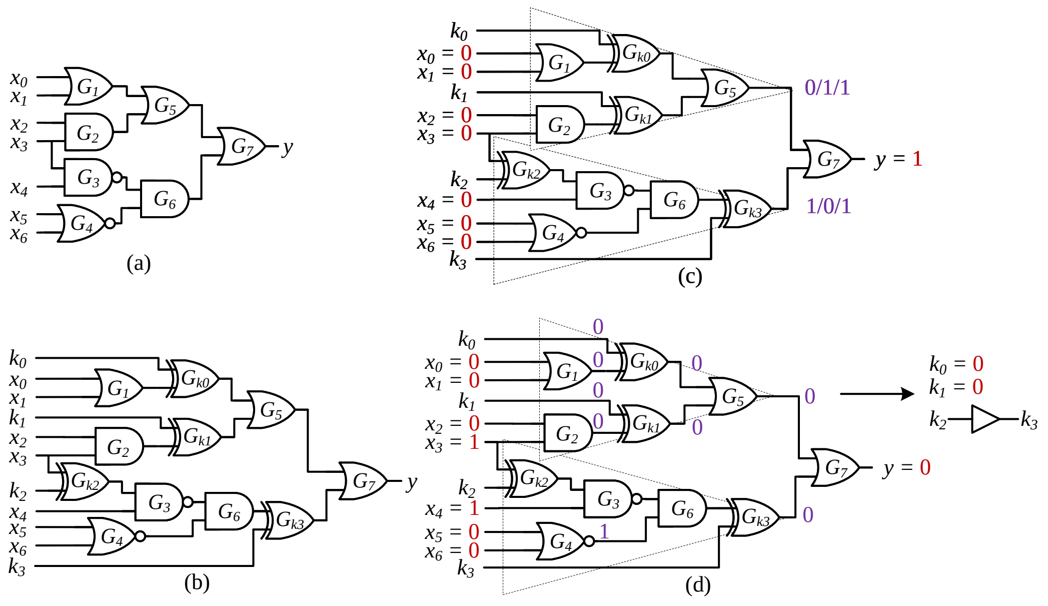

Figure 7: Key elimination. (a) original circuit (b) the locked circuit with 4 XOR key gates (c) the 1st IO pair $P_1 = \{X_1; Y_1\} = \{x_0, ...x_6; y\} = \{0000000;1\}$ from SAT attack (d) the second IO pair $P_2 = \{0001100;0\}$ and equivalent relation of k_0, k_1, k_2, k_3 under P_2 only, where k_0 and k_1 are determined.

Figure 7 shows an example to clarify the two scenarios, where four keys $(k_0 - k_3)$ are added, where the output of the cone results from an OR gate. The 1st 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 it is the output of the OR gate G_7, its two inputs could be any from 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, as shown in Column 2 of Table 1. Any key combinations that cause an output mismatch with the oracle's output are marked with ✗, indicating an incorrect key value; the key value which produces the same output as the oracle is noted with ✓. From the 1st iteration, we observe that a small number of incorrect keys (i.e., $2 << \frac{2^4}{2}$) are removed due to the logic properties of the OR gate, where no unique conclusion can be made regarding its inputs (i.e., 10, 01 or 11) if the output is 1.

Table 1: SAT attack uses 2 patterns to eliminate 15 incorrect keys from the search space. If the output differs from the oracle's, ✗ is placed, else ✓. The correct key is highlighted in red.

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	✓	✗

We now show that it is possible for certain IO pairs that a large number of incorrect keys (can be more than half of the incorrect keys) can be pruned and is shown in Figure 7(d). 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 1, 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(b). 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 get the input. These gates' outputs are similarly decided with X_1. So, the reason why P_2 trims substantially more incorrect keys does not depend on the input side. Consequently, the reason should be on the output response. 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_{k1} are 0, and subsequently, the inputs of OR gate G_5 must be 0 as well. This results in the unique solution for k_0 and k_1 with $k_0 = 0, k_1 = 0$.

A similar analysis can be performed on an AND gate, whose inputs are uniquely defined when the output is 1. In summary, having a response of 0 at the OR gate, or 1 at the AND gate is effectively splitting the cone into halves, where the 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 the inputs of the OR/AND gates and all the keys to these subcones can be evaluated simultaneously. This exponential reduction of search space helps to eliminate incorrect keys to linear complexity for the SAT attack, as presented in

Figure 4 of Section 4.1. Therefore, the efficiency of removing incorrect keys depends on IO pairs. The selection of an IO pair depends on the locked circuit topology that changes when adding more key bits.

5 Case Study: SAT-Attack Resiliency using 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. The common approach is to embed point functions right before the output of a logic cone, where the primary output response is altered under the designer's chosen input combinations. This section presents the theoretical analysis of point functions in AntiSAT [XS18] and CAS-Lock [SXTF20] and explains why they are breakable with SAT-based attacks. It is also applicable to any locking design using point functions with complementary blocks, g and $\overline{g}$. Each block has an n-bit key, *e.g.,* K_g for g and $K_{\overline{g}}$ for $\overline{g}$, where n is usually the size of the circuit input. The locked circuit is restored only if the two keys in blocks g and $\overline{g}$ are the same, $K_g = K_{\overline{g}}$. Anti-SAT [XS18] uses AND tree (Type-0, $p = 1$) to construct g and $\overline{g}$ while CAS-Lock [SXTF20] has a cascaded chain of AND gates with partial replacement to OR gates. Both approaches maximize the number of iterations required for SAT attack, where the tool finds an exponential number (2^n) of DIPs before it terminates and reports the correct key.

5.1 Brief Analysis on SAT-resistant Designs with Point Functions

Although the SAT attack takes an exponential number of iterations to prune off the incorrect keys completely, there is a major shortcoming in these SAT-resistant designs. The total size of keyspace for these logic locking schemes is 2^{2n}, as g and $\overline{g}$ each require an n-bit key. However, unlike the traditional logic locking, the correct key is not unique. Since the locked circuit can be restored when $K_g = K_{\overline{g}}$, there are 2^n correct keys. Let us now consider the ratio between the number of correct keys over the number of keys in the entire keyspace. For traditional logic locking techniques, the correct-key-ratio r_t is $r_t = \frac{1}{2^n}$ for the proper insertion of n-bit key. For the SAT-resistant design like AntiSAT and CAS-Lock, any locked circuit with the 2n-bit key has the correct-key-ratio r_s of

$$r_s = \frac{2^n}{2^{2n}} = \frac{1}{2^n} = r_t \tag{1}$$

which is the same ratio for any n-bit locking in the traditional logic locking domain. This indicates that, in theory, having an additional n-bit key does not increase the complexity of finding the correct key. In practice, there should be an attack (or multiple ones) to break these SAT-resistant schemes effectively.

Since the correct key is not unique, the attacker can render it ineffective if it is possible to shrink the key search space drastically. The goal is to trim the entire keyspace of 2^{2n} by order of 2^n, an exponential size decrease to reduce it to the equivalent search complexity of the common n-bit traditional logic locking.

Lemma 1. *Keyspace of 2^{2n} can be partitioned into 2^n subspaces with 2^n elements each, along with single a correct key for each subspace.*

Proof. Since K_g can be regarded as the free variable, we can vary the n-bit K_g to any key assignment in keyspace of 2^n so that the number of correct keys is reduced to 1, where $K_2 = K_1$. We partition the entire keyspace 2^{2n} based on the n-bit key value of K_g. Each subspace contains 2^n elements, where the n-bit $K_{\overline{g}}$ can vary with its 2^n combinations. As there are 2^n variations for K_g, there are 2^n subspaces. In addition, as the correct keys satisfy the condition $K_g = K_{\overline{g}}$, each subspace contains only one correct key. □

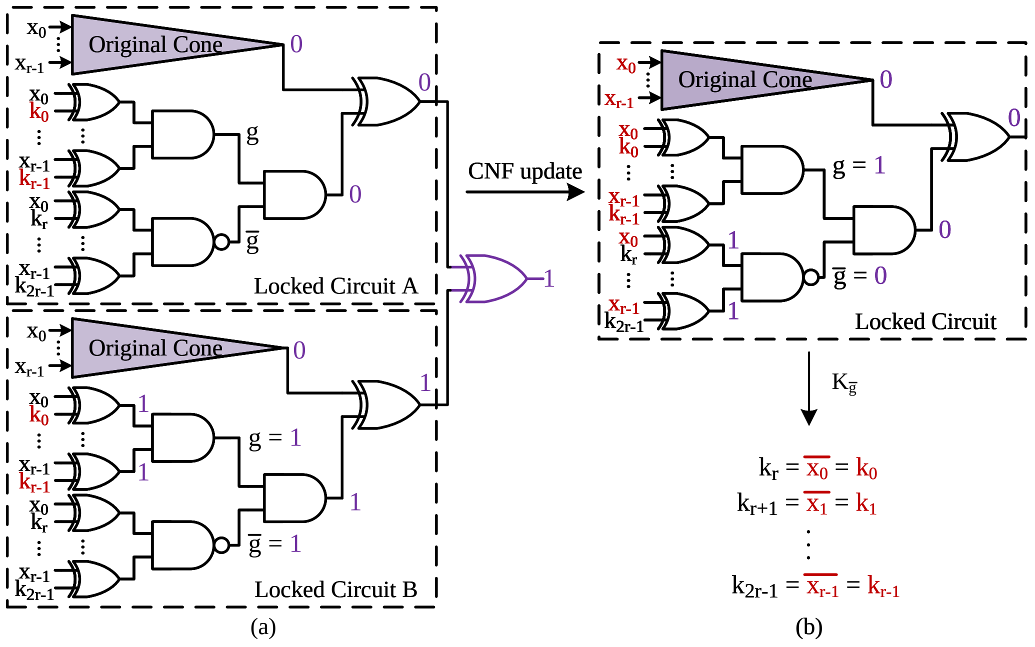

Figure 8: SAT attack on AntiSAT with fixed key K_g (a) SAT attack's miter construction (b) $K_{\overline{g}} = \{k_r - k_{2r-1}\}$ is uniquely determined with CNF update.

This indicates that the partition of key space based on K_g can effectively prune the keyspace to 2^n. The attacker only needs to focus on one key subspace and find ways to recover the unique key in that subspace. As for the adversary, the goal is to restore the locked circuit to its original state. If the attacker can discover one of the correct keys, it is sufficient to unlock the circuit, and there is no need to explore other correct key potentials. *In essence, if the adversary partition the keyspace strategically, logic locking schemes with minimal corruptibility are no better than the traditional logic locking.*

5.2 Deterministic Property of SAT Attack with Constraints

This section analyzes the SAT attack complexity on the locked circuits with a point function. Sengupta et al. [SLS21] has shown an effective approach to reduce the exponential complexity of AntiSAT and CAS-Lock to polynomial only still using the SAT attack. The proposed key-bit mapping (KBM) & SAT [SLS21] utilizes the connectivity between keys and inputs to accurately identify the $2r$-bit keys K_g and $K_{\overline{g}}$. Then, it applies the SAT attack with K_g fixed to a constant vector. In the following discussion, we show how SAT attack can uniquely determine the correct value for $K_{\overline{g}}$ with only one IO pair for AntiSAT. However, we show that constraining $K_{\overline{g}}$ to solve for K_g is still required exponential iteration.

Let us consider a circuit with r-bit input $X = \{x_0, ..., x_{r-1}\}$ and 1-bit output and we lock it with a total $2r$-bit keys with $K_g = \{k_0, k_1, ..., k_{r-1}\}$ and $K_{\overline{g}} = \{k_r, k_{r+1}, ..., k_{2r-1}\}$ of r-bit each. We assume the adversary already knows the key bit locations for K_g using the KBM of [SLS21]. Furthermore, the r-bit $K_g = \{k_0, ..., k_{r-1}\}$ is set to the value of a constant vector. As the SAT attack is applied next, we show how it finds a DIP and determines the key bits of $K_{\overline{g}}$ from it. The miter construction of the locked circuit is shown in Figure 8(a), where we highlight the key bits of K_g in red to indicate their values are fixed. As the miter creates differential output between the locked circuit A and B, without loss of generality, suppose circuit A has output 0 and circuit B has output 1. Since both circuits have the same original cone, the cone's output is the same in both circuit

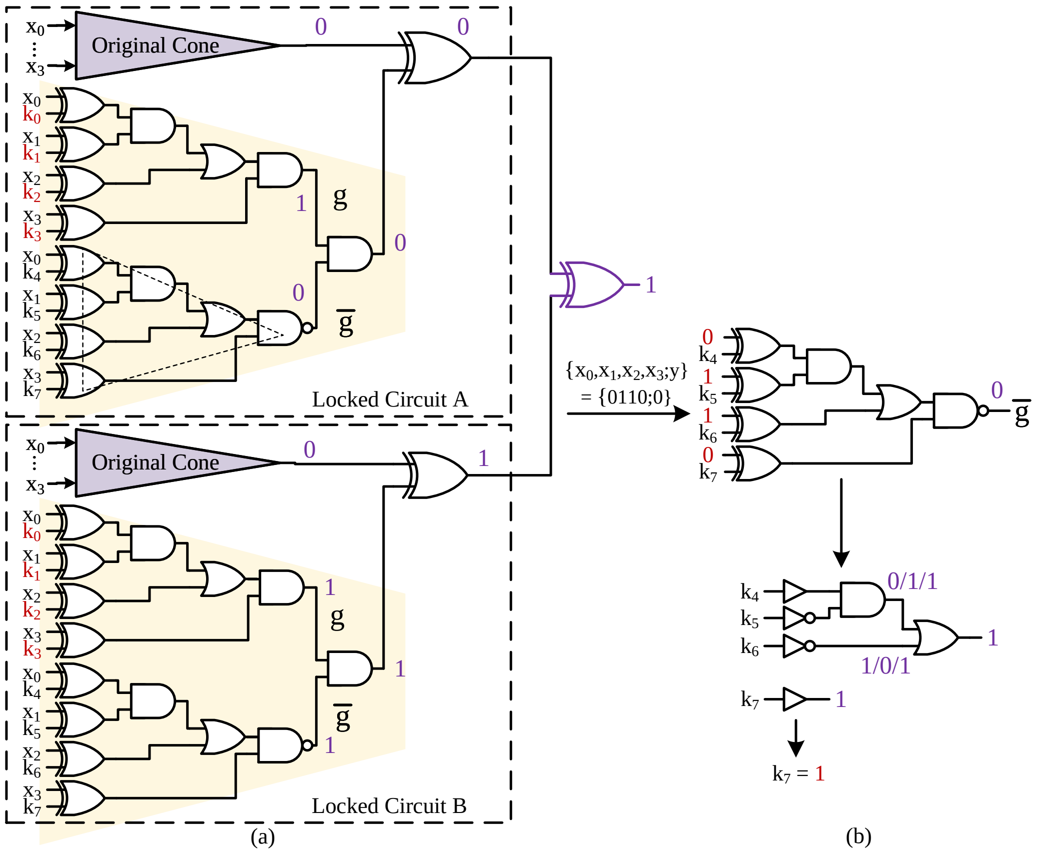

Figure 9: SAT attack on CAS-Lock with constrained K_g. (a) Miter construction. (b) Learned CNF relations of keys $k_4, ..., k_7$.

A and B as the inputs X is shared among both copies. Again, with loss of generality, we assume the output of the original cone with respect to the DIP found by the miter is logic 0. One can also assume to have logic 1 instead. This means that the AntiSAT block output in circuit A is 0 while 1 for circuit B's. As the AntiSAT block has AND at the output, both inputs of this AND gate in circuit B are 1. Within circuit B, we have $g = 1$, and all its inputs get logic 1 as well. Then, we know that the DIP X obtained through the miter must be the complement of K_g, $X = K_g$ to ensure all ones for the AND tree in g. Following the same procedures as in Section 3, the solver updates its CNF clauses with DIP X and its output response (logic 0 from the assumption), as shown in Figure 8(b). When we apply input vector X and its response to the locked circuit, we get a logic 0 at for the AntiSAT block. Since it is the same input pattern that gives $g = 1$ for circuit B, we still have $g = 1$ for the AntiSAT block. Then, $\overline{g}$ must be 0, $\overline{g} = 0$. As $\overline{g}$ is the output of the NAND gate, all its inputs have logic 1. This uniquely determines all the 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 terminates at the second iteration since the unknown r-bit key $K_{\overline{g}}$ is already resolved. Therefore, any constraint on K_g helps the SAT attack finish with 1 IO pair.

If the adversary decides to keep key $K_{\overline{g}}$ constant instead, he/she will not get the same efficiency in key derivation as in constraining K_g. Suppose we fix $K_{\overline{g}} = \{k_r, k_{r+1}, ..., k_{2r-1}\}$ to a constant r-bit vector. When SAT solver tries to find a satisfiable assignment to the miter circuit, following the same assumption as before, we know that both circuits A and B have the same logic 1 for the block $\overline{g}$. Having an output 1 at the NAND gate is equivalent to putting logic 0 to AND gate. There are $2^r - 1$ possible solutions for the r-bit input to get logic 1 at the output of $\overline{g}$. Equivalently, there are $2^r - 1$ choices for the DIP that satisfies the miter construction. When the tool updates solver's CNF with DIP and output response, we get $\overline{g} = 1$ for the NAND gate and $g = 0$ for the AND gate. Since the 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 the incorrect keys in the r-bit keyspace for K_g is $2^r - 1$. The total iterations required for the SAT attack is 2^r. Therefore, by constraining $K_{\overline{g}}$, the adversary can remove only one incorrect key. The overall SAT attack complexity still remains exponential.

We now provide an explanation of why constraining K_g to solve for $K_{\overline{g}}$ requires linear iterations for the CAS-Lock. We illustrate it with a $(2r = 8)$-bit CAS-Lock example, where block g and $\overline{g}$ have one OR gate each, as shown in Figure 9. The following description applies to any number of OR gates inside the cascaded AND chain of g and $\overline{g}$. When SAT attack searches for a DIP X to the miter circuit, as shown in Figure 9(a), one copy has logic 0 while the other has logic 1 for the CAS-Lock block. Suppose locked circuit B's CAS-Lock block is 1, where its inputs are $g = 1$ and $\overline{g} = 1$. The DIP X returned SAT solver must satisfy $g = 1$ as K_g is constrained. As block $\overline{g}$ constrains OR gate, the inputs cannot be uniquely identified as its output in 1 (shown in Figure 9(b). Instead, we can model this block as a traditional locked circuit, described in Section 3. If we apply a DIP of 0110, the key bits $(K_4, \ldots, k_7)$ can have the relations shown in Figure 9(b). In summary, the same analysis can be performed for block $\overline{g}$ mentioned in Section 3, and multiple iterations are required to determine the entire key compared to a single iteration for the AntiSAT locking.

6 Future Directions

Even though SAT attack has demonstrated linear trends in solving the secret key, we believe it is possible for a logic design to achieve SAT resiliency. Here, we discuss how future locking schemes should consider the drastical increase in the hardness of their design against the SAT attack. Besides targeting the exponential iterations required for the SAT attack, it may be feasible to significantly increase the time for SAT solver to find each satisfiable assignment for the miter circuit, which result in a longer computation time within each iteration.

6.1 Computation Time Analysis

As described in SAT attack [SRM15], the authors stated that the multiplier benchmark c6288 is inherently challenging to SAT solver, which is excluded from the analysis. We locked its largest cone, N6288, the same way as for other benchmarks in ISCAS'85 as described in Section 4.1. From the perspective of total iterations, the overall SAT complexity remains linear with key size $|K|$, which is shown in Figure 4. This suggests that c6288 behaves identically to other ISCAS'85 benchmarks. In addition, one can also observe a decrease in complexity when more keys are inserted, as shown in Figure 5. 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 preprocessing time, *i.e.,* setting up arrays of literals, initializing solver *etc.* The postprocessing time is also excluded from the CPU time, *i.e.,* displaying the correct keys and the overall status *etc.* Table 2 lists the time duration for SAT solver to derive the correct keys. The first column shows the key size $|K|$ of the locked c6288_N6288 cones. Column 2 records the total IO pairs $|P|$ required to derive the correct key. Column 3 is the total time the SAT solver spent, which we break down into two parts, the time used for generating IO pairs, Column 4, and for checking that no more DIP exists (UNSAT), Column 6. Column 5 gives the average time SAT solver takes to find each IO pair. Column 7 reports the time ratio SAT solver needs to reach the UNSAT decision with respect to the total time spent in SAT solving. The interesting observation is that the major time was not on finding DIPs to prune the keyspace, but was spent 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 with the time spent in the very last iteration (UNSAT). In particular, the time duration for UNSAT with respect to the total time span is generally over 90%.

Table 2: SAT attack running time on c6288_N6288 with XOR locking, including the total time spent on each locked circuit with different key size, total time spent on finding DIPs, averaged time per DIP, and time to reach the UNSAT conclusion.

$\|\boldsymbol{K}\|$	$\|\boldsymbol{P}\|$	**CPU time (s)**				$\frac{\textbf{UNSAT}}{\textbf{Total}}$ (%)
		Total	**IO Pairs**	**Average**	**UNSAT**	
1	1	86.35056	0.09108	0.09108	86.25948	99.89452
2	2	84.43924	0.102893	0.051447	84.33634	99.87815
3	3	86.55111	0.110187	0.036729	86.44092	99.87269
4	4	88.80421	0.11963	0.029908	88.68458	99.86529
5	4	79.61421	0.127045	0.031761	79.48717	99.84042
6	4	62.04783	0.116301	0.029075	61.93153	99.81256
7	4	88.08827	0.118216	0.029554	87.97006	99.86580
8	4	66.76216	0.113295	0.028324	66.64887	99.83030
9	5	78.43435	0.123854	0.024771	78.31049	99.84209
10	7	62.01792	0.147880	0.021126	61.87004	99.76155
11	8	72.61459	0.159248	0.019906	72.45534	99.78069
12	6	66.56000	0.195316	0.032553	66.36468	99.70656
13	9	74.61156	0.221303	0.024589	74.39026	99.70339
14	8	78.49215	0.147602	0.018450	78.34455	99.81195
15	10	77.13255	0.172048	0.017205	76.96051	99.77695
16	11	83.07691	0.237652	0.021605	82.83926	99.71394
17	11	85.08259	5.704183	0.518562	79.37841	93.29571
18	15	72.31732	0.300823	0.020055	72.01650	99.58402
19	15	89.65450	0.346194	0.023080	89.30831	99.61386
20	14	92.58854	0.325860	0.023276	92.26268	99.64806
21	15	67.43085	0.452500	0.030167	66.97835	99.32894
22	12	80.29901	0.266420	0.022202	80.03259	99.66822
23	19	88.22816	0.639116	0.033638	87.58904	99.27561
24	15	76.82473	0.421037	0.028069	76.40369	99.45195
25	20	88.29527	2.484020	0.124201	85.81125	97.18669
30	16	73.06461	0.849540	0.053096	72.21507	98.83728
35	29	86.73668	14.53748	0.501292	72.19920	83.23953
40	27	149.0966	13.34636	0.494310	135.7502	91.04851
45	41	1130.466	18.31241	0.446644	1112.154	98.38010
50	37	84.40455	6.167174	0.166680	78.23738	92.69332
55	45	1188.844	57.14645	1.269921	1131.698	95.19311

Therefore, the possible consideration for SAT-resistant designs is to increase the number of backtracks and logic reassignment required for SAT solvers so that the time spent in each iteration is dramatically increased. In addition, the locking designs should provide sufficient difficulty for the conflict-driven clause learning (CDCL) algorithm [GV20], which is commonly employed in present-day SAT solvers, to learn the correct key value.

7 Conclusion

In this paper, we provide a new perspective to analyze the efficiency of the SAT attack based on how CNF clauses are updated inside the SAT solver. In each iteration, the SAT attack records the interdependencies between key bits from the distinguishing input pattern and its output response. Any locked circuit with multiple logic cones facilitates the removal of incorrect key combinations as the effect of the keys can be propagated through multiple outputs. We further investigate the SAT attack complexity with the same locked cone of increasing key size. We observed a non-monotonous increase in SAT complexity under more key bits. The insertion of additional key bits does not guarantee a strict linear growth in the SAT attack complexity. Instead, the dips in complexity happen to all the ISCAS'85 benchmark cones we locked. We subsequently offer an explanation of this phenomenon based on the oracle's response and logic gate types, where more incorrect keys are eliminated from the search space with an IO pair. In addition, we give analytical reasoning to show how the constraining of key bits would aggressively reduce the search complexity of finding the correct key for point-function-based logic locking schemes. Finally, we furnish our discussion on SAT-resistant designs with interesting observations, expectations, and future directions.

References

[AK07] Yousra Alkabani and Farinaz Koushanfar. Active Hardware Metering for Intellectual Property Protection and Security. In *USENIX security symposium*, pages 291–306, 2007.

[AKP07] Yousra Alkabani, Farinaz Koushanfar, and Miodrag Potkonjak. Remote activation of ICs for piracy prevention and digital right management. In *Proc. of IEEE/ACM int. conf. on Computer-aided design*, pages 674–677, 2007.

[APK$^+$21] Lilas Alrahis, Satwik Patnaik, Faiq Khalid, Muhammad Abdullah Hanif, Hani Saleh, Muhammad Shafique, and Ozgur 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)*, pages 780–785. IEEE, 2021.

[BA04] Michael Bushnell and Vishwani Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*, volume 17. Springer Science & Business Media, 2004.

[BT18] Swarup Bhunia and Mark Tehranipoor. *Hardware Security: A Hands-on Learning Approach.* Morgan Kaufmann, 2018.

[BTZ10] Alex Baumgarten, Akhilesh Tyagi, and Joseph Zambreno. Preventing IC Piracy Using Reconfigurable Logic Barriers. *IEEE Design & Test of Computers*, 27(1):66–75, 2010.

[CB08] Rajat Subhra Chakraborty and Swarup Bhunia. Hardware protection and authentication through netlist level obfuscation. In *Proc. of IEEE/ACM International Conference on Computer-Aided Design*, pages 674–677, 2008.

[Cha98] Edoardo Charbon. Hierarchical watermarking in IC design. In *Proc. of the IEEE Custom Integrated Circuits Conference*, pages 295–298, 1998.

[CMBG+07] Encarnacin Castillo, Uwe Meyer-Baese, Antonio García, Luis Parrilla, and Antonio Lloris. IPP@HDL: Efficient Intellectual Property Protection Scheme for IP Cores. *IEEE Trans. on VLSI (TVLSI)*, pages 578–591, 2007.

[GSFT16] Ujjwal Guin, Qihang Shi, Domenic Forte, and Mark M Tehranipoor. FORTIS: a comprehensive solution for establishing forward trust for protecting IPs and ICs. *Transactions on Design Automation of Electronic Systems (TODAES)*, 21(4):63, 2016.

[GV20] Vijay Ganesh and Moshe Y Vardi. On the unreasonable effectiveness of sat solvers., 2020.

[HL08] Jiawei Huang and John Lach. IC activation and user authentication for security-sensitive systems. In *IEEE International Workshop on Hardware-Oriented Security and Trust*, pages 76–80, 2008.

[JRG20] Ayush Jain, Tanjidur Rahman, and Ujjwal Guin. ATPG-Guided Fault Injection Attacks on Logic Locking. In *IEEE Physical Assurance and Inspection of Electronics (PAINE)*, pages 1–6, 2020.

[KLMS+01] Andrew B Kahng, John Lach, William H Mangione-Smith, Stefanus Mantik, Igor L Markov, Miodrag Potkonjak, Paul Tucker, Huijuan Wang, and Gregory Wolfe. Constraint-based watermarking techniques for design IP protection. *IEEE Transactions on CAD of Integrated Circuits and Systems*, pages 1236–1252, 2001.

[KPKG02] Andreas Kuehlmann, Viresh Paruthi, Florian Krohm, and Malay K Ganai. Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(12):1377–1394, 2002.

[LPS21] Nimisha Limaye, Satwik Patnaik, and Ozgur Sinanoglu. Fa-SAT: Fault-aided SAT-based Attack on Compound Logic Locking Techniques. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1166–1171. IEEE, 2021.

[MATa] MATLAB. https://www.mathworks.com/help/matlab/ref/digraph.flipedge.html.

[MATb] MATLAB. https://www.mathworks.com/help/matlab/ref/graph.bfsearch.html.

[QCF+16] Shahed E Quadir, Junlin Chen, Domenic Forte, Navid Asadizanjani, Sina Shahbazmohamadi, Lei Wang, John Chandy, and Mark Tehranipoor. A survey on chip to system reverse engineering. *ACM journal on emerging technologies in computing systems (JETC)*, 13(1):1–34, 2016.

[QP07] Gang Qu and Miodrag Potkonjak. *Intellectual Property Protection in VLSI Designs: Theory and Practice.* Springer Sc. & Business Media, 2007.

[RKM08] Jarrod A Roy, Farinaz Koushanfar, and Igor L Markov. EPIC: Ending Piracy of Integrated Circuits. In *Proceedings of the conference on Design, automation and test in Europe*, pages 1069–1074, 2008.

[RPSK12] Jeyavijayan Rajendran, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. Security analysis of logic obfuscation. In *Proc. of Annual Design Automation Conference*, pages 83–89, 2012.

[RZZ$^+$13] Jeyavijayan Rajendran, Huan Zhang, Chi Zhang, Garrett S Rose, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. Fault analysis-based logic encryption. *IEEE Transactions on computers*, 64(2):410–424, 2013.

[RZZ$^+$15] Jeyavijayan Rajendran, Huan Zhang, Chi Zhang, Garrett S Rose, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. Fault Analysis-Based Logic Encryption. *IEEE Transactions on computers*, pages 410–424, 2015.

[Shi22] Willy Shih. Intel's $88 Billion European Expansion Is Part Of A New Phase In The Globalization Of The Semiconductor Industry. Forbes, Mar 2022.

[SLM$^+$17] Kaveh Shamsi, Meng Li, Travis Meade, Zheng Zhao, David Z Pan, and Yier Jin. AppSAT: Approximately deobfuscating integrated circuits. In *Int. Symposium on Hardware Oriented Security and Trust (HOST)*, pages 95–100, 2017.

[SLS21] Abhrajit Sengupta, Nimisha Limaye, and Ozgur Sinanoglu. Breaking cas-lock and its variants by exploiting structural traces. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 418–440, 2021.

[SNL$^+$20] Abhrajit Sengupta, Mohammed Nabeel, Nimisha Limaye, Mohammed Ashraf, and Ozgur Sinanoglu. Truly stripping functionality for logic locking: A fault-based perspective. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[SRM15] Pramod Subramanyan, Sayak Ray, and Sharad Malik. Evaluating the security of logic encryption algorithms. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 137–143, 2015.

[SS19] Deepak Sirone and Pramod Subramanyan. Functional analysis attacks on logic locking. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 936–939, 2019.

[SS20] Deepak Sirone and Pramod Subramanyan. Functional Analysis Attacks on Logic Locking. *IEEE Transactions on Information Forensics and Security*, 15:2514–2527, 2020.

[SXTF20] Bicky Shakya, Xiaolin Xu, Mark Tehranipoor, and Domenic Forte. Cas-lock: A security-corruptibility trade-off resilient logic locking scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 175–202, 2020.

[Syn] Synopsys Design Compiler. Synopsys, Inc., 2017.

[SZ17] Yuanqi Shen and Hai Zhou. Double DIP: Re-Evaluating Security of Logic Encryption Algorithms. In *Proceedings of the Great Lakes Symposium on VLSI*, pages 179–184, 2017.

[TGF15] Mark Mohammad Tehranipoor, Ujjwal Guin, and Domenic Forte. Counterfeit integrated circuits. In *Counterfeit Integrated Circuits*. Springer, 2015.

[TJ07] Randy Torrance and Dick James. Reverse engineering in the semiconductor industry. In *2007 IEEE Custom Integrated Circuits Conference*, pages 429–436. IEEE, 2007.

[TJ09] Randy Torrance and Dick James. The State-of-the-Art in IC Reverse Engineering. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 363–381, 2009.

[TJ11] Randy Torrance and Dick James. The state-of-the-art in semiconductor reverse engineering. In *Proc. of the Design Automation Conference*, pages 333–338, 2011.

[TW11] Mohammad Tehranipoor and Cliff Wang. *Introduction to Hardware Security and Trust*. Springer Science & Business Media, 2011.

[XS18] Yang Xie and Ankur Srivastava. Anti-sat: Mitigating sat attack on logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(2):199–207, 2018.

[XSTF17] Xiaolin Xu, Bicky Shakya, Mark M Tehranipoor, and Domenic Forte. Novel Bypass Attack and BDD-based Tradeoff Analysis Against All Known Logic Locking Attacks. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 189–210, 2017.

[YMRS16] Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan JV Rajendran, and Ozgur Sinanoglu. SARLock: SAT attack resistant logic locking. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 236–241, 2016.

[YMSR17] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu, and Jeyavijayan Rajendran. Removal attacks on logic locking and camouflaging techniques. *Transactions on Emerging Topics in Computing*, 2017.

[YSN$^+$17] Muhammad Yasin, Abhrajit Sengupta, Mohammed Thari Nabeel, Mohammed Ashraf, Jeyavijayan JV Rajendran, and Ozgur Sinanoglu. Provably-Secure Logic Locking: From Theory To Practice. In *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, pages 1601–1618, 2017.

[YSS$^+$17] Muhammad Yasin, Abhrajit Sengupta, Benjamin Carrion Schafer, Yiorgos Makris, Ozgur Sinanoglu, and Jeyavijayan JV Rajendran. What to Lock?: Functional and Parametric Locking. In *Proc. of Great Lakes Symposium on VLSI*, pages 351–356, 2017.

[ZCZG19] Yuqiao Zhang, Pinchen Cui, Ziqi Zhou, and Ujjwal Guin. TGA: An Oracle-less and Topology-Guided Attack on Logic Locking. In *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop*, pages 75–83, 2019.