An Approach to Unlocking Cyclic Logic Locking - LOOPLock 2.0

Pei-Pei Chen[1], Xiang-Min Yang[2], Yi-Ting Li[2], Yung-Chih Chen[3], and Chun-Yao Wang[2]

[1]Institute of Information Systems and Applications, National Tsing Hua University, Hsinchu, Taiwan, R.O.C.
[2]Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, R.O.C.
[3]Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan, R.O.C.

***Abstract*—Cyclic logic locking is a new type of SAT-resistant techniques in hardware security. Recently, LOOPLock 2.0 was proposed, which is a cyclic logic locking method creating cycles deliberately in the locked circuit to resist SAT Attack, CycSAT, BeSAT, and Removal Attack simultaneously. The key idea of LOOPLock 2.0 is that the resultant circuit is still cyclic no matter the key vector is correct or not. This property refuses attackers and demonstrates its success on defending against attackers. In this paper, we propose an unlocking approach to LOOPLock 2.0 based on structure analysis and SAT solvers. Specifically, we identify and remove non-combinational cycles in the locked circuit before running SAT solvers. The experimental results show that the proposed unlocking approach is promising.**

***Index Terms*—Hardware security, logic unlocking, SAT Attack, CycSAT, BeSAT, LOOPLock 2.0.**

I. Introduction

In today's global Integrated Circuits (ICs) supply chain, design companies may purchase intellectual property (IP) from IP vendors and integrate them into their designs for saving the development effort. To reduce the fabrication cost, they outsource the fabrication to third-party foundries. However, the offshore foundries may be untrusted and pose some threats to IP piracy, counterfeiting, and IC overproduction. Hence, many protection techniques have been proposed recently to deal with the hardware security issues [3] [6] [7] [8] [11] [12] [13] [14] [15] [18] [19] [21] [28] [29] [30] [31] [32].

Logic locking [15] is a useful technique to protect IC designs from potential attackers. Its main idea is to use additional key-controlled gates, key inputs, and an on-chip memory connected to the key inputs to hide the original design. The functionality of the locked IC is correct only when the correct key vector is set in the on-chip memory. As a result, attackers cannot pirate the design directly. However, many unlocking techniques [1] [9] [10] [17] [20] [22] [23] [24] [25] [26] [27] [33] [34] [35] have been proposed to attack different kinds of logic locking methods.

Even though there is a variety of logic locking methods, most of them are vulnerable to the Boolean Satisfiability-based (SAT) Attack [27]. Unlike a brute-force approach, which requires exponential time to find the correct key vector, SAT Attack can effectively and efficiently unlock the traditional logic locking techniques. SAT Attack assumes that attackers can get the locked circuit and a functional correct IC, and it uses SAT solvers to rule out incorrect key vectors iteratively by finding the distinguishing input patterns (DIPs). A DIP is an input pattern that generates different outputs O_a and O_b under two different key vectors K_a and K_b. To find DIPs, SAT Attack constructs a miter-like circuit and then transforms it into a Conjunctive Normal Form (CNF) formula. After finding a DIP, the correct I/O pair obtained from the functionally correct IC will be added into the CNF formula as a constraint for excluding at least one incorrect key vector. When no DIP can be found, the remaining key vectors are considered as the correct key vectors.

To defend against SAT Attack, point-function-based logic locking methods, such as SARLock [31] and Anti-SAT [28] [29], have been proposed. They tried to reduce the number of incorrect key vectors that can be pruned in each iteration to increase the SAT solving time exponentially. However, these methods are vulnerable to approximate attacks [20] [23] [25] and removal attacks [33] [34]. Approximate attacks, e.g., AppSAT [20] and Double DIP [25], target at getting an approximate key vector to the locked circuit. AppSAT terminates the SAT solving process when the error rate is smaller than the threshold set by the attacker. Double DIP uses 2DIP (doubly differentiating input pattern) to rule out two wrong key vectors in each iteration. Removal attacks aim at identifying locking structures and then removing or bypassing them to recover the functionality of the design.

Cyclic logic locking [19] is another method that can resist SAT Attack. It inserts key gates with feedback edges to cyclify the locked circuit and presents observable non-combinational effects in the primary outputs (POs) under incorrect key vectors. For the cyclic logic locking approaches, SAT Attack cannot obtain the correct key vectors in two situations. The first situration is called *statefulness*, where the locked circuit may have different outputs with a fixed key vector and a fixed input pattern. If this input pattern is found as a DIP by the SAT solver, the incorrect key vector causing the statefulness cannot be pruned. Hence, the SAT solver will keep finding the same DIP and cannot be terminated. The other situation is called *oscillation*, where the values of some POs in the

This work is supported in part by the Ministry of Science and Technology of Taiwan under MOST 109-2221-E-007-082-MY2, MOST 109-2221-E-155-047-MY2, MOST 110-2224-E-007-007, MOST 111-2218-E-007-010, MOST 111-2221-E-007-121, and MOST 111-2221-E-011-137-MY3

ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA © 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9217-4/22/10…$15.00
https://doi.org/10.1145/3508352.3549461

locked circuit oscillate between 0 and 1 with incorrect key vectors. The SAT solver may not detect the oscillation such that a wrong key vector is returned. The root cause of these two situations is non-combinational cycles.

However, CycSAT [35] has been shown to decrypt cyclic logic locking successfully. CycSAT first pre-analyzes the locked netlist to find the non-cyclic (NC) condition and then adds the condition to the CNF formula before running the SAT Attack. There are two types of CycSAT, CycSAT-I and CycSAT-II, using different constraints to break cycles. CycSAT-I assumes that the original circuit is acyclic, and the NC condition rules out key vectors that make the locked circuit structurally cyclic. CycSAT-II computes the NC condition to break sensitizable cycles and allows the existence of combinational cycles.

When CycSAT captures all the cycles, it can obtain the correct key vector. However, some cycles may be missed during analysis in CycSAT. Behavioral SAT-based attack (BeSAT) [22] extends CycSAT to overcome this drawback. To deal with statefulness, BeSAT records the found DIP in each iteration and checks whether this DIP is a repeated one or not. If it is a repeated DIP, BeSAT will use this DIP to find the wrong key vector causing statefulness and then adds a constraint to ban this key vector. After the DIP generation process, it uses a ternary-based SAT method to test the remaining key vectors and finds out the correct key vector without causing oscillation.

After that, some CycSAT-resistant techniques were also proposed. SRCLock [13] [14] introduces feedbacks that enlarge the number of cycles to degrade the performance of the pre-processing step in CycSAT and BeSAT. A. Rezaei et al. then proposed two cyclic locking methods [11] [12]. The first method [11] creates hard cycles in the circuit, which is a structure that makes attacks miss cycles while traversing nodes. The second method [12] defends against CycSAT using unreachable states. There exist non-combinational cycles in the locked circuit, but the non-combinational behavior occurs only in the unreachable states. As a result, when CycSAT adds constraints to exclude key vectors with non-combinational cycles, it will prune out the correct key vector though.

LOOPLock 2.0 [30] was recently proposed by Yang et al., which is an improved version of LOOPLock [6]. There are two similar structures called *Type-I cycle pair* and *Type-II cycle pair* in the locked circuit, which defend against SAT Attack, CycSAT, BeSAT, and Removal Attack simultaneously. Each cycle pair embeds a non-combinational cycle and a combinational cycle. When the correct key vector is fed, the resultant circuit is still structurally cyclic but behaves combinationally as the original circuit.

In this paper, we analyze the structures of LOOPLock 2.0 and propose an unlocking approach. The proposed approach breaks the non-combinational cycles in both Type-I and Type-II cycle pairs during the pre-analysis process, then it obtains a correct key vector by running SAT solving. The experimental results also demonstrate that the proposed approach is effective to unlock LOOPLock 2.0.

The rest of this paper is organized as follows. Section II introduces the techniques for generating combinational cycles in the circuits in the prior works. We discuss the security issue about LOOPLock 2.0 and propose our unlocking approach in Section III. The experimental results are shown in Section IV. Finally, Section V concludes this work.

II. Preliminaries

A. NM-based cycle generation

[4] [5] proposed a Node Merging (NM) approach, which merges a target node n_t with a substitute node n_s without changing the circuit's functionality for circuit optimization. In [4] [5], the substitute nodes in the fanout cone of n_t are not allowed to merge n_t. The reason is that cycles will be created by merging and may cause non-combinational effects in a combinational circuit. NM-based cycle generation [2] is a technique to find *cyclic substitute nodes* (CSNs) for n_t, which forms combinational cycles only. *Theorem 1* was proposed in [2] for describing the requirement of forming such combinational cycles.

Theorem 1 [2]: Let n_t denote a target node and n_s denote a substitute node in the transitive fanout cone of n_t. Replacing n_t with n_s forms a set of cycles C. If the value changes on n_t are never propagated to n_s, which means all the side inputs of C do not have input-noncontrolling values simultaneously, C is combinational.

According to *Theorem 1*, we need to check that the value change on n_t is not propagated to n_s under each input pattern. If this is the case, the n_s is a CSN to n_t, and the formed cycle is combinational. However, the process for finding all CSNs is computation-intensive. To find candidate CSNs efficiently, *Condition 1* was proposed in [2]. After finding a candidate CSN, a SAT-based algorithm is then used to verify whether it is a CSN or not.

Condition 1 [2]: Let n_t denote a target node, and n_s denote a substitute node in the transitive fanout cone of n_t. Replacing n_t with n_s forms a set of cycles C. If $n_s = 1$ and $n_t = D$ are MAs for the stuck-at 0 fault test on n_t, and $n_s = 0$ and $n_t = \overline{D}$ are MAs for the stuck-at 1 fault test on n_t, n_s is a candidate CSN.

The MAs, abbreviation for *mandatory assignments*, for a stuck-at 1 (0) fault test on a node n in *Condition 1* are the unique value assignments to detect this fault. A set of MAs includes the value assignments to activate the fault effect and the value assignments to propagate the fault effect. Using logic implications on these assignments forward or backward can derive more MAs. The D and $\overline{D}$ symbols are to model the stuck-at fault effects. D ($\overline{D}$) represents the value of 1/0 (0/1), where 1 (0) is the fault-free value, and 0 (1) is the faulty value.

B. LOOPLock

LOOPLock [6] is a cyclic logic locking method using NM-based techniques [2] [4] [5]. Two locking structures, Type-I and Type-II cycle pairs, are used to protect the design. Each cycle pair deliberately contains a combinational cycle and a non-combinational cycle.

According to *Theorem 1*, C is combinational as the value changes are never propagated from n_t to n_s. In other words, a *blocking node* n_b exists between n_t and n_s, which blocks the

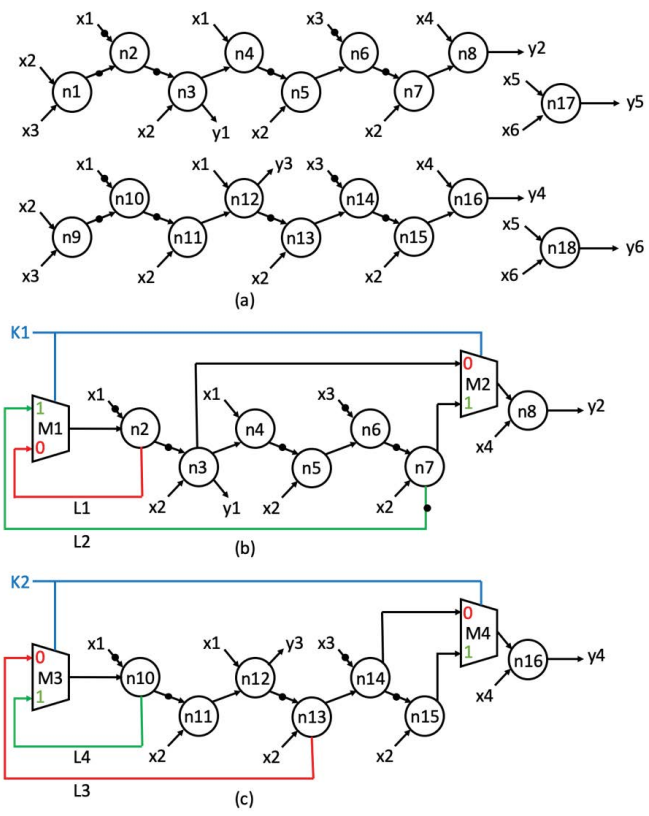

Figure 1. An example for LOOPLock. (a) The original circuit. (b) The Type-I cycle pair. (c) The Type-II cycle pair.

effect of the value changes on n_t. Hence, replacing n_t with a node after n_b can form a combinational cycle. Replacing n_t with a node between n_t and n_b can form a non-combinational cycle though. n_b can be identified by examining the fault effect propagation from n_t.

We use Fig. 1 to illustrate the locking structures of LOOPLock. Fig. 1(a) is a circuit represented in an And-Inverter Graph (AIG). Nodes $n1 \sim n18$ are two-input AND gates, and the dots on the edges are inverters. $x1 \sim x6$ are primary inputs (PIs), and $y1 \sim y6$ are POs.

The Type-I cycle pair is shown in Fig. 1(b). Two key gates $M1$ and $M2$, which are MUXes, are controlled by the same key input $K1$. The correct key value is $K1 = 1$. $M1$ connects to two cycles, where the red one is a non-combinational cycle $L1$, which affects PO $y1$ while $K1 = 0$, and the green one is a functionally correct combinational cycle $L2$ while $K1 = 1$. Let us explain the construction of the Type-I cycle pair. Let $n1$ be the n_t in the original circuit. We can use the NM techniques and fault effect propagation to identify CSN and n_b, which are $n7$ and $n4$, respectively. Then, we can create $M1$ with $L1$ and $L2$ using the obtained information. $M2$ is used to confuse attackers such that the Type-I cycle pair is very similar to the Type-II cycle pair. Since there exists at least one path from a node between n_t and n_b to any PO in the Type-I cycle pair, the Type-I cycle pair can defend against SAT Attack due to the observable non-combinational effect.

The Type-II cycle pair is shown in Fig. 1(c). The key input $K2$ also controls two key gates $M3$ and $M4$, and the correct key value of $K2$ is 1. The red cycle $L3$ connected to $M3$ is combinational while $K2 = 0$, and the green cycle $L4$ is non-combinational while $K2 = 1$. Let us explain the construction of the Type-II cycle pair. Let $n9$ be the n_t in the original circuit. Different from the Type-I cycle pair, to create $M3$, we find $n12$ as n_b and ensure that there exists no path from any node between n_t and n_b to any PO. Hence, choosing either $L3$ or $L4$ does not affect the functionality of the circuit. However, $K2 = 0$ is a wrong key value due to choosing the wrong path for $M4$. The purpose of the Type-II cycle pair is to invalidate CycSAT and BeSAT. While constructing the NC condition in the pre-processing step of CycSAT or BeSAT, the non-combinational cycles will be ruled out. Therefore, the correct key vector cannot be obtained by the attackers.

The target nodes have been removed in the locked circuit when constructing the Type-I and Type-II cycle pairs. Thus, even if Removal Attack can identify the locking structures, restoring the functionality of the original circuit is still a challenging task due to the absence of target nodes.

C. LOOPLock 2.0

LOOPLock is an effective cyclic logic locking method, which invalidates SAT Attack, CycSAT, BeSAT, and Removal Attack. However, the authors in [30] proposed an attacking method to unlock LOOPLock. The main idea is to distinguish the Type-I and Type-II cycle pairs by identifying the structural difference. That is, there exists at least a path from a node between n_t and n_b to any PO in a Type-I cycle pair, but no such path exists in a Type-II cycle pair. For ease of discussion, the MUX with two feedback edges in a cycle pair, which is located at the left side, is denoted as pre-MUX, and the other one controlled by the same key input, which is located at the right side, is denoted as post-MUX.

The first step of the unlocking method is to find the positions of n_t and n_b. Since n_t has been replaced by the pre-MUX, its position can be recognized easily. Then n_b is identified by a *Blocking Node Identification* method. After finding the positions of n_t and n_b, the locking structure can be distinguished as a Type-I or a Type-II cycle pair by finding any PO existing between n_t and n_b.

The authors in [30] then proposed LOOPLock 2.0, which is an enhanced version of LOOPLock. Its main idea is to hide the structural difference between the Type-I and Type-II cycle pairs. We use Fig. 2 to introduce the structures of LOOPLock 2.0. The Type-I cycle pair is shown in Fig. 2(a). A node $n17$ in the original circuit of Fig. 1(a) is selected to insert a key gate $M5$, which hides the connection between $n3$ and $y1$. On the other hand, the Type-II cycle pair, as shown in Fig. 2(b), also needs to be modified. A PO $y6$ and its fanin node $n18$ are first selected. Then $M6$ is inserted with a wrong path connecting to the node $n11$, which is between the pre-MUX and n_b. The structures of Type-I and Type-II cycle pairs in LOOPLock 2.0 are very similar to each other and cannot be distinguished by the unlocking method against LOOPLock.

Furthermore, the authors in [30] proposed the *Subcircuit Duplication* method for increasing the number of Type-II cycle pairs in a circuit. This is because the target node for a Type-II

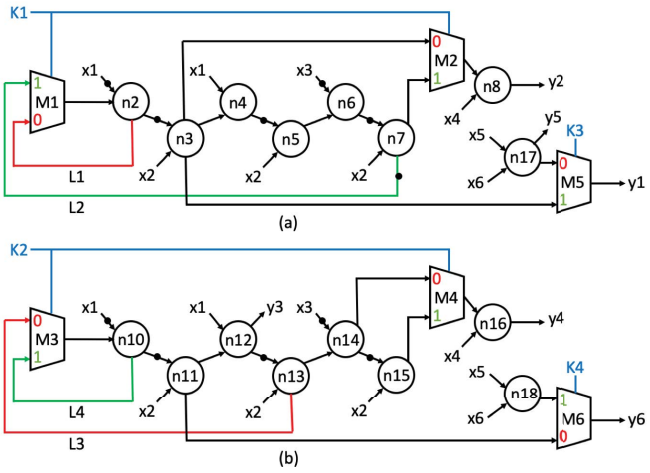

Figure 2. An example for LOOPLock 2.0. (a) The Type-I cycle pair. (b) The Type-II cycle pair.

cycle pair has to be a redundant node in the original circuit, which may be rare in practice. To overcome this difficulty, the *Subcircuit Duplication* method duplicates the nodes between n_t and n_b if necessary, and transfers the connections of their fanouts that have paths to POs from the original nodes to the duplicated nodes. Then the original n_t becomes redundant and can be used to create a Type-II cycle pair.

III. Our Approach

In Section II-C, we have introduced the locking structures in LOOPLock 2.0 and explained why they can effectively defend against different kinds of attacking methods. However, LOOPLock 2.0 still has some security concerns after analyzing the locking structures. We will discuss these security concerns and then propose an unlocking approach against LOOPLock 2.0 in this section.

A. Shortcomings of LOOPLock 2.0

In this subsection, we discuss three shortcomings of LOOPLock 2.0 that affect the security of the locked circuit.

1) The post-MUX in the Type-I cycle pair lowers the encryption strength.
2) The shared key input in the Type-II cycle pair is used to defend against CycSAT and BeSAT. However, the locking structure may malfunction while the shared key input is split into two individual key inputs.
3) The positions of the blocking nodes n_b is revealed after a comprehensive examination, which means that the non-combinational cycles and combinational ones can be distinguished.

For the first shortcoming, in fact, having the pre-MUX only in the Type-I cycle pair is enough to invalidate SAT Attack. The post-MUX in the Type-I cycle pair is used to adjust its structure such that the structure of the Type-I cycle pair is more similar to that of the Type-II cycle pair. However, inserting the post-MUX in the Type-I cycle pair may create a weakness in the locked circuit when facing SAT Attack. We use an example to demonstrate this situation. In Fig. 2(a), the wrong path of the post-MUX $M2$ is connected to $n3$, which is a suitable node for trapping SAT Attack into an infinite loop. Since the nodes for creating wrong paths of the post-MUXes are randomly selected in LOOPLock 2.0, when the wrong path of the post-MUX $M2$ is connected to $n2$ instead of $n3$ in this example, the DIP $(x1, x2, x3, x4) = (1, 1, 0, 1)$ can be found. This DIP blocks the non-combinational effect and causes different values on $y2$ under different key vectors. Thus, $K1 = 0$ will be pruned and the attack is successful.

For the second shortcoming, we can first modify the locked circuit by replacing the shared key input Kn in each cycle pair with two key inputs Kn_1 and Kn_2 controlling the pre-MUX and post-MUX, respectively. Next, we apply CycSAT-II, which allows having combinational cycles in the circuit, on the modified circuit to get a correct key vector. We use the example in Fig. 2 to explain the reason. The Type-I cycle pair is vulnerable to CycSAT-II because the correct key value selects the combinational cycle. On the other hand, the Type-II cycle pair invalidates CycSAT-II because the correct key value selecting the non-combinational cycle from the pre-MUX and the correct path from the post-MUX will be ruled out while constructing the NC condition. If $K2$ in Fig. 2(b) is replaced by $K2_1$ and $K2_2$, which control $M3$ and $M4$, respectively, $(K2_1, K2_2, K4) = (1, 1, 1)$ and $(0, 1, 1)$ are both correct key vectors to the modified circuit. The key vector $(0, 1, 1)$ can be solved by CycSAT-II, which allows having combinational cycles in the circuit. Hence, using shared key inputs to defend against CycSAT-II is not secure enough due to the method of key-splitting.

For the last shortcoming, the enhanced structures in LOOPLock 2.0 only make two cycle pairs similar, but do not hide the positions of the blocking nodes n_b. Thus, we still can use this information to unlock LOOPLock 2.0. The detailed process of our unlocking approach will be presented in Section III-B.

B. Our Unlocking Approach

In the previous sections, we know that LOOPLock 2.0 can invalidate SAT Attack due to the observable non-combinational effects at POs under incorrect key vectors. If we can remove these non-combinational cycles, the resultant circuit will be vulnerable to SAT Attack.

The proposed unlocking approach consists of two steps: preprocessing step and SAT solving step. *The preprocessing step* analyzes the locked circuit and modifies the cyclic structures. *The SAT solving step* applies SAT solvers. First, we identify the non-combinational cycle in each cycle pair. According to the last shortcoming of LOOPLock 2.0, we can distinguish the non-combinational and combinational cycles when the position of the blocking node n_b is identified. We propagate the fault effects from the pre-MUX, which is at the same position as the removed target node n_t. To avoid cycles, we do not propagate the fault effects through the feedback paths to the pre-MUX. As a result, the node that can block the fault effects is n_b. We then either identify a cycle as a non-combinational cycle when the feedback path is from a

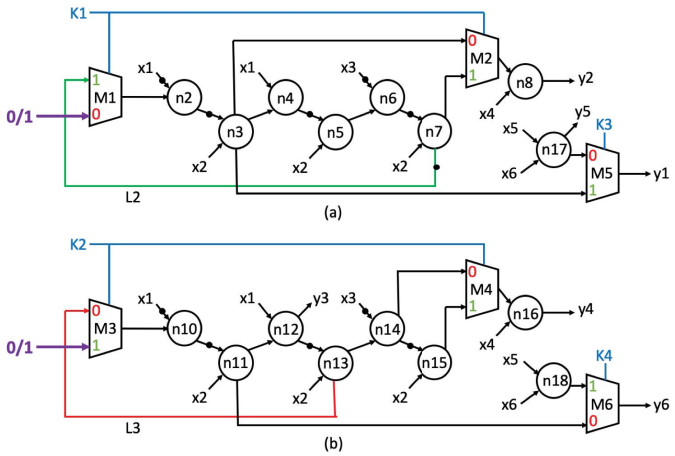

Figure 3. The modified circuit after removing non-combinational cycles. (a) The Type-I cycle pair without the non-combinational cycle. (b) The Type-II cycle pair without the non-combinational cycle.

node between the pre-MUX and n_b, or identify a cycle as a combinational cycle when the feedback path is from a node in the fanout cone of n_b.

Next, we replace the feedback paths of the non-combinational cycles with an arbitrary constant value (0 or 1) to break these cycles, which may either cause statefulness or oscillation. In the Type-I cycle pair, the correct key vector will select the combinational cycle instead of the non-combinational one. On the other hand, the non-combinational cycle in the Type-II cycle pair is unobservable at POs under the correct key vector. Hence, replacing the feedback paths of the non-combinational cycles with either 0 or 1 will not change the functionality of the circuit under the correct key vector. An example of the modified circuit with the constant replacement for Fig. 2 is shown in Fig. 3. The feedback paths of $L1$ and $L4$ are replaced by constant values. As a result, there is no non-combinational cycle in the modified circuit.

In the SAT solving step, we use the modified circuit to build a miter-like circuit and apply SAT solvers to have the correct key vector as SAT Attack. Algorithm 1 shows the pseudo-code of the proposed unlocking approach.

Algorithm 1 Our Unlocking Approach

Input: A locked netlist $C_e(x,k)$ and an activated IC $f(x)$.
Output: The correct key vector $k*$.

```
for each key input k_i in C_e do        // preprocessing step
    Find the cycle pair CP with k_i;
    Find the pre-MUX M_pre in CP;
    Propagate the fault effects D and D̄ from M_pre;
    Find the blocking node n_b;
    for each feedback path L_i connecting to M_pre do
        if L_i forms a non-combinational cycle then
            replace L_i with constant 0 or 1;
        end if
    end for
end for
k* = SAT-ATTACK(C_e(x,k));
return k*;
```

C. Cycle Groups in LOOPLock 2.0

In LOOPLock 2.0, each cycle pair only contains a combinational cycle and a non-combinational cycle. However, when the locking structures are enhanced with more cycles, they are still vulnerable to our unlocking approach since our unlocking approach is based on removing all the non-combinational cycles. We illustrate the *cycle group* structure, which contains more cycles than a cycle pair, in the example of Fig. 4 for explaining the strength of our unlocking approach. In Fig. 4(a), there are two non-combinational cycles ($L1$ and $L2$) and two combinational cycles ($L3$ and $L4$) in the Type-I cycle group. Similarly, there are two non-combinational cycles ($L5$ and $L6$) and two combinational cycles ($L7$ and $L8$) in the Type-II cycle group in Fig. 4(b). The green cycles ($L4$ and $L5$) will be selected under the correct key vector.

When applying the proposed unlocking approach on the locked circuit with cycle groups, the blocking nodes $n4$, $n12$, and the non-combinational cycles are identified. Then the feedback paths of non-combinational cycles $L1$, $L2$, $L5$, and $L6$, which are inputs of the pre-MUXes, are replaced by arbitrary constant values, either 0 or 1. Finally, we can obtain the correct key vector by applying the SAT solver to the modified circuit. This example shows that our unlocking approach works successfully even when the number of cycles increases in LOOPLock 2.0 for elevating the security level.

IV. Experimental Results

We implemented the proposed unlocking approach against LOOPLock 2.0 in C++ language. The experiment was conducted on an Intel Core i3-8100 3.60GHz Ubuntu 16.04 platform with 5.2GBytes memory. The benchmarks are IWLS 2005 suite [36], and are represented in AIG format. These benchmarks are the locked circuits by LOOPLock 2.0 and shared by the authors of [30]. The experiment is to apply the unlocking approach on these locked circuits. We compared our results with that obtained by SAT Attack, CycSAT, and BeSAT. SAT Attack, CycSAT are available publicly, but for BeSAT, we re-implemented it to prune out key vectors that

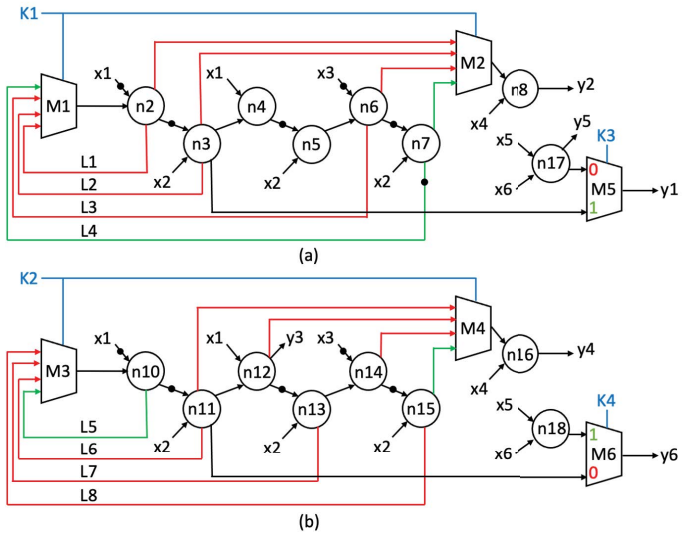

Figure 4. An example for LOOPLock 2.0 with a cycle group. (a) The Type-I cycle group. (b) The Type-II cycle group.

Table I
THE COMPARISON AMONG THE PROPOSED UNLOCKING APPROACH AGAINST SAT ATTACK, CYCSAT, AND BESAT ON THE CIRCUITS LOCKED BY LOOPLOCK 2.0.

Benchmark Information			**Ours**				**SAT Attack**		**CycSAT**		**BeSAT**	
Benchmark	|Type-I|	|Type-II|	Preprocessing	SAT solving	Time (s)	Result	Time (s)	Result	Time (s)	Result	Time (s)	Result
b20	1	1	0.063	2.479	2.542	Correct Key	Inf. loop	No Result	1.978	Wrong Key	2.193	Wrong Key
b21	1	1	0.074	3.789	3.863	Correct Key	Inf. loop	No Result	4.177	Wrong Key	2.175	Wrong Key
b22	1	1	0.170	5.837	6.007	Correct Key	Inf. loop	No Result	3.682	Correct Key	3.707	Correct Key
C1908	1	1	0.007	0.049	0.056	Correct Key	Inf. loop	No Result	0.032	No Result	0.054	No Result
C432	1	1	0.005	0.026	0.031	Correct Key	Inf. loop	No Result	0.035	Correct Key	0.053	Correct Key
i10	1	1	2.153	0.191	2.344	Correct Key	Inf. loop	No Result	0.226	Correct Key	0.269	Correct Key
i2c	1	1	0.007	0.116	0.123	Correct Key	Inf. loop	No Result	0.139	No Result	0.159	No Result
pci_brdge32	1	1	1.355	109.136	110.491	Correct Key	Inf. loop	No Result	1.512	No Result	1.996	No Result
rot	1	1	0.007	0.072	0.079	Correct Key	Inf. loop	No Result	0.052	No Result	0.076	No Result
sasc	1	1	0.005	0.072	0.077	Correct Key	0.055	No Result	0.052	No Result	0.045	No Result
systemcaes	1	1	0.107	22.143	22.250	Correct Key	Inf. loop	No Result	20.314	Wrong Key	21.022	Wrong Key
wb_conmax	1	1	2.396	42.027	44.423	Correct Key	Inf. loop	No Result	39.548	Wrong Key	39.109	Wrong Key

cause statefulness. The comparison of results are summarized in Table I. Columns 1∼3 list the information of benchmarks. Each circuit is locked with only one Type-I cycle pair and one Type-II cycle pair. This is because LOOPLock 2.0 claimed that only one Type-I and Type-II cycle pair can defend against the attacks. Columns 4 and 5 show the corresponding CPU time of preprocessing step (including identifying and removing non-combinational cycles) and SAT solving step in the unlocking approach. For most benchmarks, our approach spent less CPU time for the preprocessing than the SAT solving. However, if the node count in a cycle is large, the preprocessing required more time on identifying the position of n_b, e.g., benchmark $i10$. Columns 6 and 7 show the total CPU time and the results after applying the unlocking approach. The results show that the proposed unlocking approach can remove all the non-combinational cycles in the locked circuits, and then obtain the correct key vectors. Columns 8∼13 show the CPU time and the results after applying SAT Attack, CycSAT, and BeSAT, respectively. SAT Attack was trapped into an infinite loop (Inf. loop) in most benchmarks and returned "No Result" due to the observable non-combinational effect at POs. For CycSAT and BeSAT, the constructed NC conditions ruled out the correct key vectors and returned wrong key vectors or "No Result" for most benchmarks. The results show that the CycSAT and BeSAT were not trapped into an infinite loop but returned "No Result" for some benchmarks. There are two possible reasons. One is that there exists a contradiction in the constructed CNF formula with NC conditions so that no key vector can satisfy the CNF formula. The other is that the correct key vector was excluded by the NC conditions, and the remaining wrong key vectors were pruned out during the DIP generation process. For certain benchmarks, CycSAT and BeSAT also returned correct keys. This is because the circuit is locked with only one Type-I and Type-II cycle pair. This is just a lucky outcome. According to Table I, we can see that the proposed unlocking approach is effective to unlock LOOPLock 2.0.

V. CONCLUSION

In this work, we propose a SAT-based unlocking approach to attack LOOPLock 2.0, which is the state-of-the-art cyclic logic locking method. The experimental results show the effectiveness of the proposed unlocking approach attacking the state-of-the-art, LOOPLock 2.0.

REFERENCES

[1] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "SMT Attack: Next Generation Attack on Obfuscated Circuits with Capabilities and Performance Beyond the SAT Attacks," *IACR Trans. on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 1, pp. 97-122, 2019.

[2] J.-H. Chen, Y.-C. Chen, W.-C. Weng, C.-Y. Huang and C.-Y. Wang, "Synthesis and Verification of Cyclic Combinational Circuits," in *Proc. of SOCC*, pp. 257-262, 2015.

[3] R. Chen and H. Zhou, "Statistical Timing Verification for Transparently Latched Circuits," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 9, pp. 1847-1855, 2006.

[4] Y.-C. Chen and C.-Y. Wang, "Fast Detection of Node Mergers using Logic Implications," in *Proc. of ICCAD*, pp. 785-788, 2009.

[5] Y.-C. Chen and C.-Y. Wang, "Fast Node Merging with Don't Cares Using Logic Implications," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 11, pp. 1827-1832, 2010.

[6] H.-Y. Chiang, Y.-C. Chen, D.-X. Ji, X.-M. Yang, C.-C. Lin and C.-Y. Wang, "LOOPLock: LOgic OPtimization based Cyclic Logic Locking," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2178-2191, 2020.

[7] D.-X. Ji, H.-Y. Chiang, C.-C. Lin, C.-C. Wu, Y.-C. Chen and C.-Y. Wang, "A Glitch Key-Gate for Logic Locking," in *Proc. of SOCC*, pp. 74-79, 2019.

[8] H. M. Kamali, K. Z. Azar, H. Homayoun and A. Sasan, "Full-Lock: Hard Distributions of SAT instances for Obfuscating Circuits using Fully Configurable Logic and Routing Blocks," in *Proc. of DAC*, pp. 1-6, 2019.

[9] L. Li and A. Orailoglu, "Piercing Logic Locking Keys through Redundancy Identification," in *Proc. of DATE*, pp. 540-545, 2019.

[10] S. Potluri, A. Aysu and A. Kumar, "SeqL: Secure Scan-Locking for IP Protection," in *Proc. of ISQED*, pp. 7-13, 2020.

[11] A. Rezaei, Y. Shen, S. Kong, J. Gu and H. Zhou, "Cyclic Locking and Memristor-based Obfuscation Against CycSAT and Inside Foundry Attacks," in *Proc. of DATE*, pp. 85-90, 2018.

[12] A. Rezaei, Y. Li, Y. Shen, S. Kong and H. Zhou, "CycSAT-Unresolvable Cyclic Logic Encryption Using Unreachable States," in *Proc. of ASPDAC*, pp. 358-363, 2019.

[13] S. Roshanisefat, H. M. Kamali and A. Sasan, "SRClock: SAT-Resistant Cyclic Logic Locking for Protecting the Hardware," in *Proc. of GLSVLSI*, pp. 153-158, 2018.

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

[15] J. A. Roy, F. Koushanfar and I. L. Markov, "Ending Piracy of Integrated Circuits," *Computer*, vol. 43, no. 10, pp. 30-38, 2010.

[16] J. P. Roth, W. G. Bouricius and P. R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits," *IEEE Trans. on Electronic Computers*, vol. EC-16, no. 5, pp. 567-580, 1967.

[17] A. Saha, S. Saha, S. Chowdhury, D. Mukhopadhyay and B. B. Bhattacharya, "LoPher: SAT-Hardened Logic Embedding on Block Ciphers," in *Proc. of DAC*, pp. 1-6, 2020.

[18] B. Shakya, X. Xu, M. Tehranipoor and D. Forte, "CAS-Lock: A Security-Corruptibility Trade-off Resilient Logic Locking Scheme," in *IACR Trans. on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 1, pp. 175–202, 2019.

[19] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan and Y. Jin, "Cyclic Obfuscation for Creating SAT-Unresolvable Circuits," in *Proc. of GLSVLSI*, pp. 173-178, 2017.

[20] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan and Y. Jin, "AppSAT: Approximately Deobfuscating Integrated Circuits," in *Proc. of HOST*, pp. 95-100, 2017.

[21] K. Shamsi, D. Z. Pan and Y. Jin, "IcySAT: Improved SAT-based Attacks on Cyclic Locked Circuits," in *Proc. of ICCAD*, pp. 1-7, 2019.

[22] Y. Shen, Y. Li, A. Rezaei, S. Kong, D. Dlott and H. Zhou, "BeSAT: Behavioral SAT-based Attack on Cyclic Logic Encryption," in *Proc. of ASPDAC*, pp. 657-662, 2019.

[23] Y. Shen, A. Rezaei and H. Zhou, "A Comparative Investigation of Approximate Attacks on Logic Encryptions," in *Proc. of ASPDAC*, pp. 271-276, 2018.

[24] Y. Shen, A. Rezaei and H. Zhou, "SAT-based bit-flipping attack on logic encryptions," in *Proc. of DATE*, pp. 629-632, 2018.

[25] Y. Shen and H. Zhou, "Double DIP: Re-Evaluating Security of Logic Encryption Algorithms," in *Proc. of GLSVLSI*, pp. 179-184, 2018.

[26] Y. Shen, Y. Li, S. Kong, A. Rezaei and H. Zhou, "SigAttack: New High-level SAT-based Attack on Logic Encryptions," in *Proc. of DATE*, pp. 940-943, 2019.

[27] P. Subramanyan, S. Ray and S. Malik, "Evaluating the Security of Logic Encryption Algorithms," in *Proc. of HOST*, pp. 137-143, 2015.

[28] Y. Xie and A. Srivastava, "Mitigating SAT Attack on Logic Locking," in *Proc. of International Conference on Cryptographic Hardware and Embedded Systems, pp. 127-146*, 2016.

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

[30] X.-M. Yang, P.-P. Chen, H.-Y. Chiang, C.-C. Lin, Y.-C. Chen and C.-Y. Wang, "LOOPLock 2.0: An Enhanced Cyclic Logic Locking Approach," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 1, pp. 29-34, 2022.

[31] M. Yasin, B. Mazumdar, J. Rajendran and O. Sinanoglu, "SARlock: SAT Attack Resistant Logic Locking," in *Proc. of HOST*, pp. 236-241, 2016.

[32] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran and O. Sinanoglu, "Provably-Secure Logic Locking: From Theory To Practice," in *Proc. of CCS*, pp. 1601-1618, 2017.

[33] M. Yasin, B. Mazumdar, O. Sinanoglu and J. Rajendran, "Security Analysis of Anti-SAT," in *Proc. of ASPDAC*, pp. 342-347, 2016.

[34] M. Yasin, B. Mazumdar, O. Sinanoglu and J. Rajendran, "Removal Attacks on Logic Locking and Camouflaging Techniques," *IEEE Trans. on Emerging Topics in Computing*, vol. 8, no. 2, pp. 517-532, 2020.

[35] H. Zhou, R. Jiang and S. Kong, "CycSAT: SAT-Based Attack on Cyclic Logic Encryptions," in *Proc. of ICCAD*, pp. 49-56, 2017.

[36] IWLS2005 Benchmarks. [Online]. Available: http://iwls.org/iwls2005/benchmarks.html

[37] Berkeley Logic Synthesis and Verification Group, "ABC: a system for sequential synthesis and verification," Available: https://people.eecs.berkeley.edu/∼alanmi/abc/.