# LOOPLock 2.0: An Enhanced Cyclic Logic Locking Approach

Xiang-Min Yang, Pei-Pei Chen, Hsiao-Yu Chiang, Chia-Chun Lin, Yung-Chih Chen, *Member, IEEE*, and Chun-Yao Wang, *Member, IEEE*

***Abstract*—LOOPLock is the state-of-the-art cyclic logic locking method in hardware security. LOOPLock is able to invalidate SAT Attack, Removal Attack, and CycSAT simultaneously by introducing two types of cycle pairs in a circuit. In this work, we analyze LOOPLock's locking mechanism and propose an attacking approach based on locking structure analysis. Furthermore, to defend the new attack, we propose LOOPLock 2.0, which strengthens the original cyclic logic locking method—LOOPLock. Experimental results show the efficiency and effectiveness of the proposed attacking approach to LOOPLock and the high defense capability of LOOPLock 2.0.**

***Index Terms*—Cyclic logic locking, CycSAT, hardware security, logic decryption, LOOPLock, SAT attack.**

## I. Introduction

The GLOBALIZATION of IC design and manufacturing flow brings many benefits to the companies in the semiconductor supply chain. However, if there exists an untrusted agent in the supply chain, the companies would face some threats, such as IP/IC piracy, overproduction, or other unauthorized usages. To protect designs from these threats, many various hardware security techniques were proposed recently [1], [2], [6], [7], [9]–[17], [19], [21], [22], [24], [25], [27], [29], [33]. Logic locking [17] is one of the effective protection methods among those hardware security techniques. The main idea of logic locking is to insert some extra key gates with key inputs. In this way, for those unauthorized users, unknowing the correct key vector means that they cannot activate the IC correctly.

However, these traditional logic locking methods are on the back foot while facing the SAT Attack [26]. It aims to find the distinguishing input patterns (DIPs) by comparing the outputs between the original circuit and the locked one. Then it uses these DIPs to rule out incorrect key vectors.

Manuscript received August 20, 2020; revised November 24, 2020; accepted January 10, 2021. Date of publication January 25, 2021; date of current version December 23, 2021. This work was supported in part by the Ministry of Science and Technology of Taiwan under Grant MOST 108-2218-E-007-061, Grant MOST 109-2221-E-007-082-MY2, Grant MOST 109-2221-E-155-047-MY2, and Grant MOST 109-2224-E-007-005. This article was recommended by Associate Editor S. Ghosh. *(Corresponding author: Chia-Chun Lin.)*

Xiang-Min Yang, Pei-Pei Chen, Hsiao-Yu Chiang, Chia-Chun Lin, and Chun-Yao Wang are with the Department of Computer Science, National Tsing Hua University, Hsinchu 30013, Taiwan (e-mail: yhm19930125@gmail.com; ling8835@gmail.com; johnny19941007@gmail.com; chiachunlin@gapp.nthu.edu.tw; wcyao@cs.nthu.edu.tw).

Yung-Chih Chen is with the Department of Computer Science and Engineering, Yuan Ze University, Taoyuan 32003, Taiwan (e-mail: ycchen.cse@saturn.yzu.edu.tw).

Digital Object Identifier 10.1109/TCAD.2021.3053912

After the SAT Attack, some SAT-resistant methods were proposed [6], [8], [13]–[16], [20], [21], [27]–[30]. One method to defending SAT Attack is the cyclic logic locking [21]. The cyclic logic locking strategically creates cycles in a combinational circuit. As a result, the SAT Attack will be trapped into an infinite loop or obtain an incorrect key vector while attacking. Although the cyclic logic locking shows its effectiveness against SAT Attack, it was still cracked by CycSAT [23], [34]. CycSAT can be viewed as an SAT Attack with a new pre-analysis step, which is to search the noncyclic (NC) condition from the locked cyclic circuit to guarantee the succeeding SAT Attack will work well.

Despite the CycSAT introduces an elegant and effective algorithm to decrypt a locked cyclic circuit, it still has shortcomings. SRClock [15], [16] was proposed to defend the CycSAT by creating *Super Cycles*, aiming to drag out CycSAT's performance. On the other hand, since CycSAT assumes that there will be no noncombinational cycle when the correct key vector is fed, it will prune all the noncombinational cycles when the NC condition has been extracted. Although this assumption sounds reasonable, it is not completely comprehensive. Focusing on this shortcoming, Rezaei *et al.* [14] proposed a method to invalidate CycSAT. The method creates cycles that behave noncombinationally in unreachable states. These noncombinational cycles will not be broken under any correct key vector. When the CycSAT launches attack on this locking method, it will prune all the noncombinational cycles first. Pruning all the noncombinational cycles is equivalent to pruning the correct key vectors.

With the similar concept of [14], Chiang *et al.* [6] proposed LOOPLock to protect designs from SAT Attack, CycSAT, and Removal Attack [31], [32]. Two types of cyclic structures are created in LOOPLock, called *Type-I cycle pair* and *Type-II cycle pair*. The Type-I cycle pair is to invalidate the SAT Attack, while the Type-II cycle pair is for defending CycSAT.

In this article, we discuss the security concerns of LOOPLock by structural analysis and propose an attacking approach to unlock LOOPLock. Furthermore, we propose LOOPLock 2.0 to elevate the security level.

## II. Preliminaries

### A. Background

An input-controlling value (ICV) of a gate $g$ is the value that can determine the output value of $g$. An input-noncontrolling value (INCV) is the inverse of ICV. A gate $d$ is called a *dominator* of a gate $g$ when every path from $g$ to any primary output (PO) must pass through $d$. Given a gate $g$ and the set $G$ of dominators of $g$, the *side inputs* of $G$ are the fanins of $G$, but are not in the fanout cone of $g$. The *stuck-at fault* is a fault model used to describe manufacturing defects in the circuit. A *stuck-at fault* means that the value on the wire will be fixed to either 1 (stuck-at 1) or 0 (stuck-at 0) due to manufacturing defects. A *stuck-at fault test* is a process to generate test patterns capable of distinguishing a faulty circuit from fault-free one. The mandatory assignments (MAs) are unique values assigned to wires to test a fault on a wire $w$. The MAs are assignments for activating or propagating the fault effect. If the MAs of a fault are inconsistent, no test pattern exists for detecting the fault.

### B. Node Merging

Node Merging (NM) [4], [5] is a logic optimization technique considering observability don't cares. Let $n_t$ denote a target node, and $n_s$ denote a substitute node. Merging $n_t$ and $n_s$ is equivalent to replacing $n_t$ with $n_s$. After merging, $n_t$ is removed from the circuit and $n_t$'s original fanout nodes will be driven by $n_s$ instead. Generally, merging two nodes in a circuit will change the circuit's functionality, and it can be modeled as a *misplaced-wire error* as stated in [4] and [5]. However, if the effect of this misplaced-wire error cannot be observed at any PO of the circuit, merging these two nodes will not affect the functionality of the circuit. The sufficient condition of the node mergers with respect to a target node $n_t$ was proposed in [4] and [5].

*Condition 1 [4], [5]:* Let $f$ denote an error of replacing $n_t$ with $n_s$. If $n_s = 1$ or $D$, and $n_t = D$ are MAs for the stuck-at 0 fault test on $n_t$, and $n_s = 0$ or $\overline{D}$, and $n_t = \overline{D}$ are MAs for the stuck-at 1 fault test on $n_t$, $f$ is undetectable.

$D$ ($\overline{D}$) means that the value is 1/0 (0/1), where 1 (0) is the fault-free value, and 0 (1) is the faulty value. These two symbols in *Condition 1* are used in the ATPG algorithms [18].

### C. NM-Based Cycle Generation

Chen and Wang [4], [5] did not choose the $n_s$ that is located at the $n_t$'s fanout cone to replace $n_t$. This is because that it may form a noncombinational cycle if we choose such kind of $n_s$. Chen *et al.* [3] proposed *Theorem 1* to describe the requirement about being combinational cycles after merging.

*Theorem 1 [3]:* Let $n_t$ denote a target node and $n_s$ denote a substitute node in the 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 that all the side inputs of $C$ are not INCVs simultaneously, $C$ is combinational.

According to *Theorem 1*, distinguishing between a noncombinational cycle and a combinational cycle is equivalent to checking if the value changes on $n_t$ are propagated to $n_s$ or not. If there is no input pattern that can activate the fault effect on $n_t$ and propagates the fault effect to $n_s$, the formed cycle is a combinational cycle, and the $n_s$ is a cyclic substitute node (CSN). Chen *et al.* [3] proposed *Condition 2* based on *Condition 1* to identify *candidate CSNs* efficiently.

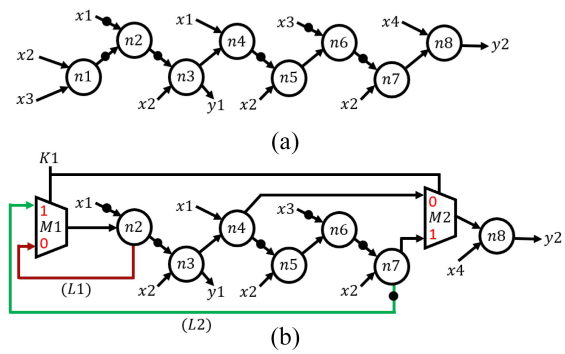


Fig. 1. (a) Original circuit before encryption. (b) Type-I cycle pair, the red cycle is incorrect while the green one is correct.

*Condition 2 [3]:* Let $n_s$ denote a substitute node in the fanout cone of the target node $n_t$. 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.

### D. LOOPLock

LOOPLock is a cyclic logic locking method that can defend circuit from the SAT Attack, CycSAT, and Removal Attack. LOOPLock contains two locking structures called the Type-I and Type-II cycle pairs. For each cycle pair, there are two cycles, where one is a noncombinational cycle and the other is a combinational cycle. In Section II-C, we have discussed how to create functionally correct combinational cycles by using the NM method [3]–[5]. Here, we further explain how to create noncombinational cycles in a circuit.

According to *Theorem 1*, the sufficient condition ensuring the formed cycle $C$ is combinational is that the value changes on $n_t$ are never propagated to $n_s$. That is, there exists a *blocking node* $n_b$ that blocks the effect of value changes from $n_t$ on the path between $n_t$ and $n_s$. Based on this observation, we can choose a node between $n_t$ and $n_b$ to replace $n_t$ for creating a noncombinational cycle. On the contrary, if we choose a node that is in the fanout cone of $n_b$ and use this node to replace $n_t$, the created cycle will be combinational.

The original circuit and the resultant circuit with the Type-I cycle pair are shown as Fig. 1(a) and (b), respectively. In the original circuit, the node $n7$ can be identified as an $n_s$ for $n_1$ by using the methods in [3]–[5]. Thus, we can use $n7$ to replace $n1$ to construct a functionally correct combinational cycle $L2$. Next, by observing the fault effect propagation, we can identify $n4$ as the $n_b$. We choose $n2$ to create the noncombinational cycle $L1$ that affects $y1$. These two cycles are connected to a MUX $M1$, and the key input $K1$ is used as a selection line for $M1$. When the correct key vector is fed ($K1 = 1$), the green cycle $L2$ will be chosen, and the circuit's functionality will be correct.

Next, we introduce the Type-II cycle pair using the example in Fig. 2. In the Type-II cycle pair, it has a noncombinational cycle $L4$ where the noncombinational effect is unobservable at any PO. The other cycle is a combinational cycle $L3$, which has no effect on the circuit's functionality. Similar to

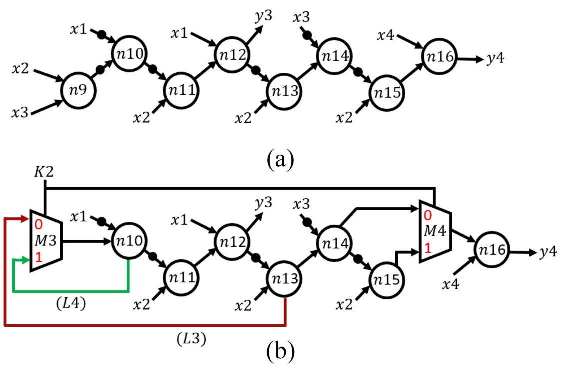


Fig. 2. (a) Original circuit before encryption. (b) Type-II cycle pair, the red cycle is incorrect while the green one is correct.

the Type-I cycle pair, these two cycles are connected to a MUX $M3$, and the key input $K2$ is used to control the MUX. In Fig. 2(a), $n9$ is the $n_t$, and $n12$ is the $n_b$. Thus, $n10$ can be used to form a noncombinational cycle $L4$. Since there is no PO located at a node prior to $n12$, the noncombinational effect of $L4$ will not change the circuit's functionality. Then LOOPLock will select a node that is behind $n12$ to construct a combinational cycle $L3$. When the correct key vector is fed ($K2 = 1$), the green cycle $L4$ will be chosen, and $K2 = 1$ also implies that $n15$ will be chosen for MUX $M4$, which can restore the original functionality.

## III. Our Unlocking Approach

Two types of cycle pairs are created in the LOOPLock. To activate the locked circuit, we have to choose the correct cycle for each type of cycle pair. If we can distinguish between these two types of cycle pairs, we can choose the correct cycles. For ease of discussion, we call the MUX that is located in the left side of a cycle pair as a pre-MUX, while the one located in the right side of a cycle pair as a post-MUX.

### A. Shortcomings of LOOPLock

To distinguish between the Type-I and Type-II cycle pairs, our strategy is to search the structural difference between these two cycle pairs. For a Type-II cycle pair, to avoid affecting the circuit's functionality, there is no PO located between $n_t$ and $n_b$. On the contrary, there exists at least one PO located in between $n_t$ and $n_b$ to invalidate the SAT Attack in a Type-I cycle pair.

Considering this shortcoming, we can distinguish between these two types of cycle pairs by checking whether there is any PO located in between $n_t$ and $n_b$. To achieve this objective, we need to recognize the positions of $n_t$ and $n_b$ in the circuit. First, although $n_t$ has been removed, $n_t$'s position for a cycle pair is still obvious. For example, in Fig. 1(a), $n1$ is $n_t$ and it connects to $n2$. After locking, $n1$ is removed and replaced by the pre-MUX $M1$. Based on this observation, we can ensure that the $n_t$'s position in the original circuit is exactly the pre-MUX's position in the locked circuit. Next, we present how to identify the blocking node $n_b$ in Section III-B.

**Algorithm 1: Pseudo code of the proposed unlocking approach**

**Input**: A locked circuit $C_e$
**Output**: The key vector $K_{vector}$
**for** each key input $KI$ in $C_e$
  $CP \leftarrow$ Find the corresponding cycle pair;
  $M_{pre}, M_{post} \leftarrow$ Find the pre-MUX and post-MUX in $CP$;
  Remove $M_{pre}, M_{post}$ and insert a virtual PI $vpi$;
  Propagate the fault effects $D$ and $\overline{D}$ from $vpi$;
  $n_b \leftarrow$ Find the position of the blocking node;
  **if** (there exists any PO between $vpi$ and $n_b$) **then**
    $CP$ is a Type-I cycle pair;
    Choose the combinational cycle in $CP$;
    Add the corresponding key value $K_c$ into $K_{vector}$;
  **end**
  **else**
    $CP$ is a Type-II cycle pair;
    Choose the non-combinational cycle in $CP$;
    Add the corresponding key value $K_{nc}$ into $K_{vector}$;
  **end**
return $K_{vector}$;

Fig. 3. Pseudocode of the proposed unlocking approach.

### B. Blocking Node Identification

Before identifying the blocking node $n_b$ in each cycle pair, we first conduct a *removal process* on each Type-I and Type-II cycle pair in the locked circuit. The removal process will break the cycles and eliminate MUXes in each cycle pair. In this way, we can obtain an acyclic circuit. After removing these MUXes in a cycle pair, we find that the wire at the pre-MUX's output becomes floating. Thus, we assign a virtual primary input *vpi* in the circuit. Then, we propagate the fault effects from this *vpi* for identifying the position of $n_b$.

We first assign a fault effect (either $D$ or $\overline{D}$) on *vpi*, and propagate the fault effect by assigning proper side input values. Also, we propagate the fault effect $\overline{D}$ in the same manner. Then, we can know the location where the fault effects $D$ and $\overline{D}$ are blocked, and the node is the $n_b$ for the cycle pair.

### C. Flow

The pseudocode of the proposed unlocking approach is shown in Fig. 3. Given a locked circuit $C_e$, for each key input *KI* in $C_e$, we search the corresponding cycle pair *CP*. For each *CP*, we find out its pre-MUX and post-MUX. Then, we remove these MUXes and insert a virtual primary input *vpi*. Afterward, we propagate the fault effects $D$ and $\overline{D}$ from the *vpi* to identify the position of $n_b$. Next, we check if there exists any PO between *vpi* and $n_b$. If so, the *CP* is a Type-I cycle pair; otherwise, the *CP* is a Type-II cycle pair. For the Type-I cycle pair, we choose the combinational cycle, and the key value $K_c$ will be collected in the $K_{\text{vector}}$. For a Type-II cycle pair, we choose the noncombinational cycle with the key value $K_{\text{nc}}$. Finally, the $K_{\text{vector}}$ will be returned after each *CP* has been analyzed.

## IV. LOOPLock 2.0

### A. Enhanced Locking Structure in LOOPLock 2.0

From the discussion in Section III, the difference between the Type-I and Type-II cycle pairs is whether there exists any

PO located between $n_t$ and $n_b$. Thus, to defend the unlocking approach, we first create a new structure having at least one PO located between $n_t$ and $n_b$ in the Type-II cycle pair. In this way, the structures of two types of cycle pairs would be similar.

Since our goal is to create a path connecting from one node between $n_t$ and $n_b$ to a PO in the Type-II cycle pair, we randomly select a PO $y_n$ and its fanin nodes $n_a$ from the original circuit as a subcircuit. Now, we can create a path connecting one node $n_x$ between the pre-MUX and the blocking node $n_b$ to the PO $y_n$, such that $y_n$'s functionality could be intact. We choose $n_x$ and $n_a$ to connect to an additional MUX *Ma* with the selection line *Ka*. When the correct key value of *Ka* is assigned, $n_a$ will be selected to connect to $y_n$. For attackers, however, they cannot directly judge the type of this cycle pair using the proposed unlocking approach. This is because there *does* exist a path from $n_x$ to the PO $y_n$.

On the other hand, the structure of the original Type-I cycle pair also has to be modified. There is at least one node $n_x$ connecting to the PO $y_n$ in the original Type-I cycle pair. Similarly, we randomly select an additional node $n_a$ from the original circuit, and insert a MUX *Ma* between $n_x$ and a PO $y_n$ with the selection line *Ka*. The other input of MUX *Ma* is the node $n_a$. When the correct key value of *Ka* is assigned, the functionalities of the original and enhanced Type-I cycle pairs are identical.

With the enhanced structures, the Type-I and Type-II cycle pairs look similar. In fact, they are identical from the viewpoint that there exists a PO located between *nt* and *nb*. Thus, the unlocking approach cannot attack LOOPLock 2.0 by distinguishing the Type-I and Type-II cycle pairs.

### B. Subcircuit Duplication

In Section IV-A, we introduce the LOOPLock 2.0, which strengthens the security level of the LOOPLock. However, another critical concern in LOOPLock is that the structure of the Type-II cycle pair requires many constraints, which are not easy to meet in practice. In fact, the target node in the Type-II cycle pair has to be a redundant node to the original circuit. We explain this phenomenon from two aspects. First, from the discussion in Section II-D, a blocking node *nb* is the node that can block the fault effect. The side inputs of the nodes between *nt* and *nb* cannot be INCVs simultaneously under any input vector. That is, some side inputs of the nodes between *nt* and *nb* have to be complemented, e.g., $x_i$ and $\overline{x_i}$. Second, there is no path connecting to a PO between *nt* and *nb* in the Type-II cycle pair. Thus, the nodes between *nt* and *nb* have to be the dominators of *nt*. However, from the first aspect, we can find that the side inputs of the dominators cannot be INCVs simultaneously. Therefore, if we conduct the stuck-at 0 and stuck-at 1 fault tests and derive the MAs for a target node *nt*, which are used to construct a Type-II cycle pair, we will find that the *nt* is a redundant node due to inconsistent MAs.

Generally, redundant nodes are not popular in the circuits. Thus, in this section, we further introduce a subcircuit duplication approach to create redundancy. With this approach, we can increase the number of Type-II cycle pairs in a circuit.

TABLE I
COMPARISON OF THE PROPOSED UNLOCKING APPROACH AGAINST THE CYCSAT AND SAT ATTACK ON THE LOCKED CIRCUITS BY LOOPLOCK AND THE SUBCIRCUIT DUPLICATION APPROACH

| **Benchmark** | **Ours** | | **CycSAT** | | **SAT Attack** | |
|---|---|---|---|---|---|---|
| Bench. | Time (s) | key | Result | Time (s) | Result | Time (s) |
| b20 | <0.01 | yes | Inf.loop | Inf.loop | UNSAT | 0.0139 |
| b21 | <0.01 | yes | Inf.loop | Inf.loop | UNSAT | 0.0079 |
| b22 | <0.01 | yes | Inf.loop | Inf.loop | UNSAT | 0.0149 |
| C1908 | <0.01 | yes | Inf.loop | Inf.loop | Inf.loop | Inf.loop |
| C432 | <0.01 | yes | Inf.loop | Inf.loop | Inf.loop | Inf.loop |
| i10 | <0.01 | yes | Inf.loop | Inf.loop | UNSAT | 0.0019 |
| i2c | <0.01 | yes | Inf.loop | Inf.loop | UNSAT | 0.0009 |
| pci_brdge32 | <0.01 | yes | Inf.loop | Inf.loop | UNSAT | 0.0149 |
| rot | <0.01 | yes | Inf.loop | Inf.loop | UNSAT | 0.0001 |
| sasc | <0.01 | yes | Inf.loop | Inf.loop | UNSAT | 0.0039 |
| systemcaes | <0.01 | yes | Inf.loop | Inf.loop | UNSAT | 0.0069 |
| wb_conmax | <0.01 | yes | Inf.loop | Inf.loop | UNSAT | 0.0329 |

Intuitively, if there exists any node connecting to PO between $n_t$ and $n_b$, this subcircuit cannot be used to create the original Type-II cycle pair. Thus, our approach will remove all the paths connecting to POs between $n_t$ and $n_b$. Then, we can use the remaining subcircuit to create the original Type-II cycle pair. Next, to keep the circuit's functionality intact, we duplicate the nodes between $n_t$ and $n_b$ and use these nodes to drive the removed paths connecting to POs. Note that these duplicated nodes should be connected to the same inputs as the original circuit. In this way, we can create more Type-II cycle pairs without changing the circuit's functionality.

Based on the discussion, we find that there are three MUXes needed to be inserted for each cycle pair. Thus, the area overhead with LOOPLock 2.0 may be high on small circuits while it is still very low for most circuits.

## V. EXPERIMENTAL RESULTS

The proposed unlocking approach and LOOPLock 2.0 were implemented in C language within ABC [36] environment in a 3.0-GHz Linux platform (CentOS 4.6). The benchmarks are from the IWLS 2005 suite [35]. We used LOOPLock to generate these locked benchmarks. Every benchmark was represented in AIG in blif format.

First, we conducted experiments for demonstrating the effectiveness of the proposed unlocking approach. We compared our results with the well-known methods—the CycSAT and SAT Attack. We reimplemented the program in [6], which are about using the LOOPLock to defend CycSAT and SAT Attack, and obtained the results.

The experimental results are shown in Table I. In this experiment, we generated only one Type-I cycle pair and one Type-II cycle pair in each circuit. However, some circuits do not have the required structure for constructing the Type-II cycle pair. Thus, we conducted the subcircuit duplication approach to increase the Type-II cycle pairs. The column "key" represents whether the key vector is correct (yes) or not (no). The "Inf.loop" means that the unlocking method was trapped into an infinite loop such that no key vector can be returned. The experimental results show that our unlocking approach can efficiently obtain the correct key vector. This is because the computation complexity of our approach comes

TABLE II
COMPARISON OF THE MAXIMUM NUMBER OF TYPE-II CYCLE PAIRS BETWEEN LOOPLOCK AND LOOPLOCK 2.0

| **Benchmark Information** | | | **\|MAX. Type-II cycle pair\|** | |
|---|---|---|---|---|
| Bench. | \|PI\|/\|PO\| | \|Node\| | \|LOOPLock\| | \|LOOPLock 2.0\| |
| b20 | 522/512 | 12219 | 0 | 129 |
| b21 | 522/512 | 12782 | 0 | 135 |
| b22 | 767/757 | 18488 | 1 | 196 |
| C1908 | 33/25 | 414 | 0 | 25 |
| C432 | 36/7 | 209 | 0 | 40 |
| i10 | 257/224 | 2673 | 0 | 59 |
| i2c | 147/142 | 1306 | 0 | 8 |
| pci_brdge32 | 3521/3566 | 24369 | 0 | 100 |
| rot | 135/107 | 1063 | 1 | 23 |
| sasc | 133/129 | 784 | 0 | 5 |
| systemcaes | 930/799 | 13054 | 0 | 138 |
| wb_conmax | 1900/2186 | 48429 | 15 | 150 |

TABLE III
RESULT OF THE PROPOSED UNLOCKING APPROACH FOR LOOPLOCK ON THE LOCKED CIRCUITS BY LOOPLOCK 2.0

| **Benchmark Information (locked by LOOPLock 2.0)** | | | | | **Ours** | |
|---|---|---|---|---|---|---|
| Bench. | \|PI\|/\|PO\| | \|Node\| | \|Type-I\| | \|Type-II\| | Time (s) | key |
| b20 | 526/512 | 12258 | 1 | 1 | <0.01 | no |
| b21 | 526/512 | 12809 | 1 | 1 | <0.01 | no |
| b22 | 771/757 | 18535 | 1 | 1 | <0.01 | no |
| C1908 | 37/25 | 443 | 1 | 1 | <0.01 | no |
| C432 | 40/7 | 240 | 1 | 1 | <0.01 | no |
| i10 | 261/224 | 2685 | 1 | 1 | <0.01 | no |
| i2c | 151/142 | 1331 | 1 | 1 | <0.01 | no |
| pci_brdge32 | 3525/3566 | 24394 | 1 | 1 | <0.01 | no |
| rot | 139/107 | 1089 | 1 | 1 | <0.01 | no |
| sasc | 137/129 | 809 | 1 | 1 | <0.01 | no |
| systemcaes | 934/799 | 13082 | 1 | 1 | <0.01 | no |
| wb_conmax | 1904/2186 | 48460 | 1 | 1 | <0.01 | no |

from propagating the fault effects on two designated *vpis*, which is not computation-intensive. For CycSAT, the results are all Inf.loop, this is because CycSAT cannot effectively find the condition to break the cycle. This situation causes the succeeding SAT Attack to find no DIPs due to the noncombinational cycle. Similar to CycSAT, BeSAT [23] still constructs the same NC condition as CycSAT. Thus, it is quite challenging for BeSAT to unlock the circuit locked by LOOPLock 2.0. For the SAT Attack, the results are either UNSAT or Inf.loop due to the existence of the noncombinational cycle in the Type-I cycle pair.

For the second experiment, we show the number of Type-II cycle pairs that we can construct for each benchmark when applying the subcircuit duplication approach. Table II shows the results in identifying the Type-II cycle pairs as compared with LOOPLock. The columns |PI|/|PO| and |Node| show the information of each original benchmark. The columns |LOOPLock| and |LOOPLock 2.0| show the numbers of identified Type-II cycle pairs in LOOPLock and our approach, respectively. The experimental results show that the average number of Type-II cycle pairs in our approach is much more than that in LOOPLock.[1]

For the last experiment, we show the results of using the proposed unlocking approach for LOOPLock to attack the locked circuits by LOOPLock 2.0. Table III shows that the returned key vectors are incorrect when applying the proposed unlocking approach to the locked circuits. This is because both the Type-I and Type-II cycle pairs have at least one PO located between *nt* and *nb* in LOOPLock 2.0. In the proposed unlocking approach for LOOPLock, we will identify a cycle pair as a Type-I cycle pair when there exists a path connecting to the PO from a node in between *nt* and *nb*. Thus, the returned key vectors were incorrect under the proposed unlocking approach. The result indicates that the security level of circuit is elevated by LOOPLock 2.0.

[1]The original data about the number of the Type-II cycle pairs shown in [6] is incorrect due to a bug. In this article, we correct the data.

## VI. CONCLUSION

In this article, we discussed the weakness of LOOPLock and proposed an unlocking approach to attack LOOPLock. The experimental results show that the proposed approach is able to unlock the locked circuits effectively and efficiently. Furthermore, we proposed LOOPLock 2.0 having enhanced structures to strengthen the security of circuit. Finally, we proposed a subcircuit duplication approach that enriches the construction of Type-II cycle pairs in a benchmark.

## 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. Cryptogr. Hardw. Embedded Syst.*, vol. 2019, no. 1, pp. 97–122, 2019.

[2] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware trojan attacks: Threat analysis and countermeasures," in *Proc. IEEE*, vol. 102, no. 8, pp. 1229–1247, Aug. 2014.

[3] 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. SOCC*, 2015, pp. 257–262.

[4] Y.-C. Chen and C.-Y. Wang, "Fast detection of node mergers using logic implications," in *ICCAD Dig. Tech. Papers*, 2009, pp. 785–788.

[5] Y.-C. Chen and C.-Y. Wang, "Fast node merging with don't cares using logic implications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 11, pp. 1827–1832, Nov. 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. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2178–2191, Oct. 2020.

[7] S. Dupuis, P.-S. Ba, G. D. Natale, M.-L. Flottes, and B. Rouzeyre, "A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans," in *Proc. IOLTS*, 2014, pp. 49–54.

[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. DAC*, 2019, pp. 1–6.

[9] L. Li and A. Orailoglu, "Piercing logic locking keys through redundancy identification," in *Proc. DATE*, 2019, pp. 540–545.

[10] A. Marcelli, M. Restifo, E. Sanchez, and G. Squillero, "An evolutionary approach to hardware encryption and trojan-horse mitigation," in *Proc. DATE*, 2017, pp. 1593–1598.

[11] J. Rajendran *et al.*, "Fault analysis-based logic encryption," *IEEE Trans. Comput.*, vol. 64, no. 2, pp. 410–424, Feb. 2015.

[12] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Logic encryption: A fault analysis perspective," in *Proc. DATE*, 2012, pp. 953–958.

[13] 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. DATE*, 2018, pp. 85–90.

[14] A. Rezaei, Y. Li, Y. Shen, S. Kong, and H. Zhou, "CycSAT-unresolvable cyclic logic encryption using unreachable states," in *Proc. ASPDAC*, 2019, pp. 358–363.

[15] S. Roshanisefat, H. M. Kamali, and A. Sasan, "SRClock: SAT-resistant cyclic logic locking for protecting the hardware," in *Proc. GLSVLSI*, 2018, pp. 153–158.

[16] S. Roshanisefat, H. M. Kamali, H. Homayoun, and A. Sasan, "SAT-hard cyclic logic obfuscation for protecting the IP in the manufacturing supply chain," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 4, pp. 954–967, Apr. 2020.

[17] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending piracy of integrated circuits," *Computer*, vol. 43, no. 10, pp. 30–38, Oct. 2010.

[18] 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. Electron. Comput.*, vol. EC-16, no. 5, pp. 567–580, Oct. 1967.

[19] M. S. Samimi, E. Aerabi, Z. Kazemi, M. Fazeli, and A. Patooghy, "Hardware enlightening: No where to hide your hardware trojans!" in *Proc. IOLTS*, 2016, pp. 251–256.

[20] B. Shakya, X. Xu, X. Xu, and D. Forte, "CAS-lock: A security-corruptibility trade-off resilient logic locking scheme," *IACR Trans. Cryptogr. Hardw. Embedded Syst.*, vol. 2020, no. 1, pp. 175–202, 2019.

[21] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "Cyclic obfuscation for creating SAT-unresolvable circuits," in *Proc. GLSVLSI*, 2017, pp. 173–178.

[22] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," in *Proc. HOST*, 2017, pp. 95–100.

[23] Y. Shen, Y. Li, A. Rezaei, S. Kong, D. Dlott, and H. Zhou, "BeSAT: Behavioral SAT-based attack on cyclic logic encryption," in *Proc. ASPDAC*, 2019, pp. 657–662.

[24] Y. Shen, A. Rezaei, and H. Zhou, "A comparative investigation of approximate attacks on logic encryptions," in *Proc. ASPDAC*, 2018, pp. 271–276.

[25] Y. Shen and H. Zhou, "Double DIP: Re-evaluating security of logic encryption algorithms," in *Proc. GLSVLSI*, 2018, pp. 179–184.

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

[27] Y. Xie and A. Srivastava, "Mitigating SAT attack on logic locking," in *Proc. Int. Conf. Cryptogr. Hardw. Embedded Syst.*, 2016, pp. 127–146.

[28] Y. Xie and A. Srivastava, "Anti-SAT: Mitigating SAT attack on logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 2, pp. 199–207, Feb. 2019.

[29] M. Yasin, B. Mazumdar, J. J. V. Rajendran, and O. Sinanoglu, "SARlock: SAT attack resistant logic locking," in *Proc. HOST*, 2016, pp. 236–241.

[30] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proc. CCS*, 2017, pp. 1601–1618.

[31] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Security analysis of anti-SAT," in *Proc. ASPDAC*, 2016, pp. 342–347.

[32] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal attacks on logic locking and camouflaging techniques," *IEEE Trans. Emerg. Topics Comput.*, vol. 8, no. 2, pp. 517–532, Apr.–Jun. 2020.

[33] M. Yasin, J. Rajendran, O. Sinanoglu, and R. Karri, "On improving the security of logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 9, pp. 1411–1424, Sep. 2016.

[34] H. Zhou, R. Jiang, and S. Kong, "CycSAT: SAT-based attack on cyclic logic encryptions," in *Proc. ICCAD*, 2017, pp. 49–56.

[35] (Jun. 2005). *IWLS 2005 Benchmarks*. [Online]. Available: http://iwls.org/iwls2005/benchmarks.html

[36] Berkeley Logic Synthesis and Verificaiton Group. *ABC: A System for Sequential Synthesis and Verification*. Accessed: Nov. 24, 2020. [Online]. Available: http://www.eecs.berkeley.edu/ alanmi/abc