SRCLock: SAT-Resistant Cyclic Logic Locking for Protecting the Hardware

Shervin Roshanisefat
George Mason University
sroshani@gmu.edu

Hadi Mardani Kamali
George Mason University
hmardani@gmu.edu

Avesta Sasan
George Mason University
asasan@gmu.edu

ABSTRACT

In this paper, we claim that cyclic obfuscation, when properly implemented, poses exponential complexity on SAT or CycSAT attack. The CycSAT, in order to generate the necessary cycle avoidance clauses, uses a pre-processing step. We show that this pre-processing step has to compose its cycle avoidance condition on all cycles in a netlist, otherwise, a missing cycle could trap the SAT solver in an infinite loop or force it to return an incorrect key. Then, we propose several techniques by which the number of cycles is exponentially increased with respect to the number of inserted feedbacks. We further illustrate that when the number of feedbacks is increased, the pre-processing step of CycSAT faces an exponential increase in complexity and runtime, preventing the correct composition of loop avoidance clauses in a reasonable time before invoking the SAT solver. On the other hand, if the pre-processing is not completed properly, the SAT solver will get stuck or return incorrect key. Hence, when the cyclic obfuscation in accordance to the conditions proposed in this paper is implemented, it would impose an exponential complexity with respect to the number of inserted feedback, even when the CycSAT solution is used.

ACM Reference Format:
Shervin Roshanisefat, Hadi Mardani Kamali, and Avesta Sasan. 2018. SRCLock: SAT-Resistant Cyclic Logic Locking for Protecting the Hardware. In *GLSVLSI '18: 2018 Great Lakes Symposium on VLSI, May 23–25, 2018, Chicago, IL, USA*. https://doi.org/10.1145/3194554.3194596

1 INTRODUCTION

The cost of building a new semiconductor fab was estimated to be $5.0 billion in 2015, with large recurring maintenance costs [5][7], and sharply increases as technology migrates to smaller nodes. Due to the high cost of building, operating, managing, and maintaining state-of-the-art silicon manufacturing facilities, many major U.S. high-tech companies have been always fabless or went fabless in recent years (e.g., AMD, Broadcom, Marvell, Nvidia, Qualcomm, and Xilinx, to name a few). Thus, to reduce the fabrication cost, and for economic feasibility, most of the manufacturing and fabrication is pushed offshore [5]. 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 prior to production, and an 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 [7].

To prevent the adversaries from such attacks, researchers have proposed various obfuscation methods for hiding and/or locking the functionality of an netlist. 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 the de-obfuscation attacks leveraging satisfiability (SAT) solvers [13][6] combined with Signal Probability Skew (SPS) attacks [16] 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 [18]. The CycSAT added a pre-analysis step to the original SAT attack for detection and avoidance of cycles in the netlist during SAT attack. In this paper, we further investigate the CycSAT attack and illustrate its pre-processing step has to compose the cycle avoidance condition by traversing all cycles in a netlist. We illustrate that by having a small number of methodically constructed feedbacks in a netlist, an exponentially large number of simple and nested cycles could be generated in a netlist, and we propose two different techniques for building such behavior. Since a successful SAT attack on a cyclic circuit requires the avoidance clauses, and time it takes to generate such avoidance clauses has an exponential relation with the number of inserted feedbacks, the CycSAT attack faces exponential runtime at its processing step. Hence, when deploying the CycSAT, the complexity of the problem is not in the SAT solver step of the problem, but in its pre-processing step.

The rest of this paper is organized as follows. In section 2 we cover the background on logic obfuscation. Then in section 3 we elaborate on the limitation of CycSAT and our approach for breaking the CycSAT. In section 4 we introduce our techniques for building an exponential relation between the number of feedbacks 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 CycSAT pre-processing step. Our experimental results are summarized in section 5. Section 6 concludes the paper.

2 BACKGROUND ON LOGIC OBFUSCATION AND SAT ATTACKS

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 and circuit locking are two of the widely explored obfuscation mechanisms [3][17][4] for this purpose. A camouflaged gate is a gate that after reverse engineering (by means of 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 a number of key inputs such that only when a

GLSVLSI '18, May 23–25, 2018, Chicago, IL, USA

ACM ISBN 978-1-4503-5724-1/18/05…$15.00
https://doi.org/10.1145/3194554.3194596

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 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 (for DFT), 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 satisfiability (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 [13][6]. 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 key is referred to as a *Discriminating Input* 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 Discriminating Input Validation Circuit (DIVC) as illustrated in algorithm 1. When the SAT solver can no longer find a X_{DI}, the DIVC circuit contains a complete set of discriminating 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 [13][6].

Algorithm 1 SAT Attack on Obfuscated Circuits

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

This reevaluation redirected the attention of the researchers to find harder obfuscation schemes that are more resilient to SAT attacks. SARLock and Anti-SAT [15][14] obfuscation methods were proposed for this purpose, however further research proved that these obfuscation techniques are prone to a simple removal attack after identification of these blocks using Signal Probability Skew (SPS) attack [16], or identification of most key values using approximate SAT attacks [11], leaving the problem of finding a SAT and SPS resilient obfuscation still unresolved.

A different direction for obfuscating a netlist was proposed in [12] where by introducing feedbacks 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 cycle breaks the original SAT attack in [13][6]. This cyclic obfuscation was later broken with the introduction of CycSAT attack in [18]. In CycSAT attack, before invoking the SAT solver, the netlist is checked for key conditions that may result in the creation of cycles. This conditions are translated to a set of cycle avoidance clauses and are added to the list of clauses that represent the circuit SAT

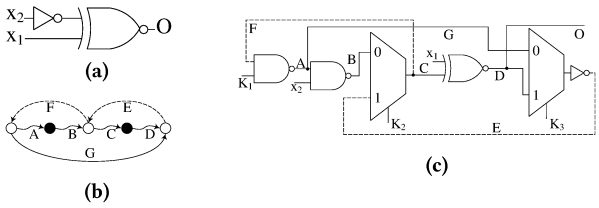

Figure 1: (a) Original circuit (b) Flow diagram of the netlist (c) Obfuscated circuit.

problem. The algorithm 2 illustrates the flow of utilizing the cycle avoidance-clauses in CycSAT attack.

Algorithm 2 CycSAT Attack on Cyclic Obfuscated Circuits

1: Find a set of feedback signals $(w_0, w_1, \ldots w_m)$;
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: $SAT_{circut} = C(X, K_1, Y_1) \wedge C(X, K_2, Y_2) \wedge (Y_1 \neq Y_2)$;
6: **while** $((X_{DI}, K_1, K_2) \leftarrow SAT_F(SAT_{circut}) = T)$ **do**
7: $\quad Y_f \leftarrow C_{BlackBox}(X_{DI})$;
8: $\quad DIVC = DIVC \wedge C(X_{DI}, K_1, Y_f) \wedge C(X_{DI}, K_2, Y_f)$;
9: $\quad SAT_{circut} = SAT_{circut} \wedge DIVC$;
10: $KeyGenCircuit = DIVC \wedge (K_1 = K_2)$
11: $Key \leftarrow SAT_F(KeyGenCircuit)$

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. The function $F(w_i, j)$ is a function that construct the condition for "*having no structural path*" between signal w_i to signal j. The $F(w_i, j)$ 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:

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

In this function, the $NK(j)$ are the non-key inputs of signal j, and $bk(l, j)$ is the condition on 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.

3 BREAKING CYCSAT

The computation for $F(w_i, j)$ could be done in two ways: (i) traversing through a cycle starting from w_i until w_i is visited again and ignoring the cycle break conditions imposed by fanins of other nested cycles; or (ii) traversing through one cycle and adding the cycle break conditions imposed by other nested cycle. We demonstrate that the first choice results in missing some NC conditions, leaving cycles in a design that could break the SAT solver, and by choosing the condition (2) we are forced to build the NC condition by visiting all cycles in the netlist.

Considering the obfuscated netlist in Fig. 1 and a topological sort from gate A, the edge E and F are identified as feedbacks. When following rule (i), and 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 $(k_1, k_2, k_3) = (0, 1, 0)$. In this case, the NC condition is satisfied, however, the larger nested cycle *EFGE* is not broken. Hence, the NC condition would not resolve the cycles if nested or multi-path scenarios exist. In this case, if the wrong key $(k_1, k_2, k_3) = (0, 1, 0)$ is chosen by SAT solver, it will enter

a loop. Depending on whether the cycle is stateful or oscillating, the SAT solver will either be trapped in an infinite loop or will exit with an incorrect key assignment. 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 [13][6].

To avoid the problem imposed by rule (i), we need to follow the rule (ii) where the key contribution of all fanins in all stages are considered. When using rule (ii) for building the NC condition for the same circuit we have:

1: $F(F, A) = F(F, F) \vee bk(k_1) = k_1'$
2: $F(F, F') = (F(F, A) \vee bk(k_2)) \wedge (F(F, E) \vee bk(k_2)) = (k_1' \vee k_2) \wedge (k_1' \vee k_3 \vee k_2')$
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 rule (ii), the previous assignment of keys $(k_1, k_2, k_3) = (0, 1, 0)$ will no longer be a valid assignment, preventing the SAT solver from being stuck or suggesting 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 [18], in order to derive the "no structural path" condition, some of the combinational cycles (such as *EFGE* in Fig. 1) have been visited more than once. Hence, the number of times the key conditions has to be generated is even larger than the number of cycles in a netlist.

The problem of visiting nested cycles more than once in a CycSAT attack could be resolved by a slight modification to the CycSAT pre-processing step. In the modified attack, instead of applying rule (ii) on one-cycle-per feedback, we could apply the rule (i) on all cycles. It is intuitive to see that both approaches produce the same NC clauses. For example, in Fig. 1 when following condition (i), and traversing cycle *EFGE*, 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 the rule (i), the NC condition of rule (ii) is generated. However, in this case, the combinational cycle *EFGE* 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 is used to break the CycSAT attack in this paper.

A different method of introducing complexity is by eliminating 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 pre-processing step has to build the NC condition by checking for "*no sensitizable path*" condition [18], instead of "*no structural path*" condition. The no sensitizable path condition from [18] is recited in equation 2:

$$F(w_i, j) = \bigwedge_{l \in fanin(j)} F(w_i, l) \vee ns(l, j) \tag{2}$$

The "no sensitizable path" condition generates a clause for each multi-input gate in a cycle. As the result, NC clauses are much longer and much weaker. Hence, adding even a small number of feedbacks to such circuits (that have valid Boolean cycles) for the purpose of obfuscation, will significantly increase the size of the circuitSAT problem. To illustrate the weaker and longer nature of the NC clauses, the no "sensitizable path condition" for the circuit in Fig. 1 is constructed below:

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: $NC = F(F, F') \wedge F(E, E') = (k_1' \vee x_2' \vee k_2) \wedge (k_1' \vee k_3 \vee k_2') \wedge (k_2' \vee k_3') \wedge (k_2' \vee k_1' \vee x_2' \vee k_3)$

The CycSAT pre-processing time, as illustrated in equation 3, is linearly related to the number of discovered cycles N and the time for composing the NC condition per cycle t_{NC}. Our approach for breaking the CycSAT is to exponentially increase the time needed for composing the NC condition in the pre-processing 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 feedbacks m, and increasing the time required for processing each cycle (t_{NC}) by forcing the pre-processing step to consider the "no sensitizable path" condition instead of " no structural path" condition.

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

In the next section we propose two methods for building an exponential relation between the number of cycles in a netlist with the number of inserted feedbacks and subsequently introduce three techniques for transforming a netlist to contain cyclic Boolean functions which forces an attacker to use the "no sensitizable path" condition in CycSAT attack.

4 CYCLIC OBFUSCATION

4.1 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 feedbacks, we introduce two approaches: (1) building Super Cycles (SC) and, (2) building Logarithmic Feedback Networks (LFN).

4.1.1 ***Building Super Cycles (SC).*** The process of building a SC is illustrated in Fig. 2. To define and build a SC, let us first define a Micro Cycle (MC). A MC is a cycle created by following the cycle creation conditions adopted from [12], which are recited below:

MC Condition 1: Any created cycle has to be non-reducible
MC Condition 2: At least $n \geq 2$ edges in each small cycle have to be removable

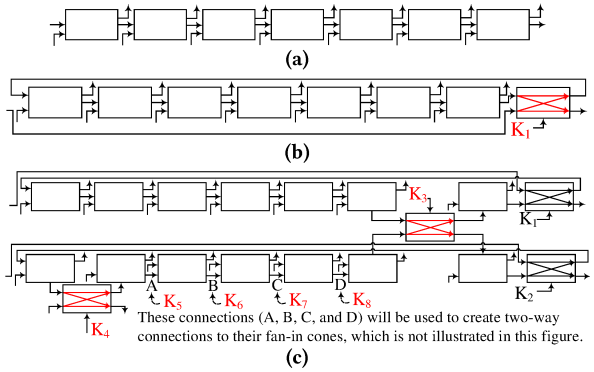

Figure 2: Building a Super Cycle from 7 gate MC. (a) A path segment containing 7 gates, (b) building a Micro Cycle, (c) building a SC by strongly connecting multiple MCs.

A reducible cycle has a single entry point. Hence, the depth-first-search (DFS) traversal of a netlist, that only contain reducible cycles is unique allowing reducible cycles to be opened by removing a unique set of edges which can be found efficiently [12]. By having multiple entries into each MC, the non-reducable condition is satisfied. Combined with having more than one removable edge, this forces an adversary to use the CycSAT pre-processing step to generate the necessary cycle avoidance clauses before invoking the SAT solver. At this point, when each MC is considered as a graph

vertice, a SC is the strongly connected graph of these vertices. In graph theory, a strongly connected graph is defined as a graph with at least one path between any two pairs of its vertices. Finally in the last step, the edge density of the SC is increased, creating additional paths between MCs. The process of building a SC is captured in algorithm 3.

Algorithm 3 Steps for building a Super Cycle

1: Construct MCs in the fanin of smallest possible number of primary outputs.
2: Strongly connect all MC cycles (this is illustrated in Fig. 2.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. 2.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.

By forcing the MC cycles to the fanin of the smallest number of primary outputs, we increase the possibility of shared edges or connecting edges between created MCs. By having all MC cycles 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 the result section, we illustrate that the number of created cycles, by following the steps in algorithm 3 has an exponential relation with the number of inserted feedbacks.

4.1.2 ***Building Logarithmic Feedback Networks (LFN).*** In this method, as illustrated in Fig. 3.a, several path segments in the fanin cone of the same primary output are selected. By breaking a signal in the midpoint of each path, two path segments are created. The signal entering and the signal exiting each half segment is marked as its start point (SP) and end point (EP) respectively. Then the SP and EP of multiple such path segments are used to build a logarithmic switching network (e.g., Omega, Butterfly, Benes, or Banyan network). When connecting M EPs to M SPs we need $M(1 + log_2(M))$ muxes for a non-blocking logarithmic network. In this case, when the correct key is applied, the switching network is configured correctly, otherwise, an 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 feedbacks and l is the log base two of the number of cycles of size l.*

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 a 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 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 $l = 8$, the cycles are paths like $\{SE_{ii}, ES_{ij}, SE_{jj}, ES_{jk}, SE_{kk}, ES_{ki}\}$. Since, we have two intermediate node, j and k, we should consider their permutation as a new cycle, i.e. $\{SE_{ii}, ES_{ik}, SE_{kk}, ES_{kj}, SE_{jj}, ES_{ji}\}$. So, for $l = 8$, we have $\binom{m}{3}2!$. 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 the $\sum_{l=1}^{m} \binom{m}{l}(l-1)!$ is the lower bound of the number of simple and nested cycles created by using the logarithmic network. As a matter of fact, 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,

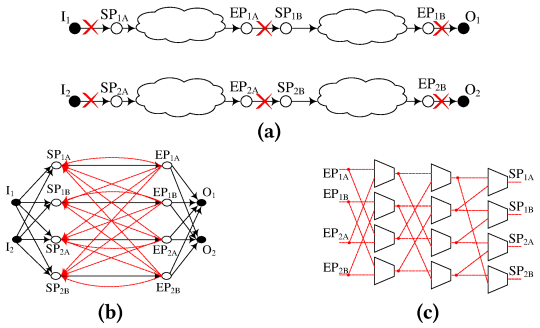

Figure 3: Building a logarithmic feedback network in which the number of cycles exponentially increase with the number of feedbacks.

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 feedbacks and the number of resulting cycles in the netlist.

4.2 Building Cyclic Boolean Functions

A Boolean function does not need to be acyclic; it is possible to reduce the number of gates in a circuit if a function could be implemented in its acyclic form [2][1][9][10]. For example, the work in [10] presents an n-input 2n-output positive unate Boolean function which can be realized with $2n$ two-input gates when feedback is used but requires $3n - 2$ gates if feedback is not used. Hence, cyclification of a circuit in addition to forcing the CycSAT pre-processing 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 the t_{NC} in equation 3, we suggest three approaches: (1) Template-based cyclic-function mapping, (2) Input-dependency based cycle generation and, (3) Node-merging based cycle generation.

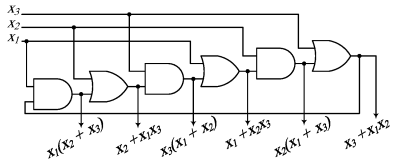

Figure 4: 3-input Rivest circuit implementing six functions.

4.2.1 ***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 [10] where a special case of it (for 3 inputs) is illustrated in Fig. 4. To introduce cycles, the circuit could be 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 (by means of gate and route obfuscation) the template will be hidden. Note that many such templates could be made [10][2][1][9], and by not knowing the template type and the camouflaged technique used to hide the connection, an attacker has no prior information to identify and remove these templates.

4.2.2 ***Input-dependency based cycle generation.*** This method explores the correlations between signals that share common primary inputs in their fanin cone. Considering N such signals in an arbitrary stage of a DAG, some of the 2^N input possibilities may

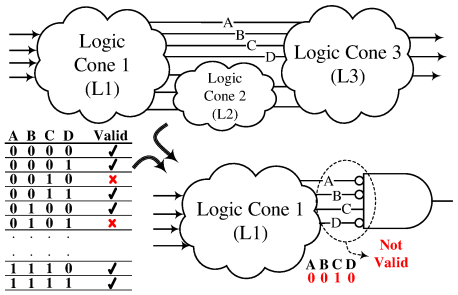

Figure 5: Due to correlation of intermediate signals, certain signal combinations may never occur.

never occur. For example, when tracking 4 signals A, B, C, and D in Fig. 5, we may find that $ABCD = \{0010\}$ could not occur. A SAT solver cold be used for finding the non-occurring input scenarios; This process is illustrated in Fig. 5, where the logic clusters L2 and L3 are removed, and the 4 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 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 non-occurring input scenario to the state preserving input of the sequential element. For example, by using SR latch in Fig. 6.a, If $SR = 11$ doesn't happen, the Q_{next} is the inverse of input S. Hence, we can build a circuit that ties the discovered non-occurring input scenario to the $SR = 11$. For example, let's assume wires A, B, C and D have a non-occurring combination $ABCD = 0010$ and these signals construct the signal $Y = A + B + CD$. Fig 6.c illustrates the signal Y reconstructed when the non-occurring combination of the inputs is tied to SR input of the latch. In the next step, to hide the correlation between input signals, they are further obfuscated. The SR latch feedback is also obfuscated using a set of muxes that create alternative paths for its feedback signals. This assures that the 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.

4.2.3 ***Node-merging based cycle generation.*** The third approach for cyclification of a netlist is based on the work in [2] 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 [2] 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.

5 RESULTS

In this section, we analyze the effectiveness of our proposed defense against SAT and CycSAT attacks. For finding the list of cycles in a netlist after cyclic obfuscation, the algorithm in [8] was implemented in C++. Our computational platform is a Dell PowerEdge R620 equipped with Intel Xeon E5-2670 2.50GHz and 64GB of RAM. We have used our proposed Super Cycle scheme for exponentially increasing the number of cycles on ISCAS-85 benchmarks, and the input-dependency-based netlist cyclification for forcing the CycSAT pre-processing to use the "no sensitizable path" condition instead of "no structural path" condition. However, similar results are observed when LFN is deployed, or other proposed techniques (template-based cyclification, and node-merging) are used for cyclification of a netlist.

Table 1: SAT-attack and CycSAT execution time after insertion of a SC (N=2) and insertion of a SC and 10 SR latches (N=2 + SR-L=10).

Circuit	N=2			N=2 + SR-L=10			
	SAT	#Cycles	CycSAT-I	SAT	#Cycles	CycSAT-I	CycSAT-II
c432	Inf Loop	23,879	2.561 s	Inf Loop	1.65×10^5	UNSAT	11.691 s
c499	0.56 s	236	0.104 s	Inf Loop	397	UNSAT	0.118 s
c880	Inf Loop	1,601	0.245 s	Inf Loop	7.87×10^6	UNSAT	793.125 s
c1355	Inf Loop	636	0.122 s	Inf Loop	5.00×10^5	UNSAT	53.215 s
c1908	0.28 s	294	0.101 s	Inf Loop	6,467	UNSAT	0.732 s
c2670	Inf Loop	1,570	0.234 s	Inf Loop	7,412	UNSAT	0.927 s
c3540	Inf Loop	5,991	0.756 s	Inf Loop	6,026	UNSAT	0.753 s
c5315	Inf Loop	4,869	0.613 s	Inf Loop	2.59×10^5	UNSAT	26.042 s
c7552	Inf Loop	124	0.189 s	Inf Loop	164	UNSAT	0.195 s

Table 1 captures the behavior of SAT and CycSAT when dealing with obfuscated cyclic and acyclic netlists. For generating the data in this table, we have created a cyclic version of each ISCAS-85 benchmark using input-dependency based obfuscation proposed in section 4.2.2, and then implemented a super cycle with two MCs in each netlist. The pure SAT attack is trapped in an infinite loop in both cases, with an exception of two benchmarks, that SAT solver luckily chooses a sequence of keys that avoid or exit the trap. The CycSAT when uses the "no structural path" condition (CycSAT-I) for the acyclic circuit, breaks the obfuscation easily, however, as illustrated in this table and predicted in equation 3, its runtime (which include the runtime for both pre-processing step and SAT solver's invocation) almost linearly varies with the number of cycles in each netlist. But when the netlist is acyclic, CycSAT-I returns UNSAT as it produces NC clause that breaks the DCG cycles incorrectly. On the other hand, the CycSAT when uses the "no sensitizable path" condition (CycSAT-II), breaks the obfuscation in all cases. However, since the number of created cycles are larger, and the time it takes to compose the NC condition for each cycle based on "no sensitizable path" condition is longer, the runtime of the SAT solver is considerably longer. In short, this table shows the capabilities of SAT, and CycSAT (I and II) for dealing with cyclic and acyclic original netlists, and also express the linear dependence between the CycSAT runtime and the number of created cycles. Hence, if the number of cycles exponentially increase, the runtime of CycSAT (pre-processing step) also exponentially increase.

The number of cycles created in ISCAS-85 benchmarks, when adding N=1, 2, 3, 5, 10, 15 and 20 MCs of size 7 (i.e., 7 gates in a cycle), while building a super cycle is reported in Table 2. As illustrated, the number of cycles created from implementing a super cycle is primarily a function of the number of inserted feedbacks and secondarily a function of the topology of starting netlist. As expected, the number of cycles is aggressively increased by the addition of each MC to the SC. Since the number of created cycles is also a function of topology, no single equation could predict the exact

Table 2: The number of cycles reported during CycSAT attack. The exponential fitting function is in form of $y = Ae^{Bx}$.

Circuit	N=1	N=2	N=3	N=5	N=10	N=15	N=20	Exponential Fit Function
c432	3,384	23,879	2.54×10^6	N/A	N/A	N/A	N/A	–
c499	10	236	397	55,585	N/A	N/A	N/A	–
c880	67	1,601	1,903	8.22×10^6	Timeout	Timeout	Timeout	A=881.953, B=1.75536
c1355	59	636	5.67×10^6	1.96×10^9	Timeout	Timeout	Timeout	A=21.7956, B=3.66316
c1908	13	294	12,594	1.33×10^7	Timeout	Timeout	Timeout	A=0.00174972, B=4.59814
c2670	273	1,570	8,912	2.90×10^5	Timeout	Timeout	Timeout	A=1036.51, B=1.82463
c3540	1,215	5,991	8.69×10^6	4.98×10^8	Timeout	Timeout	Timeout	A=4.0758, B=3.72457
c5315	162	4,869	6,650	1.22×10^9	Timeout	Timeout	Timeout	A=0.0217371, B=4.95008
c7552	11	124	1,558	2.57×10^5	1.15×10^9	Timeout	Timeout	A=9191.65, B=1.10724

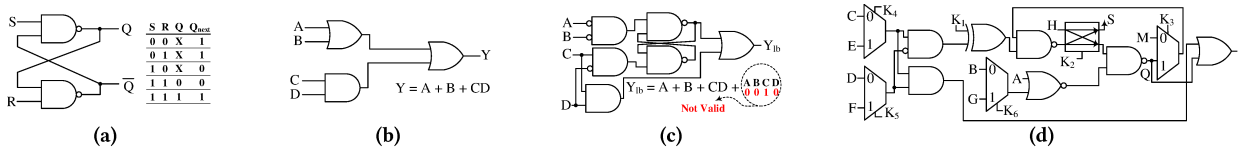

Figure 6: Input-dependency-based cyclification of a Boolean function. (a) SR latch (b) original circuit (c) cyclified circut when $ABCD = 0010$ is non-occuring. (d) obfuscated cyclified circuit using additonal random inputs E, F, G, H and M.

Table 3: Area overhead of SCs versus number of used MCs.

#MCs	N=1	N=2	N=3	N=5	N=10	N=15	N=20
#Gates	6	11	16	26	51	76	101

exponential growth for all benchmarks. Hence, using curve fitting techniques, the number of created cycles in each netlist as a function of the number of feedbacks is also reported in this table. As expected in all cases we see an exponential relation between the number of feedbacks and the number of created cycles. In smaller circuits like c432 and c499, there were not enough gates to create the required MCs for larger super cycles, hence, N/A is reported. For timeout entries, the number of cycles was not determined after 10 hours of execution on our server node. For executions resulted in timeout we also confirmed that initiating the CycSAT with incomplete NC clauses traps the SAT solver in an infinite loop. As illustrated in Table. 2, this is the case for most of the circuits with more than 10 MCs. The area overhead for building the SC in terms of the number of needed switches depends on the number of MCs and the number of gates in each MC. The area overhead for having various number of MCs of size 7 gates when building a SC is reported in Table 3. In short, as the number of inserted feedbacks increases, the pre-processing step of CycSAT faces an exponential increase in its runtime. Hence, by introducing a reasonable number of feedbacks using a methodology that exponentially increases the number of cycles (such as SC or LFN as proposed in this paper), the netlist could be protected against both SAT and CycSAT attacks by means of cyclic obfuscation.

Table 4: Number of cycles in c1908 circuit after inserting different number of cycles and SR latches.

#MCs \ #SR-L	0	1	2	5	10	20
0	0	5	30	245	7,675	7.35×10^6
1	13	189	9,875	4.37×10^6	1.78×10^7	Timeout
2	294	7,574	8.71×10^6	Timeout	Timeout	Timeout
3	12,594	6.82×10^5	1.42×10^7	Timeout	Timeout	Timeout
5	1.33×10^7	2.28×10^7	Timeout	Timeout	Timeout	Timeout
10	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout

Table 4 illustrates the strength of cyclic obfuscation when a netlist (c1908 in ISCAS-85) is first cyclified and then is subjected to cyclic obfuscation. For netlist cyclification, the input-dependency-based technique in section 4.2.2 is used, and for exponentially increasing the number of cycles with respect to number of feedbacks, the SC approach was used. After adding 5 SR-latches and only two MCs, list of cycles could not be generated under 10 hour time limit. This will again prevent the initiation of SAT-attack because cycle avoidance clauses could not be generated. In short, cyclification of a netlist before subjecting it to cyclic obfuscation could increase the runtime of CycSAT pre-processing step exponentially.

6 CONCLUSION

By introducing cycles in a netlist, the straightforward SAT-attack would be trapped in an infinite loop while CycSAT can solve such obfuscation problems. However, the problem with CycSAT is the runtime of its pre-processing step for generating the cycle avoidance clauses, which grow as a linear function of the number of cycles in the netlist. As a mean of defense, we introduced several techniques for exponentially increasing the number of cycles with respect to the number of inserted feedbacks. This, in turn, resulted in an exponential increase in runtime of CycSAT pre-processing step, disabling the SAT attack to be carried in a reasonable amount of time. Based on this study, the cyclic obfuscation when properly implemented poses an exponential runtime on CycSAT attack with respect to the number of inserted feedbacks. Hence, CycSAT or existing SAT-attacks are not an effective mean for breaking cyclic obfuscation.

REFERENCES

[1] V. Agarwal, N. Kankani, R. Rao, S. Bhardwaj, and J. Wang. 2005. An efficient combinationality check technique for the synthesis of cyclic combinational circuits. In *Proc. of the ASP-DAC*. 212–215. https://doi.org/10.1109/ASPDAC.2005.1466160

[2] J. H. Chen, Y. C. Chen, W. C. Weng, C. Y. Huang, and C. Y. Wang. 2015. Synthesis and verification of cyclic combinational circuits. In *IEEE Int'l System-on-Chip Conf. (SOCC)*. 257–262. https://doi.org/10.1109/SOCC.2015.7406959

[3] R. P. Cocchi, J. P. Baukus, L. W. Chow, and B. J. Wang. 2014. Circuit camouflage integration for hardware IP protection. In *2014 51st IEEE Design Automation Conf. (DAC)*. 1–5. https://doi.org/10.1145/2593069.2602554

[4] J. Davis, N. Kulkarni, J. Yang, A. Dengi, and S. Vrudhula. 2016. Digital IP protection using threshold voltage control. In *2016 17th Int. Symposium on Quality Electronic Design (ISQED)*. 344–349. https://doi.org/10.1109/ISQED.2016.7479225

[5] DIGITIMES. 2013. Trends in the global IC design service market. *online http://www.digitimes.com/news/a20120313RS400.html?chid=2* 2013 (2013).

[6] M. El Massad, S. Garg, and M. V. Tripunitara. 2015. Integrated Circuit (IC) Decamouflaging: Reverse Engineering Camouflaged ICs within Minutes. In *NDSS*.

[7] U. Guin, D. Forte, and M. Tehranipoor. 2013. Anti-counterfeit Techniques: From Design to Resign. In *2013 14th International Workshop on Microprocessor Test and Verification*. 89–94. https://doi.org/10.1109/MTV.2013.28

[8] K. A. Hawick and H. A. James. 2008. Enumerating Circuits and Loops in Graphs with Self-Arcs and Multiple-Arcs.. In *FCS*. 14–20.

[9] M. D. Riedel and J. Bruck. 2003. The synthesis of cyclic combinational circuits. In *Proc. 2003. Design Automation Conf. (IEEE Cat. No.03CH37451)*. 163–168. https://doi.org/10.1145/775832.775875

[10] R. L. Rivest. 1977. The Necessity of Feedback in Minimal Monotone Combinational Circuits. *IEEE TC* 26, 6 (1977), 606–607.

[11] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin. 2017. AppSAT: Approximately deobfuscating integrated circuits. In *IEEE Int'l Symp. on Hardware Oriented Security and Trust (HOST)*. 95–100.

[12] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin. 2017. Cyclic Obfuscation for Creating SAT-Unresolvable Circuits. In *Proc. of the on Great Lakes Symposium on VLSI 2017 (GLSVLSI '17)*. ACM, New York, NY, USA, 173–178.

[13] P. Subramanyan, S. Ray, and S. Malik. 2015. Evaluating the security of logic encryption algorithms. In *2015 IEEE Int. Symp. on Hardware Oriented Security and Trust (HOST)*. 137–143. https://doi.org/10.1109/HST.2015.7140252

[14] Y. Xie and A. Srivastava. 2016. Mitigating sat attack on logic locking. In *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 127–146.

[15] M. Yasin, B. Mazumdar, J. J. V. Rajendran, and O. Sinanoglu. 2016. SARLock: SAT attack resistant logic locking. In *2016 IEEE Int. Symp. on Hardware Oriented Security and Trust (HOST)*. 236–241. https://doi.org/10.1109/HST.2016.7495588

[16] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran. 2017. Security analysis of Anti-SAT. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. 342–347. https://doi.org/10.1109/ASPDAC.2017.7858346

[17] J. Zhang. 2016. A Practical Logic Obfuscation Technique for Hardware Security. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24, 3 (March 2016), 1193–1197. https://doi.org/10.1109/TVLSI.2015.2437996

[18] H. Zhou, R. Jiang, and S. Kong. 2017. CycSAT: SAT-based attack on cyclic logic encryptions. In *2017 IEEE/ACM Int'l Conf. on Computer-Aided Design (ICCAD)*. 49–56. https://doi.org/10.1109/ICCAD.2017.8203759