DIP Learning on CAS-Lock: Using Distinguishing Input Patterns for Attacking Logic Locking

Akashdeep Saha[1], Urbi Chatterjee[2], Debdeep Mukhopadhyay[1], Rajat Subhra Chakraborty[1]
[1]*Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, West Bengal, India*
[2]*Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur, Uttar Pradesh, India*
akashdeep@iitkgp.ac.in, urbic@cse.iitk.ac.in, debdeep@cse.iitkgp.ac.in, rschakraborty@cse.iitkgp.ac.in

***Abstract*—The globalization of the integrated circuit (IC) manufacturing industry has lured the adversary to come up with numerous malicious activities in the IC supply chain. Logic locking has risen to prominence as a proactive defense strategy against such threats. CAS-Lock (proposed in CHES'20), is an advanced logic locking technique that harnesses the concept of single-point function in providing SAT-attack resiliency. It is claimed to be powerful and efficient enough in mitigating existing state-of-the-art attacks against logic locking techniques. Despite the security robustness of CAS-Lock as claimed by the authors, we expose a serious vulnerability and by exploiting the same we devise a novel attack algorithm against CAS-Lock. The proposed attack can not only reveal the correct key but also the exact AND/OR structure of the implemented CAS-Lock design along with all the key gates utilized in both the blocks of CAS-Lock. It simply relies on the externally observable *Distinguishing Input Patterns* (DIPs) pertaining to a carefully chosen key simulation of the locked design without the requirement of structural analysis of any kind of the locked netlist. Our attack is successful against various AND/OR cascaded-chain configurations of CAS-Lock and reports 100% success rate in recovering the correct key. It has an attack complexity of $\mathcal{O}(m)$, where m denotes the number of DIPs obtained for an incorrect key simulation.**

***Index Terms*—Logic locking, CAS-Lock, DIP-driven attack**

I. Introduction

Over the past few years, the skyrocketing cost of IC manufacturing has forced the majority of enterprises to shift to fabless operation. The IC design is often outsourced to foreign fabrication laboratories to reduce time and cost. However, as a flip side, it has introduced various hardware security concerns at different stages in the supply chain, namely, IP piracy, illegal overproduction, design counterfeiting, etc. To mitigate these emerging threats from time to time, IC metering, camouflaging, watermarking, split manufacturing, and logic locking [1]–[4] has been proposed in state-of-the-art literature. In particular, logic locking have emerged as the most promising countermeasure against these threats. Logic locking has the capability of mitigating a potential adversarial attack at any location along the supply chain. The basic idea of logic locking is to obscure the actual design to an adversary by integrating a key-based logic into the original design. The design behaves correctly only on the application of the secret key. The correct (secret) key is often stored in a non-volatile on-chip memory module from which it is retrieved to unlock the circuit. The chip is activated by the IP owner by loading the correct key values into the memory module only after fabrication of the IC, or after implementation on a FPGA. The adversary can obtain the locked design netlist by reverse-engineering (RE); however, without the correct key being applied, the IP (and by extension, the design of which it is a part) does not function correctly. Logic locking techniques have evolved over the years from randomly inserting key gates into the original circuit to carefully analyzing the effect of the inserted gates. One of such prominent attacks that completely broke all existing locking techniques then was the Boolean satisfiability-based (SAT) attack [6]. It is based on effective pruning of the incorrect keyspace, such that the correct key is restricted in a relatively small key sub-space. To mount a SAT-based attack, an adversary requires (i) an oracle or a functional chip and (ii) a locked netlist. The attack proceeds by finding out distinguishing input patterns (DIPs), which in turn can eliminate incorrect keys. It is to be noted that DIPs are input patterns which produce different output on the application of different key values, whereas the oracle helps to distinguish the incorrect keys from the correct ones. To mitigate SAT-based attacks, two general defense strategies have been investigated: (i) SAT-Hard constructions [4], that cause successive SAT iterations to take exponential time to solve, and (ii) low output corruptibility based constructions [2], [3], [5], such that SAT-based attacks are reduced to brute-force search depending on the key complexity. These schemes use the concept of a one-point function to invert the output for a single (or very limited number) input pattern. Hence, the SAT-based attack requires an exponential number of DIPs to eliminate all the incorrect keys.

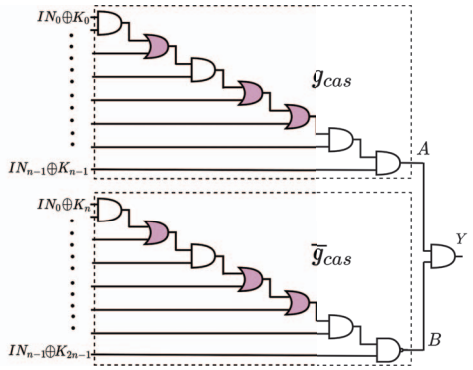

Fig. 1: An instance of CAS-Lock [5]. The blocks g_{cas} and $\overline{g}_{cas}$ are complimentary in nature. They both receive same set of input bits. However, the choice of XOR/XNOR gates are independent in the blocks.

Two locking schemes SARLock [7] and Anti-SAT [2] have utilized this concept to achieve SAT-based attack resiliency. The output corruption is extremely low for these schemes which makes them vulnerable to bypass [8] or removal attacks [9].

Since an incorrect key produces a single incorrect input-output pair in the entire input-output space, the attacker can simply add a bypass circuitry to rectify the output for that specific DIP and produce a fully functional design at a low cost. Further, the isolation and elimination of the protection circuitry exposes the original design to an adversary.

To mitigate the aforementioned threats, stripped-functionality logic locking (SFLL-HD^h) [3] was proposed, that maintains a higher output corruption. SFLL-HD^h modifies the original circuit such that it is secure against bypass and removal attacks. It deploys a Hamming distance (h) based comparator between the input and the applied key to produce incorrect outputs. Therefore, it produces several incorrect input-output pairs which makes it infeasible to attach a bypass circuitry to rectify the faulty design for an incorrect key. Similarly, removal of the protection circuitry leaves the design faulty. However, the functional analysis attack (FALL) [10] exploits the structural traces of the locked designs and extracts the hard-coded correct key by analyzing the re-synthesized design of the locked netlist. A modified version of Anti-SAT, which proposes to change AND-gates in the AND-tree structure to OR gates (likewise converting OR gates to AND gates in an OR-tree structure) is CAS-Lock [5]. This state-of-the-art logic locking technique has been proposed to combat an union of all existing attacks on logic locking schemes.

Recently, an attack against CAS-Lock was proposed [11] that largely depends on structural analysis of the locked circuit to mitigate CAS-Lock (elaborated in subsequent sections). Structural analysis is often infeasible requiring complex investigation of the locked netlist. The requirement of structural analysis can be eliminated if an adversary can *learn* crucial information regarding the locked circuit, simply by observing the input-outputs. Unfortunately, none of the state-of-the-art attack methodologies have examined *the possible existence of a relationship between the correct key of the locked circuit and the externally observable DIPs* (input patterns that disagree with the oracle). To the best of our knowledge, this work demonstrates the first successful exploitation of this property to develop an attack algorithm that could unlock CAS-Lock and its various configurations. The main contributions in this paper can be summarized as follows:

- We present a mathematical formulation to calculate the maximum number of DIPs produced for a particular CAS-Lock chain configuration, which is vital for establishing robustness of CAS-Lock towards various attacks. Further, we propose a miter circuit construction that does not generate any undetected DIPs for CAS-Lock.
- We present a novel DIP-based key retrieval attack against CAS-Lock. Our attack algorithm extracts the correct key and the exact chain configuration of CAS-Lock. It can also identify all the key gates utilized in CAS-Lock. Further, it does not require any structural analysis of the netlist.

This simple yet effective attack will encourage the development of more secure logic locking schemes, to prevent any exploitable information leakage to an external adversary.

The rest of the paper is organized as follows. In Sec. II we present an overview of CAS-Lock and related attacks against it. Sec. III highlights the exploitable vulnerabilities and our attack algorithms. Sec. IV describes the attack complexity and experimental results are discussed in Sec. V. Finally, we conclude in Sec. VI.

II. Preliminaries

In this section, we discuss in detail the CAS-Lock locking technique and the existing attack against it.

A. The CAS-Lock Locking Technique

CAS-Lock [5] is a recently proposed locking technique that differs from Anti-SAT only in the construction of the complementary blocks. Here, instead of the AND-tree structure (in Anti-SAT), the AND/OR gates are daisy-chained or cascaded together to construct the complementary blocks (g_{cas} and $\overline{g}_{cas}$) as illustrated in Fig. 1. A random combination of XOR/XNOR key gates with the input layer are present for both the blocks. These newly constructed blocks are resistant to SAT-based attacks. Further, for every wrong key, CAS-Lock produces a set of DIPs, making it infeasible to attach a bypass circuitry to circumvent them all. Similar to Anti-SAT, CAS-Lock can be prone to signal probability skew (SPS) based removal attack [9]. The authors in [5] have proposed a modified version of the scheme, referred to as Mirrored CAS-Lock (M-CAS), illustrated in Fig. 2. In M-CAS, the CAS-Locked circuit is again locked with another CAS-Lock instance identical to the former one. This ensures security against removal attack since even if the outer CAS-Lock is removed, the resultant circuit does not guarantee of correct functionality. The two CAS-Lock structures are identical to each other. It functions correctly (i.e., Y_{orig} = Y_o in Fig. 2), only if K_{inner} = K_{outer}. The authors argue that even if the outer CAS-Lock structure is removed using the SPS-based analysis [9], the locked circuit will be rendered non-functional unless the correct key for the inner CAS-Lock structure (K_{inner}) is applied.

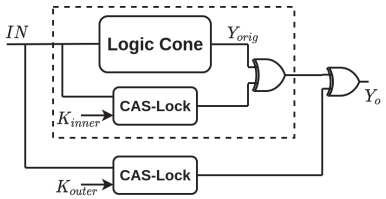

Fig. 2: An overview of the general structure of the Mirrored CAS-Lock (M-CAS) scheme.

B. Existing Attack

Initially, an attack against CAS-Lock was proposed in [12] that claimed to nullify it by simply assigning both g_{cas} and $\overline{g}_{cas}$ keys to all 0's or all 1's. This ensured that g_{cas} and $\overline{g}_{cas}$ are always complementary for all input patterns producing $Y = 0$ (in Fig. 1). However, the ineffectiveness of the aforementioned attack was highlighted in [13]. It proved that the attack [12] was based upon a misinterpretation of CAS-Lock implementation and will be successful for only one particular instance where all the key gates are either XOR or XNOR gates. Another attack defeating CAS-Lock has been proposed recently [11], that has a couple of variants namely, *Identify flip signal (IFS)*

and *Key-bit flip mapping & SAT (KBM-SAT)*. The IFS attack exploits the structural traces left behind after re-synthesis of the RE locked netlist to identify the flip signal Y. The CAS-Lock security is nullified by simply fixing $Y = 0$. This implies that a flip is never introduced in the original circuit even upon the application of an incorrect key. However, it does not extract the correct key for CAS-Lock. KBM-SAT attack variant, on the other hand, extracts the correct key using SAT-based attack [6]. It begins with identifying the mapping of key-bits in both g_{cas} and $\overline{g}_{cas}$, that acts upon the same input in both the blocks, pairing them together as bit-symmetric pairs. IFS-SAT attack exploits the structural flaws in M-CAS and utilizing SAT-based attack, it identifies the correct key of M-CAS scheme.

The proposed attack against CAS-Lock has several differences compared to the existing attacks [11]. Firstly, it does not exploit the structural traces of the locked netlist to identify the flip signal (Y) introduced by the locking circuitry. The structural analysis of the locked netlist assumes that the adversary has full knowledge of the synthesis tools, technology mapping, etc. used in the locking design [11], which is often non-trivial. Secondly, the philosophy of the proposed attack is entirely different. It simply exploits the information leaked by the primary input-output pairs upon the application of an incorrect key to derive the correct key. It does not harness the SAT-based attack directly to extract the correct key assignment. Instead, it uses the SAT solver only to extract the DIPs [8]. *Further, this attack can extract the entire structure of the implemented CAS-Lock, like the position of the AND/OR gates in the chain and the nature of the key gates utilized in g_{cas} and $\overline{g}_{cas}$.* Thirdly, it does not require the mapping constraints as defined for KBM-SAT [11]. Instead of bit-symmetric pairs, it only needs to assign key values in a block-wise manner. This can avoid various potential challenges, described in [11], to identify bit-symmetric pairs.

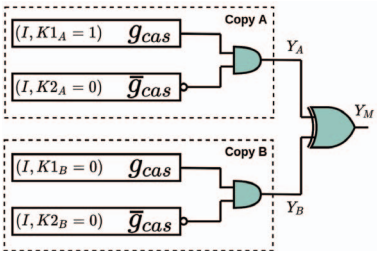

Fig. 3: Miter circuit for DIP set extraction. Copy A and copy B are two locked netlist with different key assignments for AND/NAND terminated CAS-Lock.

III. Proposed Attack Algorithm on CAS-Lock

The proposed attack on CAS-Lock can be broadly summarized as follows:

1) The attack begins with the extraction of the DIP set I_l.
2) The non-repeating DIP (DIP_{nc}) and the chain configuration (AND/OR gate positions) are derived from I_l
3) Using the chain configuration information and the DIP_{nc}, the key gates of g_{cas} are derived.
4) Using the size of I_l, the key gates of $\overline{g}_{cas}$ ranging from the position of the last OR gate (OR_{last}) till the terminating gate are extracted.
5) Finally, key gates 0 to OR_{last} of $\overline{g}_{cas}$ are identified from I_l, thus completing the CAS-Lock key recovery.

We exploit the relationship between the location of the OR gates in the chain and the number of DIPs produced to *learn* the position of AND/OR gates in the blocks. To extract the correct key of the CAS-Lock scheme we use the concept of divide-and-conquer. The first three steps are detailed in Algo. 1, whereas the last two steps are elaborated in Algo. 2.

A. Adversarial Model

The threat model assumes that the adversary possesses the locked netlist of the design. The attacker also has access to an oracle or a functional chip through which they can verify the correctness of the output obtained for a given input. S/he aims to unlock the design so that it can be pirated. The locked netlist is treated as a black box and the attacker has the capability of simulating it with arbitrary input vectors. The attack model is realistic and conforms with the standard threat models assumed for existing attacks against logic locking [6], [9].

B. The DIP Set (I_l) Extraction

We briefly discuss the procedure of I_l set extraction in this section. We follow the same procedure as proposed in the bypass attack [8] for extracting the elements of I_l. It begins with the construction of the miter circuit. *However, this proposed miter circuit construction (Fig. 3) can eliminate the occurrence of any undetected DIPs and for a particular key assignment can divulge all the DIPs.* We discuss the construction below.

- The miter circuit consists of two copies of the locked netlist: copy A and copy B. The copy A has $K1_A = 1$ (g_{cas} keys) and $K2_A = 0$ ($\overline{g}_{cas}$ keys), whereas copy B has both $K1_B = 0$ and $K2_B = 0$ as the chosen key values.
- However, as mentioned in [5], there still exists a challenge that the miter circuit will be unable to detect those input patterns which results in $Y_A = Y_B$.
- But if we take a closer look at our miter construction in Fig. 3 and intelligently choose the key assignments for the two copies, then this problem can be addressed. Such key assignment ensures that the DIPs for copy A and copy B will be distinct and differ in their last bit (LSB) values. We formally prove this in Lemma 1.

Lemma 1. ***The miter circuit for CAS-Lock with two copies A and B, does not generate any undetectable DIPs, if the keys are assigned as follows:***

Case 1: In an AND/NAND terminated CAS-Lock scheme, g_{cas} blocks are assigned '1' and '0' in copy A and copy B, respectively, and $\overline{g}_{cas}$ blocks are assigned '0' in both copies.

Case 2: In an OR/NOR terminated CAS-Lock scheme, g_{cas} blocks are assigned '0' in both copies, whereas $\overline{g}_{cas}$ blocks are assigned to '1' and '0' in copy A and in copy B, respectively.

Proof. As illustrated in Fig. 3, let I be an n-bit undetectable DIP (where n is the input size of each CAS-Lock block), which has the binary representation as $\{i_0, \cdots, i_{n-1}\}$ and produces both $Y_A = 1$ and $Y_B = 1$, guiding $Y_M = 0$. Without loss of generality, I is XOR/XNORed prior to entering the CAS-Lock

blocks. $Y_A = 0$ and $Y_B = 0$ is not considered, since it implies that I is not a DIP.

Case 1: Let us assume that g_{cas} and $\overline{g}_{cas}$ are terminated by AND and NAND gates, respectively. Here, the last bit of I (i.e. i_{n-1}) enters the terminating AND gate after the XOR/XNOR key operation with the last key bit. The $\overline{g}_{cas}$ are identical in both the copies of the miter, with the same key assignment '0'. The g_{cas} blocks for both copies differ only in their key assignment values ('1' in copy A and '0' in copy B). It can be stated that the g_{cas} block produces the DIPs corresponding to both the copies, since the $\overline{g}_{cas}$ block is identical.

The bit i_{n-1} is XOR/XNORed with '1' and '0' in copy A and copy B, respectively, and produces a '1' as input to the terminating AND gate of g_{cas} in both copies to become an overlapping/undetectable DIP. This contradicts our assumption of I, as copy A and copy B are identical instances only differing in the key assignments.

Case 2: The g_{cas} and $\overline{g}_{cas}$ blocks are terminated by OR and NOR gates, respectively. This follows from Case 1, here $\overline{g}_{cas}$ differ only in their key assignment values ('0' in copy A and '1' in copy B). The g_{cas} blocks are identical in both copy A and copy B, with the same key assignment ($=$ '0'). With this minor modification in the key assignments for the two copies of the miter circuit, we proceed the same way as for Case 1, and prove the contradiction in our assumption about I. Thus, our proposed miter circuit configuration will not generate any undetectable DIPs for CAS-Lock (the Step 1 of the attack). □

C. Attack Insight and Algorithms

Here we shall elaborate in detail the proposed algorithms in successfully attacking CAS-Lock. A gate (G) in g_{cas}, receives one input directly ($I1$) from the key gate output whereas the other input ($I2$) arrives from the output of cascaded chain presiding G (Fig. 4). Now, if G happens to be an OR gate and a controlling input value of '1' is produced at $I1$ of G in g_{cas}, irrespective of the value at $I2$, '1' gets propagated down the block. In this way, input patterns are obtained when $I1$ is set to the controlling value ('1') and for all possible value assignments of presiding inputs (till gate $G-1$), and for a 'fixed assignment' to input values for gates following G (i.e., for gates $G+1$ to $n-1$) after key operation, in g_{cas} produces '1' at g_{cas} output. The 'fixed assignment' implies an input pattern that assigns the gates $G+1$ to $n-1$ to non-controlling input values, allowing the output ('1') of G to propagate down the entire block and produce '1' at g_{cas} output. There is exactly one input assignment (corresponding to gates $G+1$ to $n-1$), that produces non-controlling input values for the gates $G+1$ to $n-1$ (after XOR/XNOR operation). This is true for all OR gates in g_{cas}. The input patterns that produce '1' at g_{cas} output are assumed to be DIPs, since $\overline{g}_{cas}$ produces a '1' for half of the entire input space. This happens due to terminating NAND gate of $\overline{g}_{cas}$. Depending upon the nature of the key gate at the input of this terminating NAND gate, either all the even input patterns or the odd input patterns produce a '1' at $\overline{g}_{cas}$ output. Further, NAND gate has a higher probability of producing a '1' at its output (= 0.75 for a 2-input NAND gate). Whenever a '0' is produced at any of an AND gate input (after the last OR gate in the chain) or at the output of OR gate in $\overline{g}_{cas}$, it propagates down to the terminating NAND gate producing $\overline{g}_{cas}$ output '1'. However, there is a case in which an input pattern produces '1' at g_{cas} output, but a '0' at $\overline{g}_{cas}$ output. We shall elaborate on such case shortly.

The binary representation of the size of the DIP set ($|I_l| = \{I_l^0, \cdots, I_l^{p-1}\}$), reveals the number and location of the OR gates in the chain. Since $|I_l|$ is always odd, the bit I_l^{p-1} is always equal to 1. There exist only one DIP (DIP_{nc}) that sets all the gates in the entire g_{cas} chain to their non-controlling value, allowing the MSB (first input of g_{cas}) value of '1' (after key operation), to propagate down to g_{cas} output. *Among the bits $\{I_l^0, \cdots, I_l^{p-2}\}$, a '1' is obtained only in positions which corresponds to an OR gate in the chain.* This is because, if the i^{th} position in g_{cas} is an OR gate, the number of DIPs introduced for this OR gate is 2^i (due to all possible value assignments of the previous input bits). Likewise, the set I_l consist of DIPs arising due to all the OR gates in the chain. Hence, a '1' is obtained at the position of the OR gates in the chain (Step 2 of the attack).

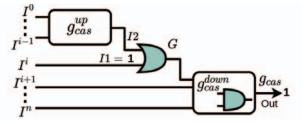

Fig. 4: G is an OR gate in g_{cas}. I^i is the primary input that produces $I1 = 1$ after key operation. $I2$ is the input to G from the upper portion g_{cas}^{up}.

DIP_{nc} sets all the gates in g_{cas} to their non-controlling values. Using the knowledge of the location of AND/OR gates in the chain and the assignment $K1 = 1$, the key gates corresponding to each input bits in g_{cas} can be known. For example, let DIP_{nc}^i input bit enters an AND gate directly (after XOR/XNOR operation). The non-controlling value of AND gate is '1'. Hence, if $DIP_{nc}^i = 1$, the key gate derived for DIP_{nc}^i is XNOR ($K1 = 1$ and $DIP_{nc}^i = 1$ implies $K1 \odot DIP_{nc}^i = 1$, $\odot$ being a XNOR gate), otherwise the derived key gate is a XOR gate (refer lines 11 - 14 in Algo. 1). In this way, all the key gates of g_{cas} can be obtained (as mentioned in Step 3 of the attack). Assigning '0' to XOR gates and '1' to XNOR gates gives a possible correct key of g_{cas} ($K1_c$), since there exists 2^n correct key combinations [2]. Assigning '0' to XOR gates and '1' to XNOR gates reduces them to buffer gates.

Let OR_{last} (Algo. 2) be the input that enters directly to the last OR gate in both the blocks. Further, I_l contains all possible assignments to the input bits ranging from 0 to $OR_{last} - 1$, when OR_{last} is set to controlling value of '1' (after key operation). Thus, the correct key for the key gates of $\overline{g}_{cas}$ in the range 0 to $OR_{last} - 1$ is also present in I_l.

To obtain the complete correct key ($K2_c$) of $\overline{g}_{cas}$, the key information for the remaining bits (i.e., from OR_{last} to $n-1$) of $\overline{g}_{cas}$ is necessary. This information is also leaked from I_l. The DIPs are obtained by setting $K1 = 1$ to g_{cas} and assigning '0' to $\overline{g}_{cas}$ key bits in the range 0 to $OR_{last} - 1$. Then a brute-force search over all possible key assignments for the key bits in the range OR_{last} to $n-1$ is performed

Algorithm 1 Attack Algorithm for extracting g_{cas} keys

1: **Inputs:** An activated chip ($eval$); RE locked netlist; **Outputs:** Nature of the key gates of g_{cas}; Correct Key ($K1_c$) of g_{cas}; Position of the OR gates in CAS-Lock
2: **Initialize:** Miter Circuit with two copies (A and B) of the locked netlist
3: $K1_A \leftarrow 1$; $K2_A \leftarrow 0$; ▷ copy A of Miter Circuit
4: $K1_B \leftarrow 0$; $K2_B \leftarrow 0$; ▷ copy B of Miter Circuit
5: $I_l \leftarrow$ SAT-based extraction of DIPs using Miter circuit ▷ DIP set corresponding to copy A
6: Position of OR gates $\leftarrow$ position of '1's in the binary representation of $|I_l|$ ▷ Gives us the position of the OR gates, remaining gates in the chain are AND
7: **for** each $i \in I_l$ **do**
8: Let $i = \{i_0, \cdots, i_{n-1}\}$ ▷ Binary representation of each I_l element i
9: **if** $\{\overline{i_0}, i_1, \cdots, i_{n-1}\} \notin I_l$ **then** $DIP_{nc} \leftarrow i$; $Break$; ▷ Extracting the non-repeating DIP from I_l
10: ▷ Extracting nature of key gates and $K1_c$ using DIP_{nc} and $K1_A = 1$
11: **for** $i \leftarrow 0$ *to* $(|DIP_{nc}| - 1)$ **do**
12: **if** (i is an input to an AND gate) **then**
13: **if** $(DIP^i_{nc} == 0)$ **then** $K1^i_c \leftarrow 0$ ▷ XOR gate; **else** $K1^i_c \leftarrow 1$ ▷ XNOR gate
else
14: **if** $(DIP^i_{nc} == 0)$ **then** $K1^i_c \leftarrow 1$ ▷ XNOR gate **else** $K1^i_c \leftarrow 0$ ▷ XOR gate
15: **return** $K1_c$ ▷ Correct key of the g_{cas} block

to obtain a DIP set (S in line 6, Algo. 2) corresponding to each key assignment. Whenever the size of set S is less than $|I_l|$ (line 7 in Algo. 2), we *learn* the key information of the aforementioned bits from the corresponding brute-force search key assignment. It is observed that the size of S reduces for a particular key assignment ($K2_{nc}$), of the bits OR_{last} to $n-1$. $K2_{nc}$ assigns the AND gates (after the last OR gate in $\overline{g}_{cas}$) and the terminating NAND gate to '1'. Since, in that case a '1' propagates down the $\overline{g}_{cas}$ to produce '0' at $\overline{g}_{cas}$ output. Thus, certain input patterns that produce a '1' at g_{cas} are no longer DIPs for the assignment $K2_{nc}$. As a result, the size of S is less than I_l. Line 10 in Algo. 2 shows the key gate extraction of the bits OR_{last} to $n-1$ in $\overline{g}_{cas}$ (Step 4 of the attack).

Using divide-and-conquer, we extract the correct key of $\overline{g}_{cas}$. As already mentioned, I_l contains the correct key assignment for bits ranging from 0 to $OR_{last}-1$ of $\overline{g}_{cas}$ corresponding to $K1_c$. Using the SAT-based key verification function provided in [6], the correct key of the CAS-Lock technique (line 14, Algo. 2) is retrieved (completing Step 5 of the attack). The '0's in $K2_c$ corresponds to XOR and '1's to XNOR key gates.

M-CAS is also susceptible to the proposed attack. If the outer CAS-Lock structure is removed harnessing the SPS-based removal attack [9], the remaining locked netlist can be subjected to this attack to retrieve K_{inner} (Fig. 2).

IV. Attack Analysis

The DIP set ($|I_l|$) drives the attack complexity and varies upon the number and location of the OR gates in the cascaded chain of CAS-Lock. In [5] it is shown that the number of DIPs obtained are always odd. However, they did not provide a bound regarding the maximum number of DIPs generated for a particular CAS-Lock chain configuration. *Here we provide a formula that calculates the maximum number of DIPs generated for a specific CAS-Lock chain configuration (Lemma 2).* Let g_{cas} and $\overline{g}_{cas}$ consist of $N-1$ gates each for N input CAS-Lock scheme. Some of these $N-1$ gates are OR gates, positioned randomly in the chain, excluding the terminating position (assuming an AND/NAND terminated CAS-Lock structure). Let C be a set consisting of the inputs directly entering the OR gates in g_{cas}. Without loss of generality, the inputs directly entering OR gates

Algorithm 2 Attack Algorithm for extracting $\overline{g}_{cas}$ keys, hence the CAS-Lock key (K_c)

1: **Inputs:** An activated chip ($eval$); RE locked netlist $N(X, K)$; $K1_c$ of g_{cas}; **Outputs:** Nature of the key gates of $\overline{g}_{cas}$; Correct key ($K2_c$) of $\overline{g}_{cas}$; Correct key of the entire CAS-Lock block (K_c)
2: **Initialize:** Position of the OR gates; Assigning g_{cas} to $K1 = 1$
3: Let $OR_{last} \leftarrow$ position of the last OR gate in the chain ▷ OR_{last} is the input to the last OR gate
4: Assign $\{K2_0, \cdots, K2_{OR_{last}-1}\} \leftarrow 0$
5: **for** $iter \leftarrow 0$ *to* $2^{\{(n-1)-OR_{last})\}}$ **do**
6: Extract the DIP set (S) by assigning $iter$ to $\{K2_{OR_{last}}, \cdots, K2_{n-1}\}$
7: **if** $(|S| < |I_l|)$ **then** $DIP_{K2} \leftarrow$ any DIP from S; $K2_{nc} \leftarrow iter$; $break$
8: ▷ Extracting $\{K2_{OR_{last}}, \cdots, K2_{n-1}\}$ of $K2_c$
9: **for** $j \leftarrow (OR_{last})$ *to* $(n-1)$ **do**
10: **if** $(K2^j_{nc} == DIP^j_{K2})$ **then** $K2^j_c \leftarrow 1$ ▷ XNOR gate; **else** $K2^j_c \leftarrow 0$ ▷ XOR gate
11: ▷ Extracting $\{K2_0, \cdots, K2_{OR_{last}-1}\}$ of $K2_c$
12: **for** each $k \in I_l$ **do**
13: Let $k = \{k_0, \cdots, k_{n-1}\}$ ▷ Binary representation of each I_l element k
14: **if** $(N(X, K1_c)||\{k_0, \cdots, k_{OR_{last}-1}\}||\{K2_c^{OR_{last}}, \cdots, K2_c^{n-1}\}) == eval(X)$ **then**
$K_c \leftarrow (K1_c||\{k_0, \cdots, k_{OR_{last}-1}\}||\{K2_c^{OR_{last}}, \cdots, K2_c^{n-1}\})$; $break$
▷ Using the SAT-based key verification
15: **return** K_c ▷ Correct key of CAS-Lock

are subjected to key operation (XOR/XNOR). $|C|$ is equal to the total number of OR gates in a chain.

Lemma 2. ***The maximum number of DIPs that will be generated for a CAS-Lock configuration is given by Eq. 1:***

$$\#DIPs = 1 + \sum_{i=1}^{|C|} 2^{c_i} \tag{1}$$

where c_i is a distinct set element of C. The summation of 2^{c_i} indicates that once an OR gate receives a '1' as input (controlling value), the other OR gate input in effect becomes don't care. Hence, we obtain those input patterns as DIP, where all possible binary value assignments are set for the inputs in the range $i = 0$ to c_i. The '1' in Eq. 1 corresponds to the non-repeating DIP (DIP_{nc}).

Proof. We prove Lemma 2 using mathematical induction. If no OR gates are present in the chain ($|C| = 0$), CAS-Lock reduces to Anti-SAT technique [2]. Maximum number of DIPs generated in Anti-SAT scheme for any incorrect key is always one. As per the Eq. 1, if C is empty, then # DIPs = 1.

Now, if only one OR gate is present ($|C| = 1$) at the j^{th} position (where $0<j<(n-1)$), the maximum number of DIPs produced according to Eq. 1 are $1+2^j$. When the j^{th} position input is '1' (controlling value for OR gate), for all possible value assignments to the input bits 0 to $(j-1)$ will produce '1' at g_{cas} output. As already discussed in Sec. III-C, the probability of getting a '1' at $\overline{g}_{cas}$ due to its terminating NAND gate is higher than g_{cas}. Half of the entire input space produces a '1' at the $\overline{g}_{cas}$. Hence, the number of DIPs produced are actually determined by the OR gates in g_{cas}.

Let us assume Eq. 1 be true for r number of OR gates in the chain, $C = \{c_1, c_2, \cdots, c_r\}$ and $|C| = r$ (inductive hypothesis), where $1<r<(N-1)$. The maximum number of DIPs produced are:

$$\#DIPs = 1 + \sum_{i=1}^{|C|} 2^{c_i} = 1 + 2^{c_1} + 2^{c_2} + \cdots + 2^{c_r} \tag{2}$$

Now, suppose $(r+1)$ OR gates are present, i.e., another OR gate is added to the chain already consisting of r OR gates. Thus,

TABLE I: Impact of our proposed attack on CAS-Lock ($|K|$ = 32 bits and 64 bits), for various AND-OR structures as shown in the column g_{cas} chain configuration. # DIPs shows number of elements in the DIP set (I_l).

Locked Benchmarks	# I/O	$\|K\|$ = 32 bits		Locked Benchmarks	$\|K\|$ = 64 bits		Structure & Key recovered?
		g_{cas} chain configuration	# DIPs (I_l)		g_{cas} chain configuration	# DIPs (I_l)	
c432	36/7	A-O-2A-O-2A-O-2A-O-2A-O-A	18725	**c2670**	2A-O-2(4A-O)-2(2A-O)-12A	598281	✓
c880	60/26	A-O-2A-O-2A-O-2A-O-2A-O-A	18725	**c5315**	4A-O-3(5A-O)-8A	8521761	✓
c1908	33/25	2A-O-5A-O-2A-2O-2A	12089	**c7552**	2A-O-9A-O-4A-O-3A-O-9A	2367497	✓
c2670	233/140	O-6A-O-5A-O-A	16643	**c5315**	2A-O-2(4A-O)-2(2A-O)-12A	598281	✓
c3540	50/22	2A-O-5A-O-2A-2O-2A	12089	**c2670**	4A-O-3(5A-O)-8A	8521761	✓
c5315	178/123	14A-O	32769	**c7552**	2A-O-2(4A-O)-2(2A-O)-12A	598281	✓
c6288	32/32	3A-2O-3A-2O-3A-O-A	17969	**c2670**	2A-O-9A-O-4A-O-3A-O-9A	2367497	✓
c7552	207/108	3A-2O-3A-2O-3A-O-A	17969	**c5315**	2A-O-9A-O-4A-O-3A-O-9A	2367497	✓

$C' = \{c_1, c_2, \cdots, c_r, c_{r+1}\}$ and $|C'| = r + 1$. The maximum number of DIPs produced are:

$$\#DIPs = 1 + 2^{c_1} + 2^{c_2} + \cdots + 2^{c_r} + 2^{c_{r+1}} = 1 + \sum_{i=1}^{|C'|} 2^{c_i} \quad (3)$$

Thus, we conclude the proof. Eq. 3 illustrates that the maximum number of DIPs that can be generated for a specific CAS-Lock configuration is given by the formula in Lemma 2. □

The brute force complexity required to extract a portion of $\overline{g}_{cas}$ is $2^{\{(n-1)-OR_{last})\}}$ (line 5 in Algo. 2). It depends on OR_{last}. The brute-force search space is usually less than the size of I_l, since the position of the OR gates down the chain helps CAS-Lock achieve higher output corruptibility implying better bypass attack resistance.

V. Experimental Evaluation

The attack was evaluated on a set of circuits belonging to the ISCAS-85 [14] benchmark suite. In the experiments, we have locked the circuits with key length of 32 and 64 bits. Only those circuits of ISCAS-85 having input size greater than 64 bits have been used for locking with a key length of 64 bits. The secure integration of CAS-Lock implementation has been considered, since the random integration variant is susceptible to the SAT-based attack [2]. Various configurations of AND-OR chains has been subjected to this attack, few of which has been listed in the column "g_{cas} chain configuration" of Table I. For example, the configuration "A-O-2A-O-2A-O-2A-O-2A-O-A" should be interpreted as: 'A' represents an AND gate whereas an 'O' represents an OR gate. The aforementioned chain represents the structure of CAS-Lock which begins with an AND gate cascaded with an OR gate followed by a couple of AND gates so on, terminated by an AND gate. Likewise, "14A-O" represents another configuration consisting of a cascaded chain of 14 AND gates terminated by an OR gate. These configurations conform with the heuristics suggested in [5] regarding output corruptibility. The proposed attack (Algo. 1 and Algo. 2) is implemented using Python programming language. The Python extension of *Cryptominisat* [15] has been utilized for finding the DIP set (I_l). Finally, our experiments are performed on an Ubuntu workstation consisting of quad-core Intel i7 - 7700 CPU, executing at 3.6 GHz and 16 GB of main memory. It is evident from Table I that the number of DIPs differ with CAS-Lock structure. Row 6 in Table I also verifies the fact that a cascaded chain consisting of AND gates and terminated by an OR gate at g_{cas} produces the maximum output corruption [5]. The proposed attack is successful in retrieving the correct key and the CAS-Lock chain configurations for all the benchmark circuits by simply observing and analyzing the DIPs.

VI. Conclusion

This paper introduces a novel attack against the state-of-the-art logic locking scheme CAS-Lock. To the best of our knowledge, this is the first work that highlights the vulnerability of "secure" logic locking against information leakage observable externally. We exploit the same to retrieve the correct key and the entire structure of implemented CAS-Lock scheme. An interesting avenue of future work could be to extend this attack and exploit the information leaked by other logic locking techniques. This work once again proves that designing a secure logic locking technique remains an open challenge.

References

[1] J. A. Roy, F. Koushanfar, and I. L. Markov, "Epic: Ending piracy of integrated circuits," in *Proceedings of the conference on Design, automation and test in Europe*, 2008, pp. 1069–1074.

[2] Y. Xie and A. Srivastava, "Anti-sat: Mitigating sat attack on logic locking," *IEEE Trans. CAD*, vol. 38, no. 2, pp. 199–207, 2018.

[3] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1601–1618.

[4] A. Saha, S. Saha, S. Chowdhury, D. Mukhopadhyay, and B. B. Bhattacharya, "Lopher: Sat-hardened logic embedding on block ciphers," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6.

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

[6] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *HOST*. IEEE, 2015, pp. 137–143.

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

[8] X. Xu, B. Shakya, M. Tehranipoor, and D. Forte, "Novel bypass attack and bdd-based tradeoff analysis against all known logic locking attacks," in *International conference on cryptographic hardware and embedded systems*. Springer, 2017, pp. 189–210.

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

[10] D. Sirone and P. Subramanyan, "Functional analysis attacks on logic locking," in *DATE*. IEEE, 2019, pp. 936–939.

[11] A. Sengupta, N. Limaye, and O. Sinanoglu, "Breaking cas-lock and its variants by exploiting structural traces," pp. 1–23, 2021.

[12] A. Sengupta and O. Sinanoglu, "Cas-unlock: Unlocking cas-lock without access to a reverse-engineered netlist." *IACR Cryptol. ePrint Arch.*, p. 1443, 2019.

[13] B. Shakya, X. Xu, M. M. Tehranipoor, and D. Forte, "Defeating cas-unlock." *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 324, 2020.

[14] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator," in *Fortran. ISCAS'85*, 1985.

[15] M. Soos, "Cryptominisat–a SAT solver for cryptographic problems." [Online]. Available: http://www.msoos.org/cryptominisat4, 2009.