LOOPLock 3.0: A Robust Cyclic Logic Locking Approach

Pei-Pei Chen, Xiang-Min Yang, Yu-Cheng He, Yung-Chih Chen, Yi-Ting Li, Chun-Yao Wang

2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC) | 979-8-3503-9354-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/ASP-DAC58780.2024.10473877

Abstract—**Cyclic logic locking is a cutting-edge hardware security method developed to defend against SAT Attack. It introduces cycles into the original circuit, which can cause the circuit to either get trapped in an endless loop or generate incorrect outputs if an incorrect key is used. Recently, a new cyclic logic locking method called LOOPLock 2.0 was proposed. Its primary feature is that the circuit retains its cyclic structure regardless of whether the correct key vector is applied or not. However, LOOPLock 2.0 can still be successfully attacked using locking structure analysis in the state-of-the-art. As a result, this paper presents a more robust cyclic logic locking approach LOOPLock 3.0 to counteract state-of-the-art attacks. The experimental results validate the effectiveness of the proposed approach.**

I INTRODUCTION

As time goes by, Integrated Circuits are now getting more and more complex. IC design companies encounter several security risks, leading to the arising need for hardware security techniques. Many protection techniques have been proposed recently to address hardware security issues [5] [9] [10] [11] [12] [13] [14] [17] [18] [19] [20] [21] [22] [23] [24] [26] [28] [29] [39] [40] [42] [43] [44] [49] [50].

Logic locking [24] is a technique that introduces additional logic gates with key inputs into the original circuit. The locked circuit only functions correctly with the correct key vector. Logic locking makes it difficult for attackers to understand the design of an IC and even replicate it. However, most of the logic locking methods are vulnerable to a multiplicity of unlocking approaches [1] [2] [4] [8] [15] [16] [27] [30] [31] [32] [33] [34] [35] [36] [37] [38] [41] [45] [46] [47] [48].

Boolean Satisfiability-based (SAT) Attack [38] shows its power to unlock logic encrypted circuits. It obtains the correct key vector by pruning out all the incorrect key vectors iteratively. At least one incorrect key vector can be found when the circuit produces different outputs under a specific input pattern and different key vectors. The input patterns that can recognize incorrect key vectors are named distinguishing input patterns (DIPs). As any DIP is observed, a constraint of generating correct output under this DIP will be added to the Conjunctive Normal Form (CNF) formula of this circuit, thereby eliminating at least one incorrect key vector. Therefore, once all the DIPs have been identified, the circuit will function correctly with the remaining key vectors.

This work is supported in part by the National Science and Technology Council (Taiwan) under MOST 109-2221-E-007-082-MY2, MOST 111-2221-E-007-121, MOST 111-2221-E-011-137-MY3, NSTC 112-2218-E-007-014, NSTC 112-2221-E-007-106-MY2, NSTC 112-2221-E-007-108, and NSTC 112-2425-H-007-002.

P.-P. Chen, X.-M. Yang, Y.-C. He, Y.-T. Li, and C.-Y. Wang are with the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 30013, R.O.C. (email: ling8835@gmail.com; yhm19930125@gmail.com; jessica30217@gmail.com; yitingli.yt@gmail.com; wcyao@cs.nthu.edu.tw).

Y.-C. Chen is with the Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan 10607, R.O.C. (email: ycchen.ee@mail.ntust.edu.tw).

Cyclic logic locking [29] is a method to counteract SAT Attack. It introduces feedback loops into the circuits, which causes *statefulness* or *oscillation* to the circuit under incorrect key vectors. Statefulness refers to a situation in which different outputs are created under a pair of identical input pattern and key vector. Statefulness may cause the SAT Attack stuck in an endless loop. Additionally, oscillation is another situation in which some wires in the circuit oscillate. Oscillation may lead to incorrect key vectors being returned by SAT Attack.

CycSAT [48] is proposed to attack cyclic logic locking methods. It assumes the circuits are acyclic under the correct key vector. Thus, it computes the non-cyclic (NC) condition of a locked circuit by structural analysis on every cycle in the circuit. By adding the NC condition into the CNF formula of the encrypted circuit, the resulting CNF formula is bounded to keeping the key vectors that will not provide cycles.

However, CycSAT cannot rule out all of the cycles. It cannot break cycles with more than one feedback edge. Behavioral SAT-based Attack (BeSAT) [32] is proposed to surmount this shortcoming of CycSAT. Same as CycSAT, BeSAT first computes the NC condition of the circuit. Additionally, it applies SAT Attack and records DIPs that are identified. When the SAT solver finds any DIP, BeSAT checks whether this DIP has already appeared. If DIPs are found repeatedly, it suggests the presence of statefulness. In such cases, the key vector responsible for causing the DIP statefulness must be eliminated. Subsequently, BeSAT utilizes a ternary-based SAT method to prune key vectors that cause oscillation in any wire of the circuit. BeSAT shows its success in breaking all of the cycles in the locked circuit.

LOOPLock 2.0 [42] is a cyclic logic locking method that can defend against SAT Attack, CycSAT, BeSAT, and Removal Attacks simultaneously. It strengthens the security level of the two locking structures proposed in LOOPLock [10], including *Type-I cycle pair* and *Type-II cycle pair*. There are one combinational cycle and one non-combinational cycle in each cycle pair. Regardless of whether the key vector fed into the encrypted circuit is correct or incorrect, one of the cycles will be activated. Therefore, when the CycSAT and BeSAT try to attack LOOPLock 2.0 by breaking every cycle, they will rule out the correct key vectors though.

Although LOOPLock 2.0 presents an effective and efficient cyclic logic locking method to resist attacks from logic unlocking methods, it has been found to have some shortcomings. It can be attacked by breaking all of the non-combinational cycles in the state-of-the-art [4]. By propagating fault effects and identifying the positions of the blocking node in cycle pairs, the cycles that cause non-combinational effects in the circuit can be recognized. These cycles can be ruled out by replacing them with arbitrary constant values (0 or 1), which will not change the functionality of the circuit. Then the correct key can be obtained by applying SAT Attack. Thus, in this paper, we propose a robust cyclic logic locking structure

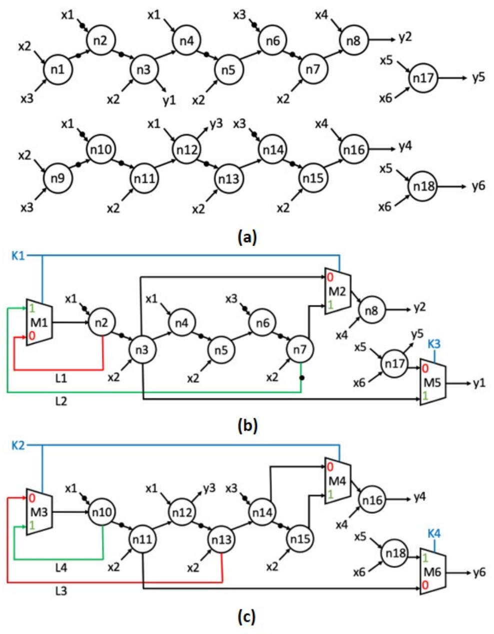

Fig. 1. An example for LOOPLock 2.0. (a) The original circuit. (b) The Type-I cycle pair. (c) The Type-II cycle pair.

LOOPLock 3.0, which is more resilient against attacks from [4].

II PRELIMINARIES

A. LOOPLock 2.0

LOOPLock 2.0 [42] enhances the security level of LOOPLock [10]. LOOPLock can be attacked by the unlocking method proposed in [42]. It first collects the MUXes controlled with the same key input. For ease of discussion, the MUX with two feedback edges in a cycle pair, located on the left side, is referred to as pre-MUX, while the other MUX controlled by the same key input, located on the right side, is referred to as post-MUX. The pre-MUX is considered as the location of *target node* n_t due to the structure of LOOPLock. Then, the position of the *blocking node* n_b can be found by its *Blocking Node Identification* method [42]. After that, it distinguishes the structural difference between Type-I and Type-II cycle pairs by identifying if there is at least a PO located between n_t and n_b. It collects the key values that select the combinational cycles in the Type-I cycle pairs and the key values that select the non-combinational cycles in the Type-II cycle pairs. These key values are correct exactly.

Therefore, the main idea of LOOPLock 2.0 is to conceal the structural difference between the Type-I and Type-II cycle pairs. Fig. 1 is an example of LOOPLock 2.0. Fig. 1(b) shows the example of the Type-I cycle pair. It inserts an additional key gate $M5$ with an additional key input $K3$. The correct key value is $K3 = 1$. The connection between $n3$ and $y1$ is hidden and the wrong signal generated by a selected node $n17$ is propagated to y1 while $K3 = 0$. Besides, Fig. 1(c) shows the example of the Type-II cycle pair. It inserts an additional key gate $M6$ with an additional key input $K4$. The correct key value is $K4 = 1$. While $K4 = 0$, a wrong path from $n11$, which is located between n_t and n_b, is selected by $M6$ to $y6$. Due to the insertion of the additional key gates $M5$ and $M6$, the structural difference cannot be identified by the unlocking method [42] to LOOPLock. Thus, the correct key vector cannot be obtained.

III SHORTCOMINGS OF LOOPLOCK 2.0

LOOPLock 2.0 prevents attackers from distinguishing the structural difference between the Type-I and Type-II cycle pairs and then choosing the correct cycles. However, LOOPLock 2.0 is still faced with some security concerns. In this section, we discuss three shortcomings of LOOPLock 2.0 that may affect the security of the locked circuit.

1) The post-MUX in the Type-I cycle pair may post the security concern on the locking structure.
2) In the Type-II cycle pair, if the shared key input is separated into two distinct key inputs, the locking structure may be decrypted.
3) The non-combinational cycles and combinational ones can still be distinguished since the position of the blocking nodes n_b can be found by [4].

For the first shortcoming, the Type-I cycle pair can resist SAT Attack with the pre-MUX only. The post-MUX in the Type-I cycle pair is just for it to be structurally similar to the Type-II cycle pair. However, since LOOPLock 2.0 randomly selects node for creating wrong path of the post-MUX, the SAT solver could find DIPs that block the non-combinational effect under the incorrect key input. It causes the wrong key input to be pruned and the correct key input is found. For the second shortcoming, the Type-II cycle pair is vulnerable to CycSAT-II after splitting the shared key inputs of the pre-MUX and post-MUX. CycSAT-II leaves the combinational cycle in the circuit rather than ruling out all the cycles. Hence, the Type-II cycle pair can be decrypted when the key inputs of the pre-MUX and post-MUX are independent. 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. With this shortcoming, an attacking approach to unlock LOOPLock 2.0 [4] is proposed.

IV THE PROPOSED LOCKING APPROACH LOOPLOCK 3.0

A. LOOPLock 3.0

To elevate the strength of cyclic logic locking, we propose a locking approach LOOPLock 3.0 in this subsection. Given the original circuit in Fig. 1(a), the resultant locked circuit is as shown in Fig. 2(e). In the following paragraphs, we will introduce the construction method of our locking approach with this circuit and the corresponding reasons.

To defend against SAT Attack, we keep the key gate $M1$ of the Type-I cycle pair in LOOPLock 2.0, which contains a non-combinational cycle $L1$ affecting POs and a combinational cycle $L2$ restoring the correct functionality of the circuit. Let us explain the construction of $M1$. Let $n1$ be the n_t in the original circuit. We first identify cyclic substitute node (CSN) [3] and n_b, which are $n7$ and $n4$, respectively. Then we construct the key gate $M1$ of the Type-I cycle pair with a feedback path from $n7$ and a feedback path from $n3$, which is located between n_t and n_b. As a result, the observable non-combinational effect can invalidate SAT Attack. The updated circuit is as shown in Fig. 2(a).

Next, we use a key gate $M3$ with a key input $K3$ to hide the position of n_b for defending against the approach [4]. The yellow node $n4$ is the real n_b while $K3 = 1$, and the

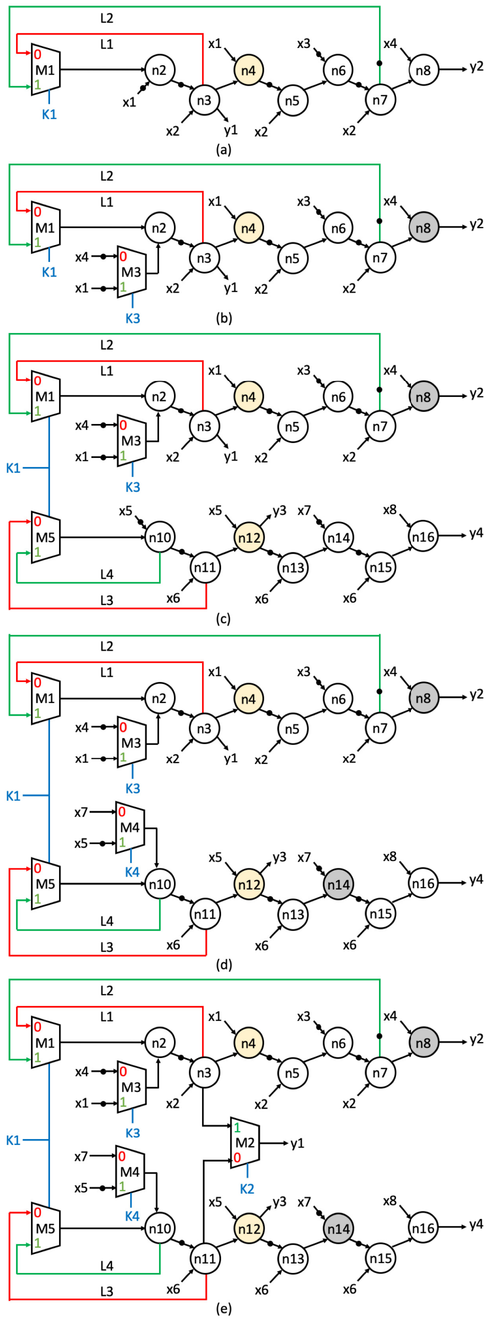

Fig. 2. The steps of constructing the proposed LOOPLock 3.0. (a) Inserting $M1$ of the Type-I cycle pair. (b) Inserting $M3$ for hiding n_b of the Type-I cycle pair. (c) Inserting $M5$ of the Type-III cycle pair. (d) Inserting $M4$ for hiding n_b of the Type-III cycle pair. (e) Inserting $M2$ for hiding the structural difference between the Type-I and Type-III cycle pairs.

gray node $n8$ is used as a fake n_b while $K3 = 0$. Let us explain the construction of $M3$. After inserting $M1$, we use the information of $n4$, which is n_b in Type-I cycle pair, to find the position for inserting $M3$. We observe that $n4$ will block the fault effects since $n2$'s side input is assigned an input-noncontrolling value ($x1 = 0$) when propagating the fault effects from $M1$. However, $x1 = 0$ is an input-controlling value to $n4$ such that $n4$ is n_b. Based on this, the critical structure is that $n2$ is prior to $n4$, and $n2$'s side input is opposite to $n4$'s side input. To hide the relation between $n2$ and $n4$, we insert a key gate $M3$ in between $n2$ and $n2$'s side input $\overline{x1}$.

Furthermore, to create the wrong path of $M3$, we select a node as a fake n_b. Note that the fake n_b can be a node before or after the real n_b. In Fig. 2(a), $n8$ is selected as the fake n_b for the Type-I cycle pair. To block the fault effects at $n8$ while $K3 = 0$, the wrong path of $M3$ needs to be opposite to $n8$'s side input. Thus, we connect the wrong path of $M3$ to $\overline{x4}$. The updated circuit is as shown in Fig. 2(b). As a result, if attackers assign $x4 = 0$ and $K3 = 0$ to propagate the fault effects forward, $n8$'s side input will be an input-controlling value and blocks the fault effects at $n8$. The reason why $M3$ can confuse attackers about the position of n_b is that the fault effects may be blocked at the fake n_b, instead of the real n_b during the fault effect propagation. If attackers cannot confirm the found n_b is real, they are unable to distinguish between non-combinational and combinational cycles correctly.

We further create a new structure called the Type-III cycle pair using a key gate $M5$ sharing the same key input $K1$ as $M1$, and **two** non-combinational cycles $L3$ and $L4$, which are unobservable at POs to invalidate CycSAT and BeSAT. Let us explain the construction of $M5$. Let $n9$ be the n_t in the original circuit. We first find $n12$ as the n_b by propagating the fault effects from $n9$ and ensure that there exists no path from any node between n_t and n_b to any PO. Then we replace $n9$ with a key gate $M5$ and choose two nodes between $n9$ and $n12$ to create feedback paths to $M5$ forming two non-combinational cycles. The updated circuit is as shown in Fig. 2(c). This structure will lead to a contradiction while deriving the NC condition in CycSAT and BeSAT. This is because non-combinational cycles are not allowed for these attacking methods. As a result, the correct key vector cannot be obtained. The detailed evaluation will be discussed in Section IV-B.

We then use $M4$ to hide the position of n_b in the Type-III cycle pair. The yellow node $n12$ is the real n_b while $K4 = 1$, and the gray node $n14$ is used as a fake n_b while $K4 = 0$. We use the same strategy as constructing $M3$ to insert $M4$. We first find $n10$, whose side input is opposite to $n12$'s side input. Then we insert $M4$ between $n10$ and $n10$'s side input to hide the relation between $n10$ and $n12$. Next, we select $n14$ as a fake n_b and connect the wrong path of $M4$ to $x7$ to block the fault effects at $n14$ while $K4 = 0$. The updated circuit is as shown in Fig. 2(d).

Since both the non-combinational cycles in the Type-III cycle pair do not affect the functionality of the circuit, we just use the same key input $K1$ to control the Type-I and Type-III cycle pairs. The green cycles in Fig. 2(d) will be selected under the correct key value of $K1 = 1$, and the red cycles will be selected as $K1 = 0$.

Another key gate $M2$ is inserted between $n3$ and $y1$, and the wrong path for $M2$ is from a node between $M5$ and the blocking node $n12$ of the Type-III cycle pair. $M2$ is used for hiding the structural difference between the Type-I and Type-III cycle pairs, and for propagating the non-combinational effect in the Type-III cycle pair to the PO $y1$. The updated circuit is as shown in Fig. 2(e). In the locking structure of our approach, we only have the Type-I and Type-III cycle pairs without Type-II cycle pair. Algorithm 1 shows the pseudo-code of the proposed locking approach.

Algorithm 1 Locking Approach

```
Input: Two sub-circuits $C_1$ and $C_2$.
Output: A locked circuit $C_L$.
1: Find CSN and $n_{b1}$ for a $n_{t1}$ in $C_1$;
2: Replace $n_{t1}$ with the structure of pre-MUX in the Type-I cycle pair;
3: $C_{L1} \leftarrow$ Obfuscate the position of $n_{b1}$ in $C_1$;
4: Find CSN and $n_{b2}$ for a $n_{t2}$ in $C_2$;
5: Replace $n_{t2}$ with the structure of pre-MUX in the Type-III cycle pair;
6: $C_{L2} \leftarrow$ Obfuscate the position of $n_{b2}$ in $C_2$;
7: Return $C_L \leftarrow$ Hide the structural difference between $C_{L1}$ and $C_{L2}$;
```

B. Evaluation of LOOPLock 3.0

Here we explain how our locking approach defends against the approach [4] with the example in Fig. 2(e). For the first shortcoming, the post-MUX in the Type-I cycle pair of LOOPLock 2.0 may lower the encryption strength. Since we only keep the pre-MUX of the Type-I cycle pair to invalidate SAT Attack, this shortcoming is vanished in our approach.

For the second shortcoming, LOOPLock 2.0 can be unlocked by the method of key-splitting. Although we use the shared key input $K1$ in our locking approach, it is still effective against CycSAT and BeSAT even $K1$ is split into two key inputs $K1_1$ and $K1_2$. The analysis is as follows: We apply the same unlocking processes as in Section III to the locked circuit in Fig. 2(e). First, we use two split key inputs $K1_1$ to control $M1$, and $K1_2$ to control $M5$. Next, we apply CycSAT-II on the modified circuit. Since CycSAT-II and BeSAT use the same strategies to construct NC conditions, we analyze the locked circuit against CycSAT-II, which avoids non-combinational cycles. The NC conditions for the four cycles, $L1 \sim L4$, are shown in EQ(1).

$$\begin{aligned} NC_{L1} &= K1_1 \vee (K3 \wedge x1) \vee (\overline{K3} \wedge x4) \vee \overline{x2}) \\ NC_{L2} &= \overline{K1_1} \vee (K3 \wedge x1) \vee (\overline{K3} \wedge x4) \vee \overline{x2} \vee \overline{x1} \vee x3) \\ NC_{L3} &= K1_2 \vee (K4 \wedge x5) \vee (\overline{K4} \wedge \overline{x7}) \vee \overline{x6}) \\ NC_{L4} &= \overline{K1_2} \vee (K4 \wedge x5) \vee (\overline{K4} \wedge \overline{x7}) \end{aligned} \tag{1}$$

Note that these NC conditions are not added to the CNF formula directly for SAT solving. CycSAT-II searches the input vectors sensitizing any cycle, and then uses these input vectors to simplify the NC conditions. The results are shown in EQ(2).

$$\begin{aligned} NC_{L1}(x1=0, x2=1, x4=0) &= K1_1 \\ NC_{L2}(x1=1, x2=1, x3=0, x4=0) &= \overline{K1_1} \vee K3 \\ NC_{L3}(x5=0, x6=1, x7=1) &= K1_2 \\ NC_{L4}(x5=0, x7=1) &= \overline{K1_2} \end{aligned} \tag{2}$$

Finally, the NC conditions for the four cycles are ANDed as shown in EQ(3), which is the resultant NC condition appended to the CNF formula. Appending the resultant NC condition prevents the SAT solver from getting the correct key vector due to the contradiction of $\boldsymbol{K1_2} \wedge \overline{\boldsymbol{K1_2}}$. Hence, CycSAT-II fails to unlock our locking approach even with split key inputs.

$$NC = K1_1 \wedge (\overline{K1_1} \vee K3) \wedge \boldsymbol{K1_2} \wedge \overline{\boldsymbol{K1_2}} = 0 \tag{3}$$

For the last shortcoming, the position of n_b is identifiable in LOOPLock 2.0. However, in our locking approach, we use key gates $M3$ and $M4$ to hide the position of the real n_b. While applying the unlocking approach [4] to the locked circuit in Fig. 2(e), it first finds out the position of n_b by propagating the fault effects from the pre-MUX. For the Type-I cycle pair, the fault effect is propagated from $M1$. The output value of $M3$ is assigned a constant 1, and then the fault effect can be propagated forward. Thus, $n8$ is found as n_b. Similarly, $n14$ is found as n_b in the Type-III cycle pair. Then the feedback paths of four cycles ($L1$, $L2$, $L3$, and $L4$) are all replaced by constant values. However, breaking the correct combinational cycle $L2$ changes the functionality of the circuit. As a result, the SAT solver cannot find the correct key vector.

It seems that assigning values to $M3$ and $M4$ for choosing a node closer to the pre-MUX can find n_b. However, fake n_b can be a node before real n_b while constructing the locked circuit in our approach. If attackers want to identify real n_b for removing non-combinational cycles, they need to assign the correct key values to $K3$ and $K4$ in advance. However, the correct key values cannot be obtained while the other key values have not been determined. Thus, the proposed locking structure is effective to obfuscate attackers.

TABLE I
RESULTS OF OUR APPROACH IN IDENTIFYING ALL THE LOCKING STRUCTURES.

Benchmark	|PI|/|PO|	|Node|	|Type-I|	|Type-III|	|Lock|
aes_core	789/659	21513	1844	228	228
b17	1454/1512	52920	126	392	126
b20	522/512	12219	66	129	66
b21	522/512	12782	60	135	60
b22	767/757	18488	98	197	98
C1908	33/25	414	8	24	8
C3540	50/22	1038	57	23	23
C432	36/7	206	12	16	12
C5315	178/123	1773	6	6	6
C7552	207/107	2074	18	27	18
dalu	75/16	1740	10	38	10
des_area	368/192	4857	2	2	2
i10	257/224	2673	146	59	59
i2c	147/142	1306	2	8	2
i8	133/81	3310	67	73	67
mem_ctrl	1198/1235	15641	176	173	173
pci_brdge32	3521/3566	24369	45	142	45
pci_spoci_ctrl	85/73	1451	14	32	14
rot	135/107	1063	9	23	9
s13207	700/790	2719	11	62	11
s38417	1664/1742	9219	86	204	86
s38584	1464/1730	12400	31	133	31
s9234	247/250	1958	14	43	14
sasc	133/129	784	3	8	3
systemcaes	930/799	13054	35	138	35
tv80	373/391	9609	413	220	220
usb_funct	1874/1867	15894	23	147	23
wb_conmax	1900/2186	48429	339	376	339
Avg.	-	-	**132.89**	**109.21**	**63.86**

V EXPERIMENTAL RESULTS

In the first experiment, we show the applicability of our LOOPLock 3.0, i.e., the quantity of the Type-I and Type-III cycle pairs that can be constructed in a benchmark. Our locking approach was implemented in C language with ABC. The experiments were conducted on an Intel Xeon E5-2650v2 2.60GHz CentOS 6.10 platform with 64GBytes memory. The experimental results are summarized in Table I. Column 1 lists the benchmark. Columns 2 and 3 list the numbers of PIs, POs, and nodes in each benchmark. Columns 4 and 5 show the

TABLE II
RESULTS OF OUR APPROACH ABOUT AREA-DELAY-PRODUCT (ADP).

Benchmark	|Lock|	|Node|	|L_Node|	|Level|	|L_Level|	ADP
aes_core	5	21513	21642 (1.01)	26	70 (2.69)	2.71
b17	5	52920	53027 (1.00)	43	52 (1.21)	1.21
b20	5	12219	12323 (1.01)	66	91 (1.38)	1.39
b21	5	12782	12887 (1.01)	67	77 (1.15)	1.16
b22	5	18488	18598 (1.01)	69	87 (1.26)	1.27
C1908	5	414	513 (1.24)	32	92 (2.88)	3.56
C3540	5	1038	1132 (1.09)	41	65 (1.59)	1.73
C432	5	206	309 (1.50)	42	84 (2.00)	3.00
C5315	5	1773	1885 (1.06)	38	89 (2.34)	2.49
C7552	5	2074	2173 (1.05)	29	69 (2.38)	2.49
dalu	5	1740	1820 (1.05)	39	44 (1.13)	1.18
des_area	2	4857	4895 (1.01)	33	45 (1.36)	1.37
i10	5	2673	2775 (1.04)	51	64 (1.25)	1.30
i2c	2	1306	1334 (1.02)	16	20 (1.25)	1.28
i8	5	3310	3410 (1.03)	27	27 (1.00)	1.03
mem_ctrl	5	15641	15729 (1.01)	36	44 (1.22)	1.23
pci_brdge32	5	24369	24449 (1.00)	31	32 (1.03)	1.04
pci_spoci_ctrl	5	1451	1548 (1.07)	19	63 (3.32)	3.54
rot	5	1063	1165 (1.10)	51	61 (1.20)	1.31
s13207	5	2719	2824 (1.04)	34	41 (1.21)	1.25
s38417	5	9219	9293 (1.01)	30	37 (1.23)	1.24
s38584	5	12400	12498 (1.01)	36	50 (1.39)	1.40
s9234	5	1958	2063 (1.05)	36	59 (1.64)	1.73
sasc	3	784	839 (1.07)	9	27 (3.00)	3.21
systemcaes	5	13054	13150 (1.01)	47	55 (1.17)	1.18
tv80	5	9609	9708 (1.01)	52	52 (1.00)	1.01
usb_funct	5	15894	15894 (1.00)	27	50 (1.85)	1.85
wb_conmax	5	48429	48536 (1.00)	27	74 (2.74)	2.75

number of identified Type-I and Type-III cycle pairs. Column 6 shows the number of identified locking structures, which is the smaller value between Columns 4 and 5. The average number of identified locking structures in a benchmark is 63.86. Although the locking structures are rare to be constructed in some benchmarks such as des_area, $i2c$, and $sasc$, inserting one locking structure is enough to defend against SAT Attack, CycSAT, BeSAT, and [4]. More locking structures elevate the security level of locked circuit as the number of key inputs increases.

The second experiment is to show the overhead of locked circuit in our approach as summarized in Table II. Columns 1 and 2 list the benchmark and the number of locking structures in the locked circuit. Five locking structures are constructed at most for a benchmark. Since each locking structure has four key inputs, inserting five locking structures results in the total number of key inputs of 20. Hence, the locked circuit is still invulnerable to the brute-force approach. Columns 3 and 4 show the number of nodes in the original circuit and the corresponding locked one. The data in parentheses is the ratio of node count between the locked circuit and the original one. Although we insert five MUXes as key gates and some auxiliary logic for each locking structure, the area overhead is still low in most benchmarks. Columns 5 and 6 show the levels of the critical path in the original circuit and the corresponding locked one. The delay of the locked circuit is affected as the locking structure is inserted on the critical path. The level of the locked circuit increases a lot for some benchmarks, like pci_spoci_ctrl and $sasc$, but this value is intact for some benchmarks, like $i8$ and $tv80$. This result shows that the timing overhead is structurally dependent. The last column shows the Area-Delay-Product (ADP) result, which is the product of the ratios in node count and level. Our locking approach trades the area and delay for improving the security level of the circuit.

For the last experiment, we applied four unlocking methods on the circuits locked by our approach. The time limit in this experiment is set to 10 hours. The results are summarized in Table III. Columns 1 and 2 list the benchmark and the number of locking structures. Column 3 displays the time spent on locking these circuits. Columns 4 and 5 show the required CPU time and the results after applying the state-of-the-art [4]. The unlocking approach cannot find the correct position of n_b to remove all the non-combinational cycles in the locked circuit; therefore, the non-combinational effects in the POs still invalidate the attack. Columns 6 and 7 show the required CPU time and the results after applying SAT Attack. The non-combinational cycles in the locking structures trap the SAT Attack into an infinite loop or terminate the SAT Attack without any result. Columns 8$\sim$11 show the required CPU time and the results after applying CycSAT and BeSAT. Most of the results are "No Result" and "Wrong Key." This is because the constructed NC conditions in CycSAT and BeSAT, which prune out all the non-combinational cycles, prevent the SAT solver from returning correct key vectors. The result of "out of memory" occurred when applying BeSAT on $pci\_bridge32$. This is because BeSAT kept recording the found DIPs and appending the constraints to the CNF formula. If the memory is enough, BeSAT still could not get the correct key vector as the other benchmarks. According to Table III, we know that our locking approach is effective to defend against the state-of-the-art [4], SAT Attack, CycSAT, and BeSAT. Furthermore, since the locations of target nodes are invisible to attackers after removing the locking structures, Removal Attack cannot easily unlock our circuit by removing the locking structures.

VI CONCLUSION

In this work, we propose a robust cyclic logic locking structure LOOPLock 3.0, which enhances the security level of the locked circuit. Our locking approach effectively defends against the state-of-the-art, SAT Attack, CycSAT, BeSAT, and Removal Attack.

REFERENCES

[1] L. Alrahis et al., "OMLA: An Oracle-Less Machine Learning-Based Attack on Logic Locking," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 3, pp. 1602-1606, 2022.
[2] K. Z. Azar et al., "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.
[3] J.-H. Chen et al., "Synthesis and Verification of Cyclic Combinational Circuits," in *Proc. of SOCC*, pp. 257-262, 2015.
[4] P.-P. Chen et al., "An Approach to Unlocking Cyclic Logic Locking: LOOPLock 2.0," in *Proc. of ICCAD*, pp. 1-7, 2022.
[5] R. Chen and H. Zhou, "Statistical Timing Verification for Transparently Latched Circuits," *IEEE TCAD*, vol. 25, no. 9, pp. 1847-1855, 2006.
[6] Y.-C. Chen and C.-Y. Wang, "Fast Detection of Node Mergers using Logic Implications," in *Proc. of ICCAD*, pp. 785-788, 2009.
[7] Y.-C. Chen and C.-Y. Wang, "Fast Node Merging with Don't Cares Using Logic Implications," *IEEE TCAD*, vol. 29, no. 11, pp. 1827-1832, 2010.
[8] Y.-C. Chen, "Enhancements to SAT Attack: Speedup and Breaking Cyclic Logic Encryption," in *ACM Transactions on Design Automation of Electronic Systems*, vol. 23 no. 52, pp. 1-25, 2018.
[9] Y.-C. Chen, "SMARTLock: SAT Attack and Removal Attack-Resistant Tree-Based Logic Locking," in *IEICE Trans*, vol. E103-A no. 5, pp. 733-740, 2020.
[10] H.-Y. Chiang et al., "LOOPLock: LOgic OPtimization based Cyclic Logic Locking," *IEEE TCAD*, vol. 39, no. 10, pp. 2178-2191, 2020.
[11] D.-X. Ji et al., "A Glitch Key-Gate for Logic Locking," in *Proc. of SOCC*, pp. 74-79, 2019.

TABLE III
THE RESULT OF APPLYING THE STATE-OF-THE-ART [4], SAT ATTACK, CYCSAT, AND BESAT ON LOOPLOCK 3.0.

Benchmark Information			**[4]**		**SAT Attack**		**CycSAT**		**BeSAT**	
Benchmark	|Lock|	Time (s)	Time (s)	Result	Time (s)	Result	Time (s)	Result	Time (s)	Result
aes_core	5	16.85	Inf. loop	No Result	Inf. loop	No Result	1.596	No Result	1.414	No Result
b17	5	11.41	Inf. loop	No Result	Inf. loop	No Result	Inf. loop	No Result	46.414	No Result
b20	5	15.21	Inf. loop	No Result	Inf. loop	No Result	0.836	No Result	0.903	No Result
b21	5	15.73	Inf. loop	No Result	Inf. loop	No Result	Inf. loop	No Result	1355.084	Wrong Key
b22	5	14.82	Inf. loop	No Result	Inf. loop	No Result	1.128	No Result	1.160	No Result
C1908	5	0.58	0.147	No Result	0.399	No Result	0.178	No Result	0.164	No Result
C3540	5	0.27	0.697	Wrong Key	Inf. loop	No Result	Inf. loop	No Result	2.263	No Result
C432	5	0.05	0.070	No Result	Inf. loop	No Result	0.077	No Result	0.078	No Result
C5315	5	0.18	0.585	No Result	0.195	No Result	1.021	No Result	0.479	No Result
C7552	5	0.05	0.156	No Result	0.157	No Result	0.379	No Result	0.168	No Result
dalu	5	0.86	13.434	Wrong Key	Inf. loop	No Result	0.131	No Result	0.124	No Result
des_area	2	37.15	Inf. loop	No Result	Inf. loop	No Result	Inf. loop	No Result	6.021	Wrong Key
i10	5	0.85	0.296	No Result	Inf. loop	No Result	0.379	No Result	0.384	No Result
i2c	2	0.58	Inf. loop	No Result	Inf. loop	No Result	Inf. loop	No Result	0.183	No Result
i8	5	2.71	Inf. loop	No Result	0.297	No Result	0.324	No Result	0.305	No Result
mem_ctrl	5	340.49	Inf. loop	No Result	Inf. loop	No Result	15.495	No Result	3.054	No Result
pci_brdge32	5	20.13	47.821	No Result	Inf. loop	No Result	Inf. loop	No Result	6697.215	**out of memory**
pci_spoci_ctrl	5	0.75	Inf. loop	No Result	Inf. loop	No Result	Inf. loop	No Result	**timeout**	No Result
rot	5	0.20	0.078	No Result	0.171	No Result	0.055	No Result	0.057	No Result
s13207	5	1.19	Inf. loop	No Result	Inf. loop	No Result	Inf. loop	No Result	593.289	No Result
s38417	5	0.47	Inf. loop	No Result	Inf. loop	No Result	Inf. loop	No Result	110.990	No Result
s38584	5	29.08	Inf. loop	No Result	Inf. loop	No Result	Inf. loop	No Result	42.375	No Result
s9234	5	0.72	0.313	Wrong Key	Inf. loop	No Result	0.492	No Result	0.496	No Result
sasc	3	0.13	0.091	No Result	Inf. loop	No Result	0.058	No Result	0.052	No Result
systemcaes	5	16.90	Inf. loop	No Result	Inf. loop	No Result	Inf. loop	No Result	62.904	No Result
tv80	5	21.97	Inf. loop	No Result	0.659	No Result	0.663	No Result	0.632	No Result
usb_funct	5	21.98	Inf. loop	No Result	Inf. loop	No Result	Inf. loop	No Result	47.230	No Result
wb_conmax	5	2.70	Inf. loop	No Result	Inf. loop	No Result	Inf. loop	No Result	468.508	No Result

[12] H. M. Kamali et al., "InterLock: An Intercorrelated Logic and Routing Locking," in *Proc. of ICCAD*, pp. 1-9, 2020.

[13] H. M. Kamali et al., "Advances in Logic Locking: Past, Present, and Prospects," in *Cryptology ePrint Archive, Report 2022/260*, 2022.

[14] H. M. Kamali et al., "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.

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

[16] S. Potluri et al., "SeqL: Secure Scan-Locking for IP Protection," in *Proc. of ISQED*, pp. 7-13, 2020.

[17] A. Rezaei et al., "Cyclic Locking and Memristor-based Obfuscation Against CycSAT and Inside Foundry Attacks," in *Proc. of DATE*, pp. 85-90, 2018.

[18] A. Rezaei et al., "CycSAT-Unresolvable Cyclic Logic Encryption Using Unreachable States," in *Proc. of ASPDAC*, pp. 358-363, 2019.

[19] A. Rezaei et al., "Rescuing Logic Encryption in Post-SAT Era by Locking & Obfuscation," in *Proc. of DATE*, pp. 13-18, 2020.

[20] A. Rezaei and H. Zhou, "Sequential Logic Encryption Against Model Checking Attack," in *Proc. of DATE*, pp. 1178-1181, 2021.

[21] A. Rezaei et al., "Global Attack and Remedy on IC-Specific Logic Encryption," in *Proc. of HOST*, pp. 145-148, 2022.

[22] S. Roshanisefat et al., "SRClock: SAT-Resistant Cyclic Logic Locking for Protecting the Hardware," in *Proc. of GLSVLSI*, pp. 153-158, 2018.

[23] S. Roshanisefat et al., "SAT-Hard Cyclic Logic Obfuscation for Protecting the IP in the Manufacturing Supply Chain," *IEEE TVLSI*, vol. 28, no. 4, pp. 954-967, 2020.

[24] J. A. Roy et al., "Ending Piracy of Integrated Circuits," *Computer*, vol. 43, no. 10, pp. 30-38, 2010.

[25] J. P. Roth et al., "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.

[26] A. Saha et al., "LoPher: SAT-Hardened Logic Embedding on Block Ciphers," in *Proc. of DAC*, pp. 1-6, 2020.

[27] A. Saha et al., "DIP Learning on CAS-Lock: Using Distinguishing Input Patterns for Attacking Logic Locking," in *Proc. of DATE*, 2022.

[28] B. Shakya et al., "CAS-Lock: A Security-Corruptibility Trade-off Resilient Logic Locking Scheme," *IACR Trans. on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 1, pp. 175–202, 2019.

[29] K. Shamsi et al., "Cyclic Obfuscation for Creating SAT-Unresolvable Circuits," in *Proc. of GLSVLSI*, pp. 173-178, 2017.

[30] K. Shamsi et al., "AppSAT: Approximately Deobfuscating Integrated Circuits," in *Proc. of HOST*, pp. 95-100, 2017.

[31] K. Shamsi et al., "IcySAT: Improved SAT-based Attacks on Cyclic Locked Circuits," in *Proc. of ICCAD*, pp. 1-7, 2019.

[32] Y. Shen et al., "BeSAT: Behavioral SAT-based Attack on Cyclic Logic Encryption," in *Proc. of ASPDAC*, pp. 657-662, 2019.

[33] Y. Shen et al., "A Comparative Investigation of Approximate Attacks on Logic Encryptions," in *Proc. of ASPDAC*, pp. 271-276, 2018.

[34] Y. Shen et al., "SAT-based bit-flipping attack on logic encryptions," in *Proc. of DATE*, pp. 629-632, 2018.

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

[36] Y. Shen et al., "SigAttack: New High-level SAT-based Attack on Logic Encryptions," in *Proc. of DATE*, pp. 940-943, 2019.

[37] D. Sisejkovic et al., "Challenging the Security of Logic Locking Schemes in the Era of Deep Learning: A Neuroevolutionary Approach," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 17, no. 3, pp. 1-26, 2021.

[38] P. Subramanyan et al., "Evaluating the Security of Logic Encryption Algorithms," in *Proc. of HOST*, pp. 137-143, 2015.

[39] 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.

[40] Y. Xie and A. Srivastava, "Anti-SAT: Mitigating SAT Attack on Logic Locking," *IEEE TCAD*, vol. 38, no. 2, pp. 199-207, 2019.

[41] F. Yang et al., "Stripped Functionality Logic Locking With Hamming Distance-Based Restore Unit (SFLL-hd) – Unlocked," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 10, pp. 2778-2786, 2019.

[42] X.-M. Yang et al., "LOOPLock 2.0: An Enhanced Cyclic Logic Locking Approach," *IEEE TCAD*, vol. 41, no. 1, pp. 29-34, 2022.

[43] M. Yasin et al., "SARlock: SAT Attack Resistant Logic Locking," in *Proc. of HOST*, pp. 236-241, 2016.

[44] M. Yasin et al., "Provably-Secure Logic Locking: From Theory To Practice," in *Proc. of CCS*, pp. 1601-1618, 2017.

[45] M. Yasin et al., "Security Analysis of Anti-SAT," in *Proc. of ASPDAC*, pp. 342-347, 2016.

[46] M. Yasin et al., "Removal Attacks on Logic Locking and Camouflaging Techniques," *IEEE Trans. on Emerging Topics in Computing*, vol. 8, no. 2, pp. 517-532, 2020.

[47] Y. Zhong and U. Guin, "Complexity Analysis of the SAT Attack on Logic Locking," *IEEE TCAD*, pp. 1-1, 2023.

[48] H. Zhou et al., "CycSAT: SAT-Based Attack on Cyclic Logic Encryptions," in *Proc. of ICCAD*, pp. 49-56, 2017.

[49] H. Zhou et al., "Resolving the Trilemma in Logic Encryption," in *Proc. of ICCAD*, pp. 1-8, 2019.

[50] M. Zuzak et al., "Evaluating the Security of Logic-Locked Probabilistic Circuits," *IEEE TCAD*, vol. 41, no. 7, pp. 2004-2009, 2022.