SAT-Hard Cyclic Logic Obfuscation for Protecting the IP in the Manufacturing Supply Chain

Shervin Roshanisefat, Hadi Mardani Kamali, Houman Homayoun, and Avesta Sasan

***Abstract*—State-of-the-art attacks against cyclic logic obfuscation use satisfiability solvers that are equipped with a set of cycle-avoidance clauses. These cycle-avoidance clauses are generated in a preprocessing step and define various key combinations that could open or close cycles without making the circuit oscillating or stateful. In this article, we show that this preprocessing step has to generate cycle-avoidance conditions on all cycles in a netlist; otherwise, a missing cycle could trap the solver in an infinite loop or make it exit with an incorrect key. Then, we propose several techniques by which the number of cycles is exponentially increased as a function of the number of inserted feedback. We further illustrate that when the number of feedback is increased, the preprocessing step of the attack faces an exponential increase in complexity and runtime, preventing the correct composition of cycle-avoidance clauses in a reasonable time. On the other hand, if the preprocessing is not concluded, the attack formulated by the satisfiability solver will either get stuck or exit with an incorrect key. Hence, when the cyclic obfuscation under the conditions proposed in this article is implemented, it would impose an exponentially difficult problem for the satisfiability solver-based attacks.**

***Index Terms*—Logic locking, obfuscation, SAT attack.**

I. Introduction

THE cost of building a new semiconductor fab was estimated to be $5.0 billion in 2015, with large recurring maintenance costs [1], [2], which increases sharply as technology migrates to smaller nodes. Thus, to reduce the fabrication cost, and for economic feasibility, most of the manufacturing and fabrication is pushed offshore [1]. However, many offshore fabrication facilities are considered to be untrusted, which has raised concern over potential attacks in the manufacturing supply chain, with an intimate knowledge of the fabrication process, the ability to modify and expand the design before production, and unavoidable access to the fabricated chips during testing. Hence, fabrication in untrusted fabs has introduced multiple forms of security threats from supply chain including that of overproduction, Trojan insertion, reverse engineering, intellectual property (IP) theft, and counterfeiting [2].

Manuscript received November 15, 2019; accepted December 29, 2019. Date of publication February 10, 2020; date of current version March 20, 2020. This work was supported by the Defense Advanced Research Projects Agency (DARPA) under Grant #FA8650-18-1-7819 of the USA, and in part by the National Science Foundation and the Semiconductor Research Corporation under Award 1718434. *(Corresponding author: Shervin Roshanisefat.)*

Shervin Roshanisefat, Hadi Mardani Kamali, and Avesta Sasan are with the Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA 22033 USA (e-mail: sroshani@gmu.edu; hmardani@gmu.edu; asasan@gmu.edu).

Houman Homayoun is with the Department of Electrical and Computer Engineering, University of California at Davis, Davis, CA 95616 USA (e-mail: hhomayoun@ucdavis.edu).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2020.2968552

To prevent the adversaries from such attacks, researchers have proposed various obfuscation methods for hiding and/or locking the functionality of a netlist [3]–[7]. However, the validity and strength of logic obfuscation to defend an IP against adversaries in the manufacturing supply chain was seriously challenged as researchers demonstrated that deobfuscation attacks leveraging satisfiability (SAT) solvers [8]–[10] combined with signal probability skew (SPS) attacks [11] could break the existing obfuscation schemes (both locking and camouflaging) in a relatively short time. Cyclic obfuscation [12] was another approach that was considered as a defense mechanism against SAT solvers. However, this technique was later broken by CycSAT attack [13]. CycSAT added a preprocessing step to the original SAT attack for detection and avoidance of cycles in the netlist before deploying a SAT attack. In this article, we illustrate that the preprocessing step of CycSAT attack has to process a cycle-avoidance condition for every cycle in the netlist; otherwise, the subsequent SAT attack could get stuck in an infinite loop or returns UNSAT. Hence, the runtime of the preprocessing step is linearly related to the number of cycles in a netlist. Besides, we illustrate that the generation of a cycle-avoidance clause for a netlist of cyclic Boolean nature is far more time consuming than an acyclic Boolean logic.

From this observation, we first propose several mechanisms for cyclification of a noncyclic Boolean netlist. Then, we propose two design techniques by which a linear increase in the number of inserted feedback in a netlist would exponentially increase the number of generated cycles. Since a successful SAT attack on a cyclic circuit requires generation of a per-cycle-avoidance clause and considering that our proposed techniques make the time it takes to generate such avoidance clauses an exponential function of the number of inserted feedback, CycSAT attack faces exponential runtime at its processing step. Hence, when deploying CycSAT, the complexity of the preprocessing of the resulting cyclic netlist goes beyond a reasonable time limit. On the other hand, skipping the prepossessing results in an unsuccessful SAT attack. Hence, cyclic obfuscation, when constructed using the proposed methodology, proves to be a strong defense against the SAT and CycSAT attacks.

Contributions of this article are as follows: 1) we provide a comprehensive background on (both cyclic and acyclic) SAT-resistant logic-encryption solutions; 2) we introduce a new attack that enables a SAT attack to break two recently published logic-locking solutions (i.e., obfuscation using nested or hard cycles); 3) we propose a new cyclic obfuscation solution that makes the number of created (real and dummy) cycles an exponential function of the number of inserted feedback and elaborates how it is as an effective means for breaking cyclic SAT attacks; 4) we propose a timing-aware cyclification algorithm to manage and control the timing overheads of our proposed solution; and 5) we assess the effectiveness of our proposed solutions on several benchmarks using an improved/modified cyclic attack and report the power, performance, and area overhead of our proposed solution.

The rest of this article is organized as follows. In Section II, we cover the background on logic obfuscation. Then, in Section III, we elaborate on the limitation of cyclic attacks and our approach for breaking/preventing these attacks. In Section IV, we introduce our techniques for building an exponential relation between the number of feedback and the number of created cycles in a circuit. We also introduce three mechanisms for building a cyclic Boolean function to further increase the complexity of preprocessing in cyclic attacks. Our experimental results are summarized in Section VI. Section VII concludes this article.

II. Background

Logic obfuscation is the process of hiding the functionality of an IP by building ambiguity or by implementing post-manufacturing means of control and programmability into a netlist. Gate camouflaging [14]–[17] and circuit locking [18], [19] are two of the widely explored obfuscation mechanisms for this purpose. A camouflaged gate is a gate that after reverse engineering (using delayering and lithography) could be mapped to any member of a possible set of gates or may look like one logic gate (e.g., AND), however functionally perform as another (e.g., XOR). In locking solutions, the functionality of a circuit is locked using several key inputs such that only when a correct key is applied, the circuit resumes its expected functionality. Otherwise, the correct function is hidden among many of the 2^K (K being the number of keys) circuit possibilities. The claim raised by such an obfuscation scheme was that to break the obfuscation, an adversary needs to try a large number of inputs and key combinations to extract the correct key, and the difficulty of this process increases exponentially as the number of keys and primary inputs increases. Hence, if enough gates are obfuscated, an adversary faces an unacceptably long time (claimed as years to decades) to break the obfuscation scheme. Note that the availability of scan chains, which is inserted following design for test (DFT) recommended flow, allows an adversary to access combinational logic in each stage of a sequential circuit, load the desired input, execute the stage for one cycle, and readout the output.

The validity and strength of logic obfuscation to defend the IP against adversaries in the manufacturing supply chain was seriously challenged, as researchers demonstrated that the SAT solvers, when formulated according to Algorithm 1, could break the obfuscation (both locking and camouflaging) in a matter of minutes as opposed to the promised claim of years and decades [8], [9]. In this algorithm, $C(X, K, Y)$ refers to the obfuscated circuit that produces output vector Y using input vector X and key vector K, and $C_{\text{BlackBox}}(X)$ refers to the output of the activated circuit for input vector X. As illustrated in Algorithm 1, to employ a SAT attack, the obfuscated circuit is transformed into a circuit SAT problem, in which the SAT solver looks for an input value X for which the obfuscated circuit produces two different outputs for two different input keys. Such input is referred to as a *Distinguishing Input* (DIP) X_{DI}. Each time a new X_{DI} is found, the circuit SAT is updated to make sure that the next two keys that will be found in the next iteration of SAT solver invocation produce the same output for all previously discovered X_{DI}. This is done by building a distinguishing input validation circuit (DIVC) as illustrated in Algorithm 1. When the SAT solver can no longer find X_{DI}, the DIVC circuit contains a complete set of distinguishing inputs. At this point, any key that satisfies the DIVC (by calling a SAT solver on this circuit) is the key to the obfuscated circuit [8], [9], [20].

Algorithm 1 SAT Attack on Obfuscated Circuits

```
1: $DIVC = 1;$
2: $SAT_{circuit} = C(X, K_1, Y_1) \wedge C(X, K_2, Y_2) \wedge (Y_1 \neq Y_2);$
3: **while** $((X_{DI}, K_1, K_2) \leftarrow SAT_F(SAT_{circuit}) = T)$ **do**
4:     $Y_f \leftarrow C_{BlackBox}(X_{DI});$
5:     $DIVC = DIVC \wedge C(X_{DI}, K_1, Y_f) \wedge C(X_{DI}, K_2, Y_f);$
6:     $SAT_{circuit} = SAT_{circuit} \wedge DIVC;$
7: $KeyGenCircuit = DIVC \wedge (K_1 = K_2)$
8: $Key \leftarrow SAT_F(KeyGenCircuit)$
```

A. Acyclic Logic Obfuscation

The revelation of this attack redirected the attention of the researchers to find harder obfuscation schemes that protect acyclic Boolean logic and resist the SAT attack. These methods have targeted a number of weaknesses in the SAT attack and could be categorized into three categories:

1) Weaker Distinguishing Inputs: Original SAT attack was powerful because each DIP could rule out several wrong keys and constrain the key space effectively. The SARLock and Anti-SAT [21], [22] logic-locking methods were proposed to mitigate this vulnerability. In a circuit protected by these solutions, a wrong key produces a wrong output only for one input. This will create a much weaker DIP as each DIP can only rule out one wrong key. Hence, a SAT attack will be reduced to a Brute-force attack as it requires an exponential number of DIPs to find the correct key. A design protected by these mechanisms, regardless of the key used for its activation, behaves very similar to the original design (except for one input). Hence, this group of obfuscation solutions suffers from low output corruption. To increase the output corruption, they could be augmented with other (output corruption-oriented) obfuscation mechanisms. However, by using approximate SAT attack [23], almost all key values for the augmented obfuscation mechanism could be correctly identified.

Further research revealed that these obfuscation techniques are vulnerable to removal [11], Bypass [24] and FALL [25] attacks. In a removal attack, these SAT hard blocks are identified using SPS attack [11] and removed. In a Bypass attack [24], an auxiliary circuit that recovers the wrong output in these locking schemes is created. This attack identifies the input combinations that produce the wrong output for a wrong key; then it adds a bypass circuit to flip the wrong output when that specific input is applied. In FALL attacks, a functional analysis of the circuit will be performed and have two stages. In the first stage, it analyses the functionality of the obfuscated circuit and tries to identify the locking keys. If there was more than one candidate for the locking key, it tries to use the SAT to find the correct locking key from a list of alternatives and using simulations on the unlocked circuit.

2) Increasing Circuit-SAT Complexity: Another feature that makes the SAT attack powerful is the fast execution time of the underlying SAT solver in solving the circuit SAT and extracting DIPs. For locking schemes in this category, the netlist is designed in a way that translates into a large circuit SAT with possibly a SAT-hard portion and thus requires more time to solve. Cross-Lock [26] exploited this vulnerability by adding cross-bars to the netlist and obfuscates circuit connections. Equivalent circuit SAT in this method requires large multiplexers and the symmetric nature of this block will make it a SAT-hard problem [27]–[29]. Without any additional clauses, any SAT solver requires a long execution time to find a single distinguishing input.

Netlists with camouflaged or memory-based blocks could also be used for this purpose. For these blocks, an equivalent circuit should be used that replaces them. For blocks with a large number of input and key ports, the equivalent circuit could be very large. This is especially true in the case of a locked circuit with large LUTs. This could lead to a large circuit SAT with lots of SAT clauses.

3) SAT Unsolvable Structures: SAT attack needs to translate the reverse-engineered netlist into CNF clauses to be able to use the underlying SAT solver. Memory blocks and Boolean gates could be easily translated into CNF clauses using equivalent circuits and Tseitin [30] transformation. Boolean limitation of SAT solvers could be used as a vulnerability to implement non-Boolean structures to counter the SAT attack.

Delay locking [31] is one such method. It uses key gates to lock both the functionality and the timing behavior of the obfuscated circuits. The logic aspect of the locking could be easily translated into CNF, however the behavioral (timing) aspect of circuit operation cannot be easily translated into a SAT-friendly CNF. Hence, formulating a SAT attack on a delay-locked netlist will produce a circuit of correct functionality, but the timing violations will make the circuit malfunctioning. This method could potentially prevent over-production or any reuse of fabrication materials like masks, but it cannot prevent reverse engineering and IP-theft of the design. Also, an attack called TimingSAT [32] was later proposed to break this obfuscation method.

B. Cyclic Logic Obfuscation

Another method that could render SAT solvers ineffective is to invalidate the acyclic nature of netlist by using cyclic logic obfuscation. Cyclic logic obfuscation was first proposed in [12] whereby introducing feedback in the netlist, the netlist is no longer a directed acyclic graph (DAG). In their approach, each intentionally created cycle had more than one way to be opened, making such cycle irreducible by structural analysis, claiming that the existence of such a cycle breaks the original SAT attack in [8] and [9].

Attacks previously proposed for breaking logic-locking solutions are not effective on cyclically obfuscated circuits. The brute-force attack on obfuscated circuits (even those that are not SAT hard) will face exponential difficulty. The sensitization attack would not work on cyclic circuits since the key values control the multiplexers' select line and the select values cannot be sensitized to output pins. The pure SAT attack does not work on cyclic circuits as cycles could either trap the SAT solver or make it exit with an incorrect key, a problem that also occurs in approximate SAT attacks (i.e., AppSAT); the approximate attacks address the issue of separating the keys between SAT hard and conventional obfuscation. Considering that cyclic circuits trap the SAT solver, this group of attack is also would not work. Removal and SPS attacks are aimed at detecting and removing point functions which are used as a means of building SAT hard solutions in the DAG-based network. Considering that the cyclic obfuscation does not use a point function, SPS and removal attacks are not applicable.

Algorithm 2 CycSAT Attack on Cyclic Obfuscated Circuits

```
 1: Find a set of feedback signals $(w_0, w_1, ...w_m)$;
 2: Compute "no structural path" formulas $F(w_0, w_0')$, ..., $F(w_m, w_m')$;
 3: $NC(K) = \wedge_{i=0}^{m} F(w_i, w_i')$
 4: $C(X, K, Y) = C(X, K, Y) \wedge NC(K)$
 5: $SAT_{circuit} = C(X, K_1, Y_1) \wedge C(X, K_2, Y_2) \wedge (Y_1 \neq Y_2)$;
 6: while $((X_{DI}, K_1, K_2) \leftarrow SAT_F(SAT_{circuit}) = T)$ do
 7:     $Y_f \leftarrow C_{BlackBox}(X_{DI})$;
 8:     $DIVC = DIVC \wedge C(X_{DI}, K_1, Y_f) \wedge C(X_{DI}, K_2, Y_f)$;
 9:     $SAT_{circuit} = SAT_{circuit} \wedge DIVC$;
10: $KeyGenCircuit = DIVC \wedge (K_1 = K_2)$
11: $Key \leftarrow SAT_F(KeyGenCircuit)$
```

Cyclic obfuscation was later broken with introduction of cyclic (cycle aware) attacks in [13], [33], and [34]. CycSAT was the first cyclic attack, details of which are shortly discussed. Later, Chen [33] introduced an enhanced SAT attack that considers structural cycles. From a functional standpoint, this attack acts similar to the structural attack in CycSAT.

In a CycSAT attack, before invoking the SAT solver, the netlist is checked for key conditions that may result in the creation of cycles. These conditions are translated into a set of cycle-avoidance clauses and are added to the list of clauses that represent the circuit SAT problem. Algorithm 2 illustrates the flow of utilizing the cycle-avoidance clauses in CycSAT.

In this algorithm, $(w_0, w_1, \ldots, w_m)$ is a collection of feedback signals whose break will make the encrypted circuit acyclic and w_i' is a signal that feeds to w_i before the break. $F(w_i, j)$ is a function that constructs the condition for *having no structural path* between signal w_i to signal j. $F(w_i, j)$ is computed by starting from a feedback signal w_i and constructs a string of clauses that satisfy the following condition while traversing a cycle:

$$F(w_i, j) = \bigwedge_{l \in NK(j)} F(w_i, l) \vee bk(l, j). \tag{1}$$

In this function, $NK(j)$ are the nonkey inputs of signal j and $bk(l, j)$ is the condition on the key, assuring key does not affect j. This function is initiated with condition $F(w_i, w_i) = 0$ and finishes after completing the loop. In this case, the condition for no structural path is tested on all discovered feedback signals in line 3 of the algorithm.

Subsequently, Rezaie *et al.* [35] proposed two solutions to counter a CycSAT attack. In the first solution [35], by adding hard cycles to the original netlist, they create a situation that any traversal of the feedback signals will miss a cycle. Also, for this method, dependent cycles are added to the original circuit such that two nested cycles should be closed to create a working circuit. In the second solution [36], a method is introduced to create cycles that behave noncombinational in unreachable states. However, in the following section, after providing further details on these locking mechanisms, we illustrate that these solutions are still vulnerable and a simple modification to CycSAT attack could easily break them.

Finding all cycles in a cyclic circuit (a requirement for CycSAT attack) is not an easy task. Recently, Shen *et al.* introduced a new attack called BeSAT [34]. The authors of this attack argue that "it is impossible to capture all cycles in any graph with any set of feedback signals as done in CycSAT algorithm." To address this problem, BeSAT first adds "no structural path" (CycSAT-I) conditions for a "set of feedback signals." This is similar to the preprocessing step in

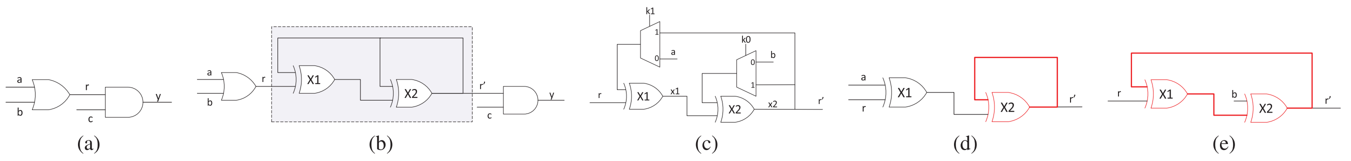


Fig. 1. Cyclification with dependent cycles. (a) Original circuit, (b) cyclified with an auxiliary circuit that acts as a buffer, (c) obfuscated auxiliary circuit, (d) auxiliary circuit with broken outer cycle, and (e) auxiliary circuit with broken inner cycle.

a CycSAT attack. Then, it performs SAT while monitoring the behavior of the attack: during the DIP generation process, due to the missing no cycle (NC) clauses, it is possible that solving the circuit-SAT problem results in repeated DIPs. Under the original SAT attack, this could trap the attack in an infinite loop. In BeSAT, every new DIP is compared with previous DIPs and if it was generated before, the algorithm uses it to determine the stateful key K_s. BeSAT compares the output of the new DIP for the two found keys with the oracle circuit. The output of the stateful key disagrees with the oracle circuit. Then, the found stateful key will be explicitly banned by adding $(K1 \neq K_s \ \wedge \ K2 \neq K_s)$ condition to the circuit-SAT problem. After finding all DIPs and banning all stateful keys, BeSAT begins pruning oscillating keys by employing ternary SAT.

III. Analyzing the Weaknesses of Cyclic Obfuscation

In this section, we first show that the nested cycles could not guarantee a secure cyclic obfuscation. Furthermore, we propose a new attack mechanism to break the hard cycles. Then, we investigate the weaknesses of a CycSAT attack, according to which we propose a new mechanism for cyclic obfuscation.

A. Breaking Nested Cycles

An obfuscation method that was previously proposed to counter a CycSAT attack is the use of nested cycles [35]. In this method, the original circuit is augmented with a pair of nested cycles such that for correct operation, both cycles should be closed. An example of such a transformation is shown in Fig. 1(b) for the original circuit in Fig. 1(a). After the transformation, the nested cycles are a needed and valid part of the original circuit, and attempting to remove one or both cycles will affect the correct functionality of the circuit. A designer may try to obfuscate these cycles using multiplexers as shown in Fig. 1(c).

Direct application of structural CycSAT attacks, as claimed in [35], results in breaking each of the nested cycles separately, creating an oscillating and un-SAT-isfiable circuit. However, as claimed earlier, we can still deploy a successful attack against this variant of cyclic obfuscation using a simple modification to the preprocessing step of CycSAT attack.

For this purpose, during the preprocessing step, in addition to composing the “no sensitizable path” clauses (as proposed in [13]), we compose and include a new set of clauses that consider “reducibility” as an alternative option to opening the loops. In this picture, the cycle could either be opened (using no sensitizable path clauses) or could be reduced using newly added reducible clauses. The *reducible* clauses are defined for possible dependent cycles that implement specific functions between their inputs and outputs. These clauses will be generated for each cycle by pairing it with matching outer cycles. The process of generating the reducible clauses is captured in Algorithm 3. The reduction attack procedure, first, sorts all cycles according to their length and then begins processing

Algorithm 3 Generating *RC* Clauses for Dependent Cycles

```
1:  procedure REDUCTION_ATTACK(circuit $K$)
2:      Find and sort all cycles in $K$ by their length $C = (c_0, c_1, \ldots c_m)$;
3:      for all $c_i$ in $C$ do
4:          $RC(c_i) = \phi$;
5:      for all $c_i$ in $C$ do
6:          if IS_COMB_CYCLE($c_i$) == False then;
7:              $RC(c_i) = RC(c_i) \ \vee \ opened(c_i)$;
8:              while $c_j \leftarrow$ next outer cycle do
9:                  $sub\_circuit \leftarrow$ sub-circuit of closed $c_i$ and $c_j$;
10:                 if IS_COMB_CYCLE($sub\_circuit$) then
11:                     $RC(c_i) = RC(c_i) \ \vee (closed(c_i) \wedge closed(c_j))$;
12:                     $RC(c_j) = RC(c_j) \ \vee (closed(c_i) \wedge closed(c_j))$;
13:             $RC(K) = RC(K) \ \wedge \ RC(c_i)$;

1:  procedure IS_COMB_CYCLE(sub_circuit $S$)
2:      $r, \ r' \leftarrow$ input and output of auxiliary-circuit;
3:      if $SAT(S_{opened} \wedge (r \neq r'))$ then
4:          return False;
5:      else
6:          return True;
```

them from the shortest to the longest cycle. For each cycle, it checks whether the cycle is combinational; if it is not, it tries to find an outer cycle that makes its behavior combinational. In this algorithm, $IS\_COMB\_CYCLE()$ validates whether a subcircuit containing a cycle is combinational or not. For this purpose, the function disconnects the cycles by breaking the feedback into two disconnected wire segments r and r'. Then by using a SAT solver, it checks whether there are any values for the wires that $r \neq r'$. If such a scenario was not found, it classifies the subcircuit as a combinational circuit. Otherwise, it is classified as a noncombinational circuit, according to which the necessary clauses are generated.

This algorithm could be applied to any netlist obfuscated using the auxiliary circuit such as the one in Fig. 1(c). This circuit has two cycles $c_1 = \{\text{X2}\}$ and $c_2 = \{\text{X1, X2}\}$. The smallest cycle c_1 is oscillating and oscillates when $X1$ output is 1 as shown in Fig. 1(d). By considering this cycle as closed and pairing it with its only outer cycle c_2, we will have $RC(c_1) = k_0' \vee (k_0 \wedge k_1)$. The outer cycle c_2 as shown in Fig. 1(e) is also noncombinational and the reducible clauses will be $RC(c_2) = k_1' \vee (k_0 \wedge k_1)$. Thus, by closing both cycles, as shown in Fig. 1(b), it can be derived that $r' = r \oplus r' \oplus r' = r$ and the circuit does act as a buffer with no oscillation. The reducible clause for this circuit will be $RC(K) = (k_0' \vee (k_0 \wedge k_1)) \wedge (k_1' \vee (k_0 \wedge k_1))$ for closing both cycles or opening both cycles since none of them has combinational behavior independently.

It should be noted that these auxiliary circuits could be in the form of partially intercepted cycles, where more than one outer cycle is partially intercepted with another outer cycle. We acknowledge that for partially intercepted cycles, our proposed algorithm would not work, and an alternative algorithm that generates the NC condition by considering the partially intercepted combinational cycles is required.

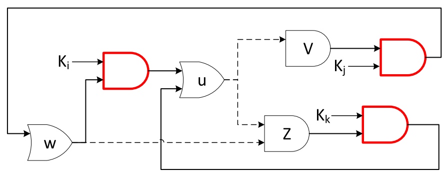


Fig. 2. Example of a circuit obfuscated with a hard cycle. Added key gates are shown in red, and the original wires are shown with dotted lines.

B. Breaking Hard Cycles

Hard cycles were proposed in [35] to create a situation that any traversal of feedback signals will miss a cycle. An example is shown in Fig. 2, where the original circuit consists of gates U, V, W, and Z. In the obfuscated netlist, the gate U is connected to V and Z, and W is connected to Z. By creating a hard cycle, new connections using AND gates have been added. These new wires connect (V, W), (W, U), and (Z, U) and shown with thicker lines. Feedback sets for the new circuit are {V, W} and {Z, U}. Application of a CycSAT attack on this circuit misses the larger cycle {U, V, W, Z, U}, and the attack fails.

Hard cycles could be easily broken by modifying the mechanisms used for the computation of $F(w_i, j)$. $F(w_i, j)$ could be computed in two ways: 1) traversing through a cycle starting from w_i until w_i is visited again and ignoring the cycle break conditions imposed by fan-ins of other nested cycles or 2) traversing through one cycle and adding the cycle break conditions imposed by other nested cycle. As shown for the example in Fig. 2 and the next example, the first choice results in missing some "NC" conditions, leaving cycles in a design that could break the subsequent SAT attack. By choosing the condition 2), we show that it is possible to build the NC condition by visiting all cycles in the netlist without missing any of the hard cycles. To better illustrate this concept, consider the following example: For the obfuscated netlist in Fig. 3 and a topological sort from gate A, the edges E and F are identified as feedback. When following rule 1), and after building the NC condition, we will have

1: $F(F, A) = F(F, F) \vee bk(k_1) = k_1'$
2: $F(F, F') = F(F, A) \vee bk(k_2) = k_1' \vee k_2$
3: $F(E, C) = F(E, E) \vee bk(k_2) = k_2'$
4: $F(E, E') = F(E, C) \vee bk(k_3) = k_2' \vee k_3'$
5: $NC = F(F, F') \wedge F(E, E') = (k_1' \vee k_2) \wedge (k_2' \vee k_3')$.

The problem with this assignment is when $(k_1, k_2, k_3) = (1, 1, 0)$. In this case, the NC condition is satisfied, however the larger nested cycle $\{E, F, G, E\}$ is not broken. Hence, the NC condition would not resolve the cycles if nested or multipath scenarios exist. In this case, if the wrong key $(k_1, k_2, k_3) = (1, 1, 0)$ is chosen by the SAT solver, it will enter a loop. Depending on whether the cycle is oscillating or stateful, the SAT solver will either be trapped in an infinite loop or will exit UNSAT. Note that this infinite loop happens during the execution of the SAT solver and not during the topological sort used in the original SAT attack proposed in [8] and [9].

To avoid the problem imposed by rule 1), we need to follow rule 2) where the key contribution of all fan-ins in all stages are considered. When using rule 2) for building the NC condition for the same circuit, we have

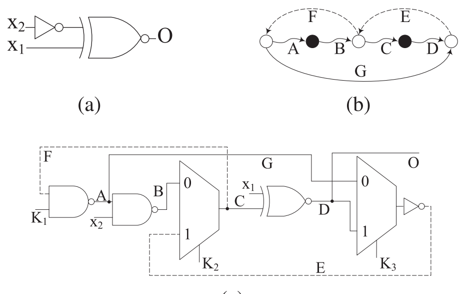


Fig. 3. (a) Original circuit. (b) Flow diagram of the obfuscated circuit. (c) Cyclically obfuscated circuit.

1: $F(F, A) = F(F, F) \vee bk(k_1) = k_1'$
2: $F(F, F') = (F(F, A) \vee bk(k_2)) \wedge (F(F, E) \vee bk(k_2)) = (k_1' \vee k_2) \wedge (k_1' \vee k_3 \vee k_2')$
3: $F(E, C) = F(E, E) \vee bk(k_2) = k_2'$
4: $F(E, E') = (F(E, C) \vee bk(k_3)) \wedge (F(E, G) \vee bk(k_3)) = (k_2' \vee k_3') \wedge (k_2' \vee k_1' \vee k_3)$
5: $NC = F(C, C') \wedge F(E, E') = (k_2' \vee k_3') \wedge (k_1' \vee k_2' \vee k_3) \wedge (k_1' \vee k_2)$.

By following the 2), the previous assignment of keys $(k_1, k_2, k_3) = (1, 1, 0)$ will no longer be a valid assignment, preventing the SAT solver from being stuck or exiting with a wrong key. However, in this case, *all cycles in the design have to be traversed and conditioned*. As a matter of fact, given the way the *NC* is formulated in [13], to derive the "no structural path" condition, some of the combinational cycles (such as $\{E, F, G, E\}$ in Fig. 3) have been visited more than once. Hence, the number of times the key conditions have to be generated is even larger than the number of netlist cycles.

The problem of visiting nested cycles more than once in a CycSAT attack could be resolved by a slight modification to the CycSAT preprocessing step. In the modified attack, instead of applying rule 2) on one-cycle-per feedback, we could apply rule 1) on all cycles. It is intuitive to see that both approaches produce the same NC clauses. For example, in Fig. 3 when following condition 1), and traversing cycle $\{E, F, G, E\}$, the condition $(k_1' \vee k_2' \vee k_3)$ is generated. Hence, by ANDing the generated condition to the two clauses generated by applying rule 1), the NC condition of rule 2) is generated. However, in this case, the combinational cycle $\{E, F, G, E\}$ is only visited once. Even by considering the improvement suggested in CycSAT formulation, it still requires visiting all cycles in a netlist to compose the NC clauses. This necessity, as described in the next section, becomes one of the key features, which is used in this article to break the CycSAT attack.

A different method of introducing complexity is by eliminating the DAG nature of the original netlist and by transforming it into a Boolean cyclic function, which could be represented using a Directed Cyclic Graph (DCG), before subjecting it to cyclic obfuscation. If the original netlist is not a DAG, the CycSAT preprocessing step has to build the NC condition by checking for the "*no sensitizable path*" condition [13], instead of the "*no structural path*" condition. The no sensitizable path condition from [13] is recited in the following equation:

$$F(w_i, j) = \bigwedge_{l \in \text{fan-in}(j)} F(w_i, l) \vee ns(l, j). \tag{2}$$

The "no sensitizable path" condition generates a clause for each multiinput gate in a cycle. As a result, NC clauses are much longer and much weaker. Hence, adding even a small number of feedback to such circuits (that have valid Boolean cycles) for obfuscation will significantly increase the size of the circuit SAT problem, as the "no sensitizable path" condition has to be generated for all cycles. To illustrate the weaker and longer nature of the NC clauses, the "no sensitizable path condition" for the circuit in Fig. 3 is constructed as follows:

1: $F(F, A) = F(F, F) \vee ns(F, A) = k_1'$
2: $F(F, B) = F(F, A) \vee ns(A, B) = k_1' \vee x_2'$
3: $F(F, F') = (F(F, B) \vee ns(B, F')) \wedge (F(F, E) \vee ns(E, F')) = (k_1' \vee x_2' \vee k_2) \wedge (k_1' \vee k_3 \vee k_2')$
4: $F(E, C) = F(E, E) \vee ns(E, C) = k_2'$
5: $F(E, D) = F(E, C) \vee ns(C, D) = k_2'$
6: $F(E, E') = (F(E, D) \vee ns(D, E')) \wedge (F(E, G) \vee ns(G, E')) = (k_2' \vee k_3') \wedge (k_2' \vee k_1' \vee x_2' \vee k_3)$
7: $NC = F(F, F') \wedge F(E, E') = (k_1' \vee x_2' \vee k_2) \wedge (k_1' \vee k_3 \vee k_2') \wedge (k_2' \vee k_3') \wedge (k_2' \vee k_1' \vee x_2' \vee k_3)$.

IV. SRC-Lock: The Proposed Cyclic Obfuscation

The issue with the original method of generating cycle-avoidance (NC) clauses using CycSAT was shown and discussed in Section II-B using two simple examples in which traversal of wires based on a single topological sort of gates resulted in a missing cycle. When using the original CycSAT, because of the missing NC clauses for such cycles and due to the randomness of assigned key and input values by the SAT solver, the SAT attack can be stuck in an infinite loop or exit with a wrong key. The possibility of facing an oscillating or stateful cycle greatly increases as the number of generated cycles in the design increases to a point that majority of key space (to be tested by the SAT solver) could result in oscillating or stateful cycles, vanishing the chances of a successful attack to unreasonably small probability. On the other hand, attacks such as BeSAT [34] that can track the behavior of the SAT attack at runtime, could detect oscillating or stateful scenarios (due to missing cycles in preprocessing time) and eliminate the incorrect key. However, at runtime, BeSAT eliminates one key at a time. Hence, it is successful if the number of such key combinations is small. In other words, the BeSAT attack runtime is linearly dependent on the number of such keys. When such key combinations are (exponentially) large (which is the case in our to-be-proposed obfuscation solution), the BeSAT attack's runtime becomes unacceptably large.

CycSAT preprocessing time is characterized in (3). As illustrated, the processing time is linearly related to the number of discovered cycles N and the time for composing the NC condition t_{NC} per cycle. Our approach for breaking CycSAT is to exponentially increase the time needed for composing the NC condition in the preprocessing step of CycSAT beyond acceptable. This is achieved by exponentially increasing the number of cycles N in a design with respect to the number of inserted feedback m and increasing the time required for processing each cycle (t_{NC}) by forcing the preprocessing step to consider the "no sensitizable path" condition instead of the "no structural path." Next, we provide two solutions for building an exponential relation between the number of feedback and number of generated cycles, and three solutions for converting an acyclic circuit to a valid cyclic circuit

$$T_{\text{NC}} = \sum_{i=1}^{N} t_{\text{NC}} \mid N = 2^m. \tag{3}$$

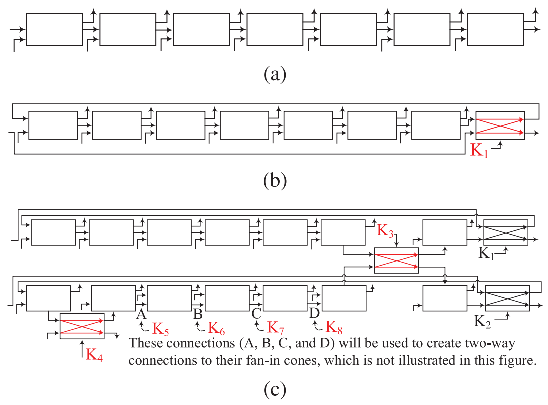

(a)
(b)
K1
(c)
K3
K1
K2
A
B
C
D
K5
K6
K7
K8
K4
These connections (A, B, C, and D) will be used to create two-way connections to their fan-in cones, which is not illustrated in this figure.

Fig. 4. Building an SC from seven gate MCs. (a) Path segment containing seven gates. (b) Building an MC. (c) Building an SC by strongly connecting multiple MCs.

A. Exponentially Increasing the Number of Cycles in a Netlist

In order to exponentially increase the number of cycles in a given netlist with respect to the number of inserted feedback, we introduce two approaches: 1) building super cycles (SCs) and 2) building logarithmic feedback networks (LFNs).

1) Building SCs: The process of building an SC is illustrated in Fig. 4. A microcycle (MC) is a cycle created by following the cycle creation conditions adopted from [12], which are recited in the following.

MC Condition 1: Any created cycle has to be nonreducible.

MC Condition 2: At least $n \geq 2$ edges in each small cycle have to be removable.

A reducible cycle has a single entry point. Hence, the depth-first-search (DFS) traversal of a netlist that only contains reducible cycles is unique. This allows the reducible cycles to be easily opened by removing a unique set of feedback edges which can be found efficiently [12]. By having multiple entries into each MC, the nonreducible condition is satisfied, forcing an adversary to use the CycSAT preprocessing step to generate the necessary cycle-avoidance clauses before invoking the SAT solver. In graph theory, a strongly connected graph is defined as a graph with at least one path between any two pairs of its vertices. Adopting from this definition, in our solution, an SC is defined as a strongly connected graph of MCs. To substantially increase the number of generated cycles, in the last step of SC generation, the edge density of the generated strongly connected graph is increased, creating additional paths between MCs. The process of building an SC is summarized in Algorithm 4.

In this algorithm, the requirement of generating the MCs in the fan-in of the smallest number of primary outputs increases the likelihood of shared and/or connecting edges between created MCs. By having all MCs strongly connected, we create the possibility of larger combinational cycles. And finally, adding the random connections increase the density of the edges in the strongly connected graph, increasing the number of resulting cycles. In Section VI, we illustrate that the number of created cycles, generated from following these steps as

Algorithm 4 Steps for Building an SC

```
1: Construct MCs in the fanin of smallest possible number of primary
   outputs.
2: Strongly connect all generated MCs (this, as illustrated in Fig. 4.b, is
   done by creating a two-way connection between each newly created MC,
   and the existing SC).
3: Select signals in MCs (A, B, C, D in Fig. 4.c) that are not used for SC
   connectivity and provide a two way path from them to unused edges in
   other MCs or random signals in their fanin cone.
```

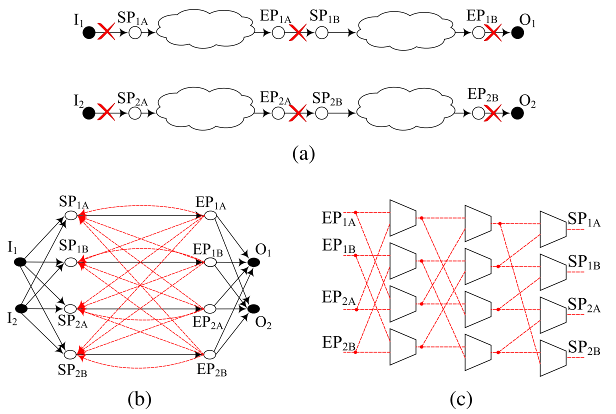


Fig. 5. Building an LFN in which the number of cycles exponentially increase with the number of feedback. (a) Selected logic paths for creating LFN. (b) Selected paths are broken and reorganized to create four smaller logic segments. (c) SP and EP of multiple such logic path segments are used to build a logarithmic switching network.

described in Algorithm 4, becomes an exponential function of the number of inserted feedback.

Lemma: The lower bound on the number of cycles created when using an SC is 2^m, when m is the number of inserted feedback.

Informal Proof: The proposed SC method adds two paths (from and to paths) to connect each new cycle to the existing SC. This way, the new cycle could be added or not added to any of the previously existing cycles. Hence, the addition of a new cycle at least doubles the number of potential cycles. Note that the number of connecting edges between the new cycle and the existing cycle could be more than 1, resulting in an increase in the number of cycles with a much higher rate. From this discussion, after inserting m feedback and connecting them, at least 2^m cycles will be created. ■

2) Building Logarithmic Feedback Networks: In this method, as illustrated in Fig. 5(a), several logic paths (preferably from the fan-in cone of a single primary output) are selected. Then, by breaking a wire in the midpoint of each logic path, we create two smaller logic segments. The signal entering and the signal exiting each half segment is marked as its start point (SP) and endpoint (EP), respectively. Then, the SP and EP of multiple such logic path segments are used to build a logarithmic switching network (e.g., Omega, Butterfly, Benes, or Banyan network). When connecting M number of EPs to M number of SPs, for Ms of power of 2, we need $M(1 + \log_2(M))$ multiplexers for a logarithmic network. In this case, when the correct key is applied, the switching network is configured correctly; otherwise, invalid connectivity obfuscates the netlist functionality.

Lemma: The lower bound on the number of cycles created when using LFN is $\sum_{l=1}^{m}\binom{m}{l}(l-1)!$, when m is the number of inserted feedback and l is the cycle size divided by 2.

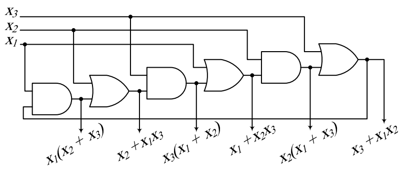


Fig. 6. Three-input Rivest circuit implementing six functions.

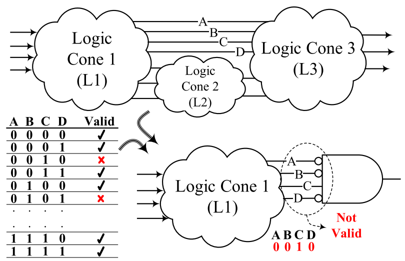
Informal Proof: The proposed LFN is a special case of a complete bipartite graph that contains no odd cycles. Suppose that SE_{ij} indicates a vertex from SP_i to EP_j. Similarly, ES_{ij} indicates a vertex from EP_i to SP_j. For $l = 2$, the cycles are all paths from an SP to its corresponding EP and return path $\{SE_{ii}, ES_{ii}\}$. If we start from SP_i, the second visited node is its EP (EP_i). Since each EP is connected to all SP_s, for intermediate nodes, we have all permutations as alternative possible paths. Cycles with $l = 2$ have no intermediate node. So, there are $\binom{m}{1}0!$ cycles when $l = 2$. For $l = 4$, the cycles are paths like $\{SE_{ii}, ES_{ij}, SE_{jj}, ES_{ji}\}$. There is only one intermediate node in cycles when $l = 4$ resulting in $\binom{m}{2}1!$ cycles. Similarly, for $l = 6$, the cycles are paths like $\{SE_{ii}, ES_{ij}, SE_{jj}, ES_{jk}, SE_{kk}, ES_{ki}\}$. Since we have two intermediate nodes, j and k, we should consider their permutation as a new cycle, that is, $\{SE_{ii}, ES_{ik}, SE_{kk}, ES_{kj}, SE_{jj}, ES_{ji}\}$. So, for $l = 6$, we have $\binom{m}{3}2!$ with a similar relation, for $l = 8$, we have $\binom{m}{4}3!$ cycles. We can extend this relation to all cycles with different length. The summation of these cycles indicates the number of cycles in our logarithmic network, which is $\sum_{l=1}^{m}\binom{m}{l}(l-1)!$ ■

Note that $\sum_{l=1}^{m}\binom{m}{l}(l-1)!$ is the lower bound of the number of simple and nested cycles created by using the logarithmic network. The number of paths from each SP to each EP could be more than 1, and there are possibilities of having a connection between SPs and EPs of the different paths in the original circuit, increasing the number of cycle possibilities to a far larger number. Based on the lower bound formula, the number of created cycles is $O(\sum_{l=1}^{m}\binom{m}{l}(l-1)!) \leq O(m!) = O(m^m)$. Hence, there exists an exponential relation between the number of inserted feedback and the number of resulting cycles in the netlist.

B. Building Cyclic Boolean Functions

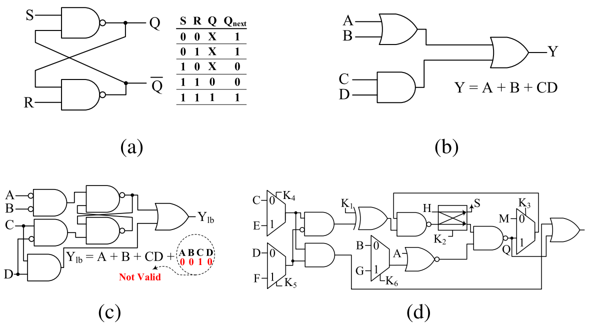
A Boolean function does not need to be acyclic. Furthermore, it is possible to reduce the number of gates in a circuit if a function could be implemented in its acyclic form [37]–[40]. For example, the work in [40] presents an n-input $2n$-output positive unate Boolean function which can be realized with $2n$ two-input gates when the feedback is used but requires $3n - 2$ gates if the feedback is not used. Hence, cyclification of a circuit in addition to forcing a CycSAT preprocessing step to consider the "no sensitizable path," could also remedy the area overhead of introducing new gates for cyclic obfuscation. To cyclify a netlist and to increase t_{NC} in 3, we suggest three approaches: 1) template-based cyclic-function mapping; 2) input-dependence-based cycle generation; and 3) node-merging cycle generation.

1) Template-Based Cyclic-Function Mapping: In this approach, many small cyclic Boolean circuits are collected as templates in our obfuscation library. Then, a netlist is scanned for opportunities (with and without logic manipulation) to replace a cluster of logic gates with such templates. An example of such feedback template is the circuit introduced in [40] where a special case of it (for three inputs) is illustrated in Fig. 6. To introduce cycles, the circuit could be



A	B	C	D	Valid
0	0	0	0	✓
0	0	0	1	✓
0	0	1	0	✗
0	0	1	1	✓
0	1	0	0	✓
0	1	0	1	✗
.	.	.	.	
.	.	.	.	
1	1	1	0	✓
1	1	1	1	✓

Fig. 7. Due to correlation of intermediate signals, certain signal combinations may never occur.



S	R	Q	Q_{next}
0	0	X	1
0	1	X	1
1	0	X	0
1	1	0	0
1	1	1	1

Fig. 8. Input-dependence-based cyclification of a Boolean function. (a) SR latch. (b) Original circuit. (c) Cyclified circuit when $ABCD = 0010$ is nonoccurring. (d) Obfuscated cyclified circuit using additional random inputs E, F, G, H, and M.

modified to introduce at least one of the possible functions in this circuit. The candidate logic cluster is then replaced by the template. To prevent template scanning and removal attacks, in a subsequent camouflaging step (using the gate and route obfuscation), the template will be hidden. Note that many such templates could be made [37]–[40], and by not knowing the template type and the camouflaged technique used to hide the connection, an attacker cannot identify and remove these templates.

2) Input-Dependence-Based Cycle Generation: This method explores the correlations between signals that share common primary inputs in their fan-in cone. Considering N such signals in an arbitrary stage of a DAG, some of the 2^N inputs may never occur. For example, when tracking four signals A, B, C, and D in Fig. 7, we may find that $ABCD = \{0010\}$ could not occur. A SAT solver could be used for finding the nonoccurring input scenarios. This process is illustrated in Fig. 7, where the logic clusters L2 and L3 are removed, and the four signals are ANDed together such that for a certain case, for example, $ABCD = 0010$, the output of AND gate is evaluated to 1. Then, this circuit is given to a SAT solver to find a satisfying input assignment. If the SAT solver returns UNSAT, this combination of input is chosen since it would never happen; otherwise, a different combination is checked.

In the next step, we use a sequential element and tie the discovered nonoccurring input scenario to the state-preserving input of the sequential element. For example, by using an SR latch in Fig. 8(a), if SR = 11 does not happen, Q_{next} is the inverse of input S. Hence, we can build a circuit that ties the discovered nonoccurring input scenario to the SR = 11. For example, let us assume wires A, B, C, and D have a nonoccurring combination $ABCD = 0010$ and these signals construct the signal $Y = A + B + CD$. Fig. 8(c) illustrates the signal Y reconstructed when the nonoccurring combination

Algorithm 5 Timing-Aware Cyclic Obfuscation

```
1:  procedure SWITCH_INSERTION($int\ required\_paths$, $circuit\ K$)
2:      largest_cone ← output port with largest cone;
3:      b = BFS(largest_cone);
4:      while (number of inserted feedbacks < required_paths) do
5:          tail ← pop(b);
6:          if (slack(tail) > delay of a keygate and tail not marked) then
7:              path ← DFS on tail considering slacks;
8:              mark path as selected in the circuit;
9:              add feedback to the path;
10:             update the circuit's timing using STA/EDA;
11:     for (each selected path) do
12:         for (each gate in the path) do
13:             if (slack(gate) > delay of a multiplexer) then
14:                 disconnect gate output;
15:                 insert multiplexer;
16:                 connect gate and multiplexer based on SC/LFN;
17:                 update circuit timing using STA/EDA;
```

of the inputs is tied to SR input of the latch. After generating the cyclic logic, to hide the correlation between input signals, the wire selection is obfuscated. Finally, the SR-latch feedback is obfuscated using a set of multiplexers. This assures that CycSAT can only generate the correct NC clauses if the "no sensitizable path" condition is processed; otherwise, it breaks the SR-latch feedback and invalidates the netlist.

3) Node-Merging-Based Cycle Generation: The third approach for cyclification of a netlist is based on the work in [37] where the logic implication is used to identify cyclifiable structure candidates directly or to create them aggressively in circuits. At its core, the work in [37] introduces active combinational feedback cycles by merging two nodes in the original DAG. To check the validity of the generated cyclic netlist, they use a SAT-based algorithm and validate whether the formed cycles are combinational or not.

V. Timing-Aware Cyclic Obfuscation

During logic locking, each modification to the original netlist affects the timing characteristics of the original circuit. A timing oblivious obfuscation solution could result in changes to the delay of one or more timing critical path(s) (via insertion of key gates), leading to a slower design. In this section, we argue that our proposed obfuscation solution could be designed to be timing aware, minimizing (or removing) the impact of obfuscation on circuit timing. This can be achieved by incorporating a simple static timing analysis (STA) in our obfuscation procedure.

Our proposed solution for timing-aware cyclic obfuscation is presented in Algorithm 5. Both SC and LFN methods (supported in this algorithm) require selection of nonoverlapping logic paths in the circuit for intertwined cycle creation. In our solution, presented in Algorithm 5, we find these nonoverlapping logic paths in the fan-in cone (FIC) of a single primary output. The reason for selecting the logic paths in the same FIC is to take advantage of the existing connections between selected logic subpaths when one subpath is in the FIC of at least one of the gates in the other subpath. This condition results in the generation of many additional cycles, on top of those generated by LFNs or SCs. This is because each feedback could create a cycle when combined with each of the path forward edges. After selection of a logic subpath and before committing to the insertion of a new switch, the netlist is assessed for timing violation. If there is no violation, the cycle is generated and the slack of affected timing paths are updated. Finally, the logic gates in the selected

TABLE I

DESCRIPTION OF ISCAS-85 CIRCUITS USED IN THIS ARTICLE

Circuit	#Gates	#PIs	#POs	Circuit	#Gates	#PIs	#POs	Circuit	#Gates	#PIs	#POs
c432	160	36	7	c1355	546	41	32	c3540	1669	50	22
c499	202	41	32	c1908	880	33	25	c5315	2307	178	123
c880	383	60	26	c2670	1269	233	140	c7552	3513	207	108

subpath are marked as used, removing them from future searches.

Our proposed algorithm selects new logic paths in the FIC of the selected primary output until there are no more viable subpaths. The algorithm could be modified to continue finding new paths by selecting the next primary output candidate that has the largest number of unused gates.

VI. RESULTS

In this section, we analyze the effectiveness of our proposed defense against SAT, CycSAT, and BeSAT attacks. For finding cycles in a netlist (after cyclic obfuscation), we implemented the cycle identification algorithm proposed in [41] using C++. Considering that the source code for BeSAT was not openly available, we implemented the BeSAT attack based on the description in [34] using Yices SAT solver [42]. Our computational platform is a Dell PowerEdge R620 equipped with Intel Xeon E5-2670 and 64 GB of RAM. We used ISCAS-85 benchmarks listed and described in Table I to evaluate our solution and to compare it with the prior work. The timeout limit in our experiments is set to 10 h: if an experiment does not conclude within the timeout limit, its table entry is marked as "t/o." In an experiment, if the netlist is too small for insertion of the number of required feedback, its table entry is marked as "Netlist is Small (NiS)."

A. Exponential Growth in the Number of Cycles

1) Cyclification Using SCs: The number of cycles created in ISCAS-85 benchmarks, when using $N = 1$, 2, 3, 5, 10, and 15 MCs of size 7 (i.e., seven gates in a cycle) for building an SC is reported in Table II. Using curve-fitting techniques, the number of cycles in each netlist is also reported as a function of the number of feedback X, in the form of 2^{mX}, in which m is the netlist-specific exponential acceleration factor. The minimum bound for m (according to the discussion in Section IV-A1) when using SC is one. However, as reported in Table II, the value of m is usually far larger than one, meaning there would be a far larger number of cycles than that expected from the SC-imposed minimum bound.

As illustrated in Table II, increasing the number of feedback exponentially increases the number of cycles, such that with only 15 feedback, the cycles in none of the netlists could be counted in a 10-h limit. Note that the designer can exponentially increase CycSAT attack's preprocessing time, by linearly increasing the number of feedback. For executions resulted in timeout, we also confirmed that initiating CycSAT with incomplete NC clauses traps the SAT solver in an infinite loop. Hence, the attacker cannot complete the preprocessing in a reasonable time, and incomplete preprocessing traps the subsequent invocation of the SAT solver. The area overhead for building the SC in terms of the number of switches depends on the number of MCs and the number of gates in each MC. The area overhead for having various numbers of MCs of seven gates when building an SC is reported in Table III.

TABLE II

NUMBER OF CYCLES REPORTED DURING A CYCSAT ATTACK. THE EXPONENTIAL FITTING FUNCTION IS IN FORM OF $c = 2^{mX}$

Circuit	N=1	N=2	N=3	N=5	N=10	N=15	m
c432	3,384	23,879	$4.6*10^5$	NiS	NiS	NiS	6.3
c499	10	331	1528	$1.4*10^6$	NiS	NiS	4.1
c880	67	1,601	1,903	$5.0*10^6$	t/o	t/o	4.5
c1355	59	636	$5.7*10^5$	$1.9*10^9$	t/o	t/o	6.2
c1908	13	294	12,594	$1.3*10^7$	t/o	t/o	4.8
c2670	273	1,570	8,912	$2.9*10^5$	t/o	t/o	3.6
c3540	1,215	5,991	$8.7*10^5$	$4.9*10^8$	t/o	t/o	5.8
c5315	162	4,869	6,650	$1.2*10^9$	t/o	t/o	6.0
c7552	11	124	1,558	$2.6*10^5$	$1.2*10^9$	t/o	3.0

TABLE III

PERCENTAGE OF AREA OVERHEAD FOR SC CREATION WHEN USING DIFFERENT NUMBER OF MCs (N) OF LENGTH 7.

Circuit	N=1	N=2	N=3	N=5	N=10	N=15	N=20
	Area Overhead Percentages (%)						
c432	7.50	13.75	20.00	NiS	NiS	NiS	NiS
c499	5.94	10.89	15.84	25.74	NiS	NiS	NiS
c880	3.13	5.74	8.36	13.58	26.63	39.69	52.74
c1355	2.20	4.03	5.86	9.52	18.68	27.84	37.00
c1908	1.36	2.50	3.64	5.91	11.59	17.27	22.95
c2670	0.95	1.73	2.52	4.10	8.04	11.98	15.92
c3540	0.72	1.32	1.92	3.12	6.11	9.11	12.10
c5315	0.52	0.95	1.39	2.25	4.42	6.59	8.76
c7552	0.34	0.63	0.91	1.48	2.90	4.33	5.75

2) Cyclification Using Logarithmic Feedback Networks: As discussed and proved in Section IV-A2, the lower bound on the number of generated cycles, when the LFN method for cyclification flow is adopted, is an exponential function of the number of feedback. Furthermore, similar to an SC, the edge density of the original netlist may substantially increase the number of created cycles. This is because of the gates with fan-outs greater than 1 in selected logic path segments. If the output of a gate in the LFN is connected to the input(s) of another gate(s) in the same network, the resulting net counts as an additional forward path. Then, each forward path could be matched with a feedback, resulting in an additional cycle. Considering that path segments are selected from the FIC of the same primary output, there exist many such connections (forward edges), resulting in the generation of a far larger number of cycles than the guaranteed minimum bound expected from using LFN. To illustrate this, both the number of created cycles for each benchmark and the theoretical lower bound (calculated using $\sum_{l=1}^{N}\binom{N}{l}(l-1)!$ as proved in Section IV-A2) is reported in Table IV. As illustrated, for most of the obfuscated benchmarks with an LFN larger than 4, cycle enumeration results in timeout after 10 h due to the exponential number of created cycles. This indicates an exponential runtime at the CycSAT preprocessing stage. The area overhead for creating LFNs of different sizes (different number of input paths) is reported in Table V. Note that in both SCs and LFNs, the area overhead scales with the number of inserted feedback and not the size of the circuit. Hence, the area overhead is smaller in larger circuits.

Capturing the power overhead of cyclic obfuscation is more involved. The leakage component of power overhead is a function of the area overhead of the obfuscation solution,

TABLE IV

NUMBER OF CYCLES REPORTED DURING A CYCSAT ATTACK USING LFN METHOD. N IS THE NUMBER OF SELECTED PATHS FOR CREATING LFNS

	N=2	N=4	N=8	N=16	N=32
Lower Bound	3	24	16072	3.8×10^{12}	2.3×10^{32}
c432	26,578	NiS	NiS	NiS	NiS
c499	192	278,577	1.3×10^{10}	t/o	NiS
c880	8,836	4.5×10^{8}	t/o	t/o	NiS
c1355	8.3×10^{6}	t/o	t/o	t/o	NiS
c1908	8.4×10^{7}	t/o	t/o	t/o	t/o
c2670	1.2×10^{7}	t/o	t/o	t/o	t/o
c3540	8.5×10^{9}	t/o	t/o	t/o	t/o
c5315	1.2×10^{9}	t/o	t/o	t/o	t/o
c7552	2.9×10^{9}	t/o	t/o	t/o	t/o

TABLE V

PERCENTAGE OF AREA OVERHEAD FOR AN INSERTED LFN FOR DIFFERENT NUMBER OF SELECTED PATHS (N)

Circuit	N=2	N=4	N=8	N=16	N=32
	Area Overhead Percentages (%)				
c432	5.00	NiS	NiS	NiS	NiS
c499	3.96	11.88	27.72	79.21	NiS
c880	2.09	6.27	14.62	41.78	NiS
c1355	1.47	4.40	10.26	29.30	NiS
c1908	0.91	2.73	6.36	18.18	43.64
c2670	0.63	1.89	4.41	12.61	30.26
c3540	0.48	1.44	3.36	9.59	23.01
c5315	0.35	1.04	2.43	6.94	16.64
c7552	0.23	0.68	1.59	4.55	10.93

and threshold voltage (VT) of inserted multiplexers. Using a high-VT switch cell reduces the leakage impact, however it introduces additional delay [43]. In a simple implementation where standard cells are selected from a single VT, the increase in the leakage power is similar to the increase in the area. The dynamic power consumption, on the other hand, depends on the switching activity of the inserted switches. After proper activation, the switching activity of the inserted multiplexers depends on the toggling rate of the correct input net to the multiplexer. The net toggling activity, in turn, depends on the level of controllability of that net and the probable input scenario to the netlist. The power consumption of both the LFN and SC-based solutions of size $N = 16$ is provided in Table VI. However, note that the power consumption could improve (at the expense of timing and security) by modifying the SC or LFN algorithm to choose nets with small toggling rate to reduce the overhead of dynamic power consumption.

B. SAT, CycSAT, and BeSAT Attack Resilience

Table VII captures the result of SAT, CycSAT, and BeSAT attacks on ISCAS-85 benchmarks that are obfuscated using our proposed solution. For generating the data in this table, we prepared three sets of obfuscated benchmarks. The first set of benchmarks is obfuscated with only two MCs using the SC approach for the obfuscation method. This group of obfuscated benchmarks represents cyclification with a small number of dummy cycles, with no real cycles. The netlists in the second set are first obfuscated using 10 SR-latches (by using the input-dependence-based obfuscation as described in Section IV-B2) and then are cyclified by inserting two MCs. The second group

TABLE VI

POWER OVERHEAD OF SCS AND LFNS OF SIZE $N = 16$

Circuit	SC (N=16)		LFN (N=16)	
	Switching (%)	Leakage (%)	Switching (%)	Leakage (%)
c432	NiS	NiS	NiS	NiS
c499	NiS	NiS	212.64	75.13
c880	38.09	44.85	56.67	38.82
c1355	12.79	32.77	13.26	24.6
c1908	8.42	19.1	13.38	15.66
c2670	14.14	15.96	13.17	12.32
c3540	8.76	10.79	3.86	8.88
c5315	5.75	8.51	6.13	6.7
c7552	2.88	5.78	7.79	4.4

represents the case where there are some real cycles in the design, while the total number of cycles is still small. The third group is similar to the second group, however the number of inserted MCs is increased to 15. It represents obfuscated solutions with both real and exponentially large number of dummy cycles. The results of running SAT, CycSAT, and BeSAT is captured in Table VII. For c432 and c499, generating a large number of MCs (15) was not possible, and hence the largest number of possible MCs were used in the generation of SC.

The first group introduces a small number of removable cycles. As reported in Table VII, even the existence of simple cycles traps the original SAT attack in an infinite loop in most cases (except for two benchmarks that SAT solver luckily chooses a sequence of inputs that avoid or exit the trap). However, CycSAT, when uses the “no structural path” condition (CycSAT-I) for generating the cycle-avoidance clauses, easily breaks all obfuscated netlists. As illustrated in this table and predicted in (3), CycSAT runtime (which includes the runtime for both preprocessing step and SAT solver’s invocation) almost linearly varies with the number of cycles in each netlist.

For the second group, where the original circuit is also cyclified (using real cycles), the usage of CycSAT-I returns UNSAT as it produces NC clauses that breaks the real Boolean cycles. However, when CycSAT uses the “no sensitizable path” conditions (CycSAT-II), it breaks the obfuscation in all cases. Most notable in this data is the increase in the runtime of CycSAT attack (when compared to the first group) as the time it takes to compose the NC condition for each cycle based on “no sensitizable path” condition is longer. This validates the impact of logic cyclification on the runtime of CycSAT attack. Another attack possibility is the BeSAT attack. However, the BeSAT attack should be slightly modified: considering that the design contains real Boolean cycles, the “no sensitizable path” condition (instead of “no structural path” in the BeSAT attack as described in [34]) should be used for the generation of the NC clauses. Hence, the attack could be carried by generating a set of NC clauses (given a deadline) and then use BeSAT to attack the obfuscation and recover from oscillating and stateful cycle conditions. To model this attack, we set “no sensitizable path” preprocessing deadline to 2 h, and BeSAT attack time to 8 h (total of 10 h attack time). As shown for “$N = 2$ + SR-L = 10,” all but one deobfuscation was successful and, in general, BeSAT underperforms compared to the CycSAT-II attack. This is because there exist a small number of cycles, and both CycSAT-II and BeSAT have found and conditioned all cycles, however, BeSAT due to the runtime monitoring of DIPs is slower compared to CycSAT-II attack.

TABLE VII

SAT ATTACK, CYCSAT, AND BESAT EXECUTION TIME AFTER INSERTION OF AN SC ($N = 2$), INSERTION OF AN SC AND 10 SR-LATCHES ($N = 2 +$ SR-L $= 10$), AND INSERTION OF 15 MCS AND 10 SR-LATCHES ($N = 15 +$ SR-L $= 10$)

Circuit	N=2			N=2 + SR-L=10					N=15 + SR-L=10				
	SAT	#Cycles	CycSAT-I	SAT	#Cycles	CycSAT-I	CycSAT-II	BeSAT	SAT	#Cycles	CycSAT-I	CycSAT-II	BeSAT
c432	Inf	23,879	2.56s	Inf	1.65×10^5	UNSAT	11.69s	35.48s	Inf	t/o	UNSAT	t/o	t/o
c499	0.56s	236	0.10s	Inf	397	UNSAT	0.11s	0.79s	Inf	t/o	UNSAT	t/o	t/o
c880	Inf	1,601	0.24s	Inf	7.87×10^6	UNSAT	793.12s	t/o	Inf	t/o	UNSAT	t/o	t/o
c1355	Inf	636	0.12s	Inf	5.00×10^5	UNSAT	53.21s	134.56s	Inf	t/o	UNSAT	t/o	t/o
c1908	0.28s	294	0.10s	Inf	6,467	UNSAT	0.73s	170.74s	Inf	t/o	UNSAT	t/o	t/o
c2670	Inf	1,570	0.23s	Inf	7,412	UNSAT	0.92s	17.22s	Inf	t/o	UNSAT	t/o	t/o
c3540	Inf	5,991	0.75s	Inf	6,026	UNSAT	0.75s	22.67s	Inf	t/o	UNSAT	t/o	t/o
c5315	Inf	4,869	0.61s	Inf	2.59×10^5	UNSAT	26.04s	370.08s	Inf	t/o	UNSAT	t/o	t/o
c7552	Inf	124	0.189s	Inf	164	UNSAT	0.19s	18.30s	Inf	t/o	UNSAT	t/o	t/o

Finally, for the third group, where the number of inserted feedback is increased to 15, all three attacks fail. CycSAT-I is not applicable, as it will open real cycles, resulting in netlist malfunction, and even if preprocessing of this attack finishes, it will exit as UNSAT. CycSAT-II fails as it cannot finish the preprocessing on time. Note that by increasing the number of feedback, the designer can easily and exponentially increase the required preprocessing time unreasonably long. The remaining attack possibility is the BeSAT attack. In this case, the preprocessing of NC clauses is carried until the time limit (2 h) and then BeSAT attack is carried out. Note that in this condition, BeSAT starts the SAT attack with a partial set of clauses generated in the preprocessing step. However, as illustrated in Table VII, BeSAT will reach the deadline after invalidating 100 s of thousands of keys. This is when there exist millions (or larger) other keys that cause oscillating behavior which BeSAT has not yet examined and pruned (one at a time) in the time limit.

As explained in Section IV, BeSAT only works when the number of undetected cycles (and unconditioned keys) is small. The BeSAT attack is slow and eliminates one incorrect key at a time. This is when, in our proposed obfuscation solution, there exists an exponentially large number of invalid keys even after partial preprocessing: As a part of our obfuscation solution (and to create real cycles), we are using (diffused) SR latches. To prevent stateful behavior, through careful input logic section (as described in Section IV-B2), we ensure that the value of SR input cannot evaluate to 11 (condition for statefulness). For this purpose, the input logic cone to S and R input is constructed by exploiting the interdependency of selected wires in the netlist. However, the selection of inputs is further hidden through routing obfuscation. In this case, only with the application of the correct key, the interdependence of the input wires will render the SR-latch nonstateful (by skipping the 11 input). Let us assume $S = g(K_1, X)$ and $R = f(K_2, X)$, where the g and f are the logic representing the input cone of S and R input to the SR latch, K_1 and K_2 are the key gates in the fan-in cone of S and R, and the X is the choice of primary input. In this scenario, any choice of K_1, K_2, and X that could make the SR $= 11$ will result in a stateful circuit. From this analysis, the worst case scenario for BeSAT is a function of the size of primary input X, and key selection K_1 and K_2 for which the wire S and R evaluate to 1, which is an exponential function of the key length $K = (K_1 \bigcap K_2)$. Considering that our solution builds a strongly connected graph, the FIC of S and R could span to all the key gates. Hence, the number of invalid keys that should be banned is exponentially large. Considering this discussion, and for a large number of key combinations that should be banned (one at a time), as shown in the results for "$N = 15 +$ SR-L $= 10$," BeSAT attack does not work against our proposed solution.

C. SAT, CycSAT, and BeSAT Resiliency of Previous Methods

In this section, we study the effectiveness of previously proposed cyclic logic solutions and compare them with our proposed solution. To attack the prior art solutions, we use the modified CycSAT attack as described and formulated in Section III-B. The modified CycSAT attack works similar to the original CycSAT attack, however instead of composing the NC clauses per detected feedback, it composes the NC clauses per detected cycle.

The original cyclic-locking method was introduced in [12] where authors proposed inserting multiplexers in the circuit to create cycles. This obfuscation solution attempts to create irreducible cycles. This method can only create dummy cycles as it does not affect the DAG nature of a combinational netlist and is referenced in this article as *glsvlsi17*. The second method discussed here [35] considers CycSAT attack and tries to defeat CycSAT-I using an auxiliary circuit. This method was discussed in Section III-A. By adding the proposed auxiliary circuits to a design, real cycles are formed, converting the DAG nature of the netlist into a DCG. The netlist is then augmented with additional dummy cycles (similar to the glsvlsi17 method), making the netlist to contain both real and dummy cycles. In this article, we use the name *date18* to refer to this cyclic obfuscation solution.

To assess the effectiveness of prior art solutions, we modeled each of glsvlsi17 and date18 to obfuscate the ISCAS-85 benchmarks. To compare the evaluation results of prior art to that of our proposed solution (in Table VII), the glsvlsi17 method is implemented using 15 randomly selected feedback of length 7, while the benchmarks prepared using date18 solution are obfuscated using the same number of feedback (15) and 10 real cycles (for DAG to DCG transformation), implemented using the auxiliary circuit as described in [35]. For smaller benchmarks, where insertion of this many feedback was not feasible, we have inserted the largest feasible number of feedback. To show the effectiveness of our solution in increasing the runtime of the CycSAT preprocessing step, we have also evaluated the number of generated cycles for

TABLE VIII
SAT ATTACK, MODIFIED CYCSAT, AND BESAT RESULTS FOR EVALUATION OF GLSVLSI17 METHOD [12]

Circuit	#Cycles	SAT		CycSAT-I		BeSAT	
		Time	Iteration	Time	Iteration	Time	Banned
c432	32	t/o	-	0.02	1	0.45	0
c499	282	t/o	-	0.05	1	0.88	0
c880	36	1.35	61	0.13	15	3.59	0
c1355	t/o	t/o	-	t/o	-	7220.83	3
c1908	1,625	t/o	-	0.95	83	8.44	0
c2670	129	t/o	-	2.26	19	55.11	0
c3540	606	0.63	41	0.70	14	10.03	0
c5315	4,216	1.7	33	1.19	45	32.75	0
c7552	1,117	2.35	105	1.77	73	43.24	0

TABLE IX
EVALUATING DATE18 OBFUSCATION [35] AGAINST SAT, CYCSAT, AND BESAT

Circuit	#Cycles	SAT		CycSAT-I		CycSAT-II		BeSAT	
		Time	Iteration	Time	Iteration	Time	Iteration	Time	Banned
c432	62	t/o	-	0.02	UNSAT	0.3	18	9.19	0
c499	1,157	t/o	-	0.06	UNSAT	0.14	18	2.00	0
c880	56	t/o	-	0.04	UNSAT	0.31	23	5.11	0
c1355	t/o	t/o	-	t/o	-	t/o	-	7268.77	12
c1908	1,645	1.99	144	0.02	UNSAT	0.88	68	205.02	0
c2670	149	t/o	-	0.03	UNSAT	0.53	41	10.53	0
c3540	626	6.49	187	0.1	UNSAT	1.53	37	18.19	0
c5315	4,236	t/o	-	0.05	UNSAT	2.06	60	30.45	0
c7552	1,137	t/o	-	0.08	UNSAT	1.9	31	40.75	0

each of the prior cyclic obfuscation (glsvlsi17 and date18) solutions.

Table VIII captures our evaluation results for glsvlsi17 when attacked using SAT, CycSAT-I, and BeSAT. As expected the success of SAT attack on selected benchmarks is random, as generated cycles could trap the SAT solver. Note that by increasing the number of feedback, the chances of trapping the SAT solver increases. CycSAT-I breaks the obfuscation and finds the key to all but one obfuscated benchmark. For c1355, cycles could not be processed within the 10-h time limit, and the attack is timed out. But this case is a great showcase to see the power of BeSAT. As expected, BeSAT could also break this obfuscation. Considering that the preprocessing for most of the benchmarks could be done in less than 2 h, and all cycles could be found for such small obfuscations, the number of banned keys for all cases but one is zero. For this reason and for the additional overhead of runtime monitoring of SAT execution time, the BeSAT takes longer than CycSAT-I. The only interesting scenario is for c1355, where the CycSAT-I is timed out and cannot finish the preprocessing of all cycles. In this case, the incomplete set of NCs is used in BeSAT, and with only three banned keys, BeSAT skips the traps and finds the correct key. Note that the reason why BeSAT does work is that the number of oscillating keys generated in this obfuscation solution is small. This is unlike our proposed solution that there exists an exponentially large number of such keys, and if given to BeSAT, they have to be eliminated one at a time.

Table IX captures evaluation results for the date18 method. Aware of the shortcomings of glsvlsi17, the date18 solution was proposed as a CycSAT-resistant obfuscation solution. The proposed auxiliary circuit by itself has a minimal impact on the number of cycles. However, this method is expected to have a larger number of stateful cycles, and when the original SAT attack used there are higher chances for trapping the SAT solver in an infinite loop. The results in Table IX support this hypothesis, as only two benchmarks are successfully attacked using the base SAT attack. When attacked using CycSAT-I, the date18 solution remains resistant as the preprocessing step of CycSAT-I incorrectly opens the real cycles during NC clause generation. However, when the modified CycSAT-II attack, as described in Section III, is deployed, it could easily break all instances of obfuscated solutions except c1355 (that could not be preprocessed in a reasonable time for having a very large number of cycles). However, in the case of BeSAT and after limiting the preprocessing time to 2 h, the key for c1355 could be recovered in 68.77 s after 2 h of NC clause generation. Other benchmarks that previously was broken by CycSAT-II are also broken by BeSAT with zero banned keys since the generated NC clauses cover all undesirable cycle conditions. Note that for this attack, the NC clauses for BeSAT are generated using the "no sensitizable path" condition; otherwise, the attack will return as UNSAT.

Comparing the glsvlsi17 and date18 data in Tables VIII and IX with that of our proposed solution in Table VII illustrates the effectiveness of our solution: none of the obfuscated netlists using our solution could be broken by the SAT, CycSAT-I, CycSAT-II, or BeSAT (original and modified) attacks, as it includes a solution to trap both the SAT solver and preprocessing step of CycSAT/BeSAT. Note that, when deploying the SAT or CycSAT attack to break glsvlsi17 or date18, the runtime, in addition to the number of inserted feedback, also depends on the selection of feedback. Hence, a random selection of feedback in glsvlsi17 and date18 results in considerable variation in the attack time. Therefore, these solutions, unlike our proposed solution, cannot guarantee a monotonic increase in the runtime of the attack as the number of randomly selected feedback increases. Note that in our solution, the runtime is dominated by CycSAT's or BeSAT's preprocessing step, and this runtime is linearly dependent on the number of cycles, and the number of cycles is an exponential function of the number of inserted feedback. Hence, we can guarantee a monotonic increase in the overall runtime of the attack against our proposed solution as the number of inserted feedback increases.

D. Timing-Aware Cyclification

As described in Section V, inserting logic gates in timing-critical paths would increase the critical path of the netlist resulting in a performance penalty. To minimize the performance penalty to the extent possible, we proposed a timing-aware cyclic obfuscation flow in Section V. This solution would only affect the timing if it can no longer use noncritical timing paths for feedback insertion.

Table X captures the result of our proposed timing-aware cyclic obfuscation when allowing 0% and 5% delay overhead for cyclic obfuscation. Using this delay constraint, the algorithm tries to insert the maximum number of feasible feedback in each benchmark using the SC solution proposed in Section IV-A1. In this table, we have provided a measure of the maximum number of MCs that could be implemented in each benchmark for building a strongly connected graph before running out of usable gates. The key count is the sum of the number of key values needed for managing the MCs and the number of key values needed for managing the additional multiplexers (used for creating outgoing edges from internal

TABLE X

TIMING-AWARE OBFUSCATION RESULTS FOR THE SC METHOD. MAXIMUM NUMBER OF MCs ARE INSERTED FOR 0% AND 5% OVERHEAD OVER TIMING SLACK

Circuit	Slack = 5%				Slack = 0%				
	#Cycles	SAT(s)	#Keys	#MCs	#Cycles	SAT(s)	#Keys	#MCs	Area %
c432	303,476	0.14	15	2	NiS	NiS	NiS	NiS	NiS
c499	NiS	NiS	NiS	NiS	NiS	NiS	NiS	NiS	NiS
c880	t/o	t/o	95	18	t/o	0.23	51	12	26.63
c1355	t/o	t/o	109	23	2,766	0.55	25	8	9.16
c1908	t/o	t/o	187	38	t/o	t/o	111	24	25.23
c2670	t/o	t/o	335	70	t/o	t/o	244	53	38.46
c3540	t/o	t/o	378	75	t/o	t/o	274	57	32.83
c5315	t/o	t/o	448	110	t/o	t/o	446	95	38.66
c7552	t/o	t/o	729	183	t/o	t/o	632	158	35.98

gates in each MC). As illustrated, the maximum number of MCs and key values is a function of the netlist size and the acceptable delay overhead. Note that in larger benchmarks, even without incurring a time penalty, we can insert a large number of MCs, pushing CycSAT attack to be trapped in its preprocessing step until timeout. In addition, note that with 10 MCs, our C++ implementation of preprocessor cannot finish counting the number of generated cycles, and according to SC and LFN lemmas proved in Sections IV-A1 and IV-A2, the number of generated cycles exponentially grows with each added feedback. Hence, we can make the attack time unreasonably long with no or limited timing impact.

The number of MCs and the number of gates in each MC (e.g., cycle length) could affect the number of created cycles and defines the SAT resiliency of the circuit. Parameters like targeted frequency and area overheads should also be considered during cyclic obfuscation. However, this could create a tradeoff on how SAT resilient a circuit is versus how efficiently it could be implemented.

VII. CONCLUSION

In this article, we proposed a new mean of cyclic obfuscation that is immune to SAT, CycSAT, and BeSAT attacks. To make the preprocessing step of CycSAT and BeSAT attacks ineffective, we proposed two mechanisms (SCs and LFNs) for exponentially increasing the number of generated cycles with respect to the number of inserted feedback. In addition, we proposed three mechanisms to cyclify the circuit with real cycles (Cyclic Boolean Logic). The addition of real cycles forces an attacker to generate the "no sensitizable path" conditions during the preprocessing step of the CycSAT or BeSAT attacks, which is considerably more time consuming than "no structural path" generation. The exponential increase in the number of feedback prevents the attacker from generating NC conditions for all cycles in a reasonable amount of time. This breaks the CycSAT attack. The BeSAT attack can proceed to its SAT stage with an incomplete set of NC clauses, however, it has to ban remaining invalid keys one at a time, and there exists an exponentially large number of such keys. Hence, it also fails to break the proposed solution.

REFERENCES

[1] DIGITIMES. (2013). *Trends in the Global IC Design Service Market*. Accessed: 2013. [Online]. Available: http://www.digitimes.com/news/a20120313RS400.html?chid=2

[2] U. Guin, D. Forte, and M. Tehranipoor, "Anti-counterfeit techniques: From design to resign," in *Proc. 14th Int. Workshop Microprocessor Test Verification*, Dec. 2013, pp. 89–94.

[3] K. Zamiri Azar, H. Mardani Kamali, H. Homayoun, and A. Sasan, "Threats on logic locking: A decade later," in *Proc. Great Lakes Symp. VLSI (GLSVLSI)*. New York, NY, USA: ACM, 2019, pp. 471–476.

[4] M. Yasin and O. Sinanoglu, "Evolution of logic locking," in *Proc. IFIP/IEEE Int. Conf. Very Large Scale Integr. (VLSI-SoC)*, Oct. 2017, pp. 1–6.

[5] S. Keshavarz, C. Yu, S. Ghandali, X. Xu, and D. Holcomb, "Survey on applications of formal methods in reverse engineering and intellectual property protection," *J. Hardw. Syst. Secur.*, vol. 2, no. 3, pp. 214–224, Sep. 2018.

[6] S. Roshanisefat, H. Mardani Kamali, and A. Sasan, "SRCLock: SAT-resistant cyclic logic locking for protecting the hardware," in *Proc. Great Lakes Symp. VLSI (GLSVLSI)*. New York, NY, USA: ACM, 2018, pp. 153–158.

[7] K. Z. Azar *et al.*, "COMA: Communication and obfuscation management architecture," in *Proc. 22nd Int. Symp. Res. Attacks, Intrusions Defenses (RAID)*. Berkeley, CA, USA: USENIX, Sep. 2019, pp. 181–195.

[8] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2015, pp. 137–143.

[9] M. E. Massad, S. Garg, and M. Tripunitara, "Integrated circuit (IC) decamouflaging: Reverse engineering camouflaged ICs within minutes," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2015.

[10] S. Roshanisefat, H. K. Thirumala, K. Gaj, H. Homayoun, and A. Sasan, "Benchmarking the capabilities and limitations of SAT solvers in defeating obfuscation schemes," in *Proc. IEEE 24th Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2018, pp. 275–280.

[11] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Security analysis of anti-SAT," in *Proc. 22nd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2017.

[12] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "Cyclic obfuscation for creating SAT-unresolvable circuits," in *Proc. Great Lakes Symp. VLSI (GLSVLSI)*. New York, NY, USA: ACM, 2017, pp. 173–178.

[13] H. Zhou, R. Jiang, and S. Kong, "CycSAT: SAT-based attack on cyclic logic encryptions," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2017, pp. 49–56.

[14] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA: ACM, 2013, pp. 709–720.

[15] R. P. Cocchi, J. P. Baukus, L. W. Chow, and B. J. Wang, "Circuit camouflage integration for hardware IP protection," in *Proc. 51st Annu. Design Autom. Conf. Design Autom. Conf. (DAC)*, Jun. 2014, pp. 1–5.

[16] B. Erbagci, C. Erbagci, N. E. C. Akkaya, and K. Mai, "A secure camouflaged threshold voltage defined logic family," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2016, pp. 229–235.

[17] M. Li *et al.*, "Provably secure camouflaging strategy for IC protection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 8, pp. 1399–1412, Aug. 2019.

[18] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2017, pp. 1601–1618.

[19] H. Mardani Kamali, K. Zamiri Azar, K. Gaj, H. Homayoun, and A. Sasan, "LUT-lock: A novel LUT-based logic obfuscation for FPGA-bitstream and ASIC-hardware protection," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2018, pp. 405–410.

[20] K. Zamiri Azar, H. Mardani Kamali, H. Homayoun, and A. Sasan, "SMT attack: Next generation attack on obfuscated circuits with capabilities and performance beyond the SAT attacks," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2019, no. 1, pp. 97–122, Nov. 2018.

[21] M. Yasin, B. Mazumdar, J. J. V. Rajendran, and O. Sinanoglu, "SAR-Lock: SAT attack resistant logic locking," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2016, pp. 236–241.

[22] Y. Xie and A. Srivastava, "Mitigating SAT attack on logic locking," in *Proc. Int. Conf. Cryptograph. Hardw. Embedded Syst.* Berlin, Germany: Springer, 2016, pp. 127–146.

[23] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2017, pp. 95–100.

[24] X. Xu, B. Shakya, M. M. Tehranipoor, and D. Forte, “Novel bypass attack and BDD-based tradeoff analysis against all known logic locking attacks,” in *Cryptographic Hardware and Embedded Systems—CHES*. Cham, Switzerland: Springer, 2017.
[25] D. Sirone and P. Subramanyan, “Functional analysis attacks on logic locking,” in *Proc. Design, Autom. Test Europe Conf. Exhibit. (DATE)*, 2019, pp. 1–4.
[26] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, “Cross-lock: Dense layout-level interconnect locking using cross-bar architectures,” in *Proc. Great Lakes Symp. VLSI (GLSVLSI)*. New York, NY, USA: ACM, 2018, pp. 147–152.
[27] 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. 56th Annu. Design Autom. Conf. (DAC)*. New York, NY, USA: ACM, 2019, pp. 89:1–89:6.
[28] G. Kolhe *et al.*, “Security and complexity analysis of LUT-based obfuscation: From blueprint to reality,” in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2019, pp. 1–8.
[29] G. Kolhe, S. M. Pd, S. Rafatirad, H. Mahmoodi, A. Sasan, and H. Homayoun, “On custom LUT-based obfuscation,” in *Proc. Great Lakes Symp. VLSI (GLSVLSI)*, 2019, pp. 477–482.
[30] G. S. Tseitin, *On the Complexity of Derivation in Propositional Calculus*. Berlin, Germany: Springer, 1983, pp. 466–483.
[31] Y. Xie and A. Srivastava, “Delay locking: Security enhancement of logic locking against IC counterfeiting and overproduction,” in *Proc. 54th Annu. Design Autom. Conf. (DAC)*. New York, NY, USA: ACM, 2017, pp. 9:1–9:6.
[32] A. Chakraborty, Y. Liu, and A. Srivastava, “TimingSAT: Timing profile embedded SAT attack,” in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*. New York, NY, USA: ACM, 2018, pp. 1–6.
[33] Y.-C. Chen, “Enhancements to SAT attack: Speedup and breaking cyclic logic encryption,” *ACM Trans. Design Autom. Electron. Syst.*, vol. 23, no. 4, pp. 1–25, May 2018.
[34] Y. Shen, Y. Li, A. Rezaei, S. Kong, D. Dlott, and H. Zhou, “BeSAT: Behavioral SAT-based attack on cyclic logic encryption,” in *Proc. 24th Asia South Pacific Design Autom. Conf. (ASPDAC)*. New York, NY, USA: ACM, 2019, pp. 657–662.
[35] 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. Design, Autom. Test Europe Conf. Exhibit. (DATE)*, Mar. 2018, pp. 85–90.
[36] A. Rezaei, Y. Li, Y. Shen, S. Kong, and H. Zhou, “CycSAT-unresolvable cyclic logic encryption using unreachable states,” in *Proc. Asia South Pacific Design Autom. Conf. (ASPDAC)*. New York, NY, USA: ACM, 2019, pp. 358–363.
[37] 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. IEEE Int. System-on-Chip Conf. (SOCC)*, Sep. 2015, pp. 257–262.
[38] V. Agarwal, N. Kankani, R. Rao, S. Bhardwaj, and J. Wang, “An efficient combinationality check technique for the synthesis of cyclic combinational circuits,” in *Proc. Conf. Asia South Pacific Design Autom. (ASP-DAC)*, 2005, pp. 212–215.
[39] M. D. Riedel and J. Bruck, “The synthesis of cyclic combinational circuits,” in *Proc. 40th Conf. Design Autom. (DAC)*, 2003, pp. 163–168.
[40] R. L. Rivest, “The necessity of feedback in minimal monotone combinational circuits,” *IEEE Trans. Comput.*, vol. C-26, no. 6, pp. 606–607, Jun. 1977.
[41] K. A. Hawick and H. A. James, “Enumerating circuits and loops in graphs with self-arcs and multiple-arcs,” in *Proc. FCS*, 2008, pp. 14–20.
[42] B. Dutertre, “Yices 2.2,” in *Computer Aided Verification*. Cham, Switzerland: Springer, 2014, pp. 737–744.
[43] A. Vakil, H. Homayoun, and A. Sasan, “IR-ATA: IR annotated timing analysis, a flow for closing the loop between PDN design, IR analysis & timing closure,” in *Proc. 24th Asia South Pacific Design Autom. Conf. (ASPDAC)*. New York, NY, USA: ACM, 2019, pp. 152–159.

Shervin Roshanisefat received the B.Sc. degree in computer engineering from Azad University, Qazvin, Iran, and the M.Sc. degree in computer engineering from the University of Tehran, Tehran, Iran, in 2015. He is currently working toward the Ph.D. degree at the Electrical and Computer Engineering Department, George Mason University, Fairfax, VA, USA.

He has worked earlier on telecommunication devices for SRD in Iran. His research interests are in the areas of approximate computing, low power design, hardware-level functional safety, and security.

Hadi Mardani Kamali received the B.Sc. degree in computer engineering from Khajeh Nasir University, Tehran, Iran, in 2011, and the M.Sc. degree in computer engineering from the Sharif University of Technology, Tehran, in 2013. He is currently working toward the Ph.D. degree at the Electrical and Computer Engineering Department, George Mason University, Fairfax, VA, USA.

His research focuses on hardware security, hardware/software acceleration applications, and power management in on-chip communication.

Houman Homayoun received the B.Sc. degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2003, the M.Sc. degree in computer engineering from the University of Victoria, Victoria, BC, Canada, in 2005, and the Ph.D. degree from the Department of Computer Science, University of California at Irvine, Irvine, CA, USA, in 2010.

He is currently an Associate Professor with the Electrical and Computer Engineering Department, University of California at Davis, Davis, CA, USA.

Dr. Homayoun has been serving as an Associate Editor for the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, since 2017. He was the Technical Program Co-Chair of GLSVLSI 2018 and the General Chair of the 2019 GLSVLSI Conference.

Avesta Sasan received the B.Sc. degree *(summa cum laude)* in computer engineering and the M.Sc. and Ph.D. degrees in electrical and computer engineering from the University of California at Irvine, Irvine, CA, USA, in 2005, 2006, and 2010, respectively.

He worked at the industry in Broadcom, Newport Beach, CA, USA, and Qualcomm Company, San Diego, CA, USA, until 2016. In 2016, he joined George Mason University, Fairfax, VA, USA, where he is currently an Associate Professor with the Department of Electrical and Computer Engineering.