Cyclic Obfuscation for Creating SAT-Unresolvable Circuits

Kaveh Shamsi[1], Meng Li[2], Travis Meade[1], Zheng Zhao[2], David Z. Pan[2], and Yier Jin[1]
[1]ECE Department, University of Central Florida, Orlando, FL, USA
[2]ECE Department, University of Texas at Austin, Austin, TX, USA
[1]{kaveh, travm12}@knights.ucf.edu, [2]{meng_li,zzhao,dpan}@utexas.edu,
[1]yier.jin@eecs.ucf.edu

ABSTRACT

Logic locking and IC camouflaging are proactive circuit obfuscation methods that if proven secure can thwart hardware attacks such as reverse engineering and IP theft. However, the security of both these schemes is called into question by recent SAT based attacks. While a number of methods have been proposed in literature that exponentially increase the running time of such attacks, they are vulnerable to "find-and-remove" attacks, and only slightly hide the circuit functionality. In this paper, we present a novel approach towards creating SAT attack resiliency based on creating densely cyclic obfuscated circuit topologies by adding dummy paths to the circuit. Our methodology is applicable to both IC camouflaging and logic locking. We demonstrate that cyclic logic locking creates SAT resilient circuits with 40% less area and 20% less delay compared to an insecure XOR/XNOR-obfuscation with the same key length. Furthermore, we show that cyclic IC camouflaging can be implemented at the layout level with no substrate area overhead and little delay and power overhead with respect to the original circuit.

1. INTRODUCTION

With the globalization of the IC supply chain and the advent of fabless design houses, several security concerns have been raised [22]. These include reverse engineering by end-users, and malicious modification or overproduction by foundries. VLSI design for trust (DfTr) [12] refers to design-time techniques for thwarting these threats. Among them, IC camouflaging [4] and logic locking/-encryption [14] are two promising directions. With these approaches the designer can hide design information from a malicious foundry or end-user. We therefore categorize both schemes as circuit obfuscation, since they rely on the concept of "security through obscurity". IC camouflaging is based upon creating indistinguishable silicon layout structures that hamper reverse engineering of the IC by the end-user. In logic locking the circuit is augmented with additional key inputs such that it operates incorrectly without the correct key values applied. Therefore logic locking can prevent reverse engineering, as well as cloning or theft of intellectual property (IP) by the foundry or any other party that does not posses the correct key. Table 1 lists the protection provided by these circuit obfuscation schemes.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
GLSVLSI '17, May 10-12, 2017, Banff, AB, Canada
© 2017 ACM. ISBN 978-1-4503-4972-7/17/05…15.00
DOI: http://dx.doi.org/10.1145/3060403.3060458

Defense	Reverse engineering	Trojan Insertion	Overproduction
IC Camouflaging	by user only	no protection	no protection
Logic Locking	by user & foundry	by foundry	by foundry

Table 1: Protection provided by obfuscation schemes against a malicious end-user or foundry.

The security of these schemes has been the topic of research over the past several years through a competition of obfuscation defenses and deobfuscation attacks [11, 13, 5, 18]. Deobfuscation attacks are developed under various assumptions on the capabilities of the adversary. A prominent adversary-model in deobfuscation assumes the attacker has access to the inputs and outputs of a functional or unlocked chip. With such oracle-access to the chip the attacker can *query* the circuit for different inputs and use correct input-output pairs to find the correct key values or camouflaged functions. The most recent and strongest attack under this assumption is a disagreement based approach that queries the oracle circuit on input patterns that results in different outputs for different key values or camouflaged function hypotheses [5, 18]. These patterns are referred to as discriminating input patterns (DIP). Since the attack is formulated as a Boolean Satisfiability (SAT) problem and uses SAT-solvers, it is commonly referred to as the SAT attack. The SAT attack is successful in deobfuscating almost all known gate-level obfuscation solutions that have reasonable overhead [18].

To counter the SAT attack a number of defenses have been presented in literature [24, 26, 7, 23]. In these obfuscation schemes it is ensured that each DIP removes a limited number of key possibilities and therefore, the SAT attack will require an exponential number of queries to resolve the correct key. However, almost all these defenses rely on large low-activity blocks that can either be found and removed as was shown in [25], or are fundamentally capable of hiding only a few input patterns from the attacker due to their low corruptibility [17].

In this paper we present a novel SAT-resilient obfuscation scheme. This methodology is inspired by how configurable and cyclic interconnection networks allow modern programmable logic to implement a large set of Boolean functions with small logic components. The key idea is that if a logical loop is created in the circuit (which is typically avoided in prior work [16]), through adding dummy wires and gates, the adversary cannot launch the existing SAT attacks, since the circuit can no longer be represented as a directed acyclic graph (DAG). We show that from a graph theoretic perspective the number of possible ways for opening these loops can be made excessively large. In summary we make the following contributions:

- We present cyclic obfuscation as the first topological, wire-based approach to creating SAT attack resiliency;

- We discuss implementation details for ensuring the hardness of the attack, as well as silicon level techniques to implement the idea in both of IC camouflaging and logic locking flavors;

- We implement prototype cyclic logic locking and cyclic IC camouflaging schemes using the Nangate OpenCell 15nm technology library [1]. The results show both area and delay improvements for for cyclic logic locking compared to traditional schemes, and much smaller overheads for cyclic IC camouflaging.

The rest of the paper is organized as follows: Section 2 presents background information and prior work. Section 3 discusses SAT attack resiliency through topology obfuscation. Section 4 presents our cyclic obfuscation scheme and security and implementation aspects. Section 5 presents experimental results and section 6 concludes the paper.

2. BACKGROUND

2.1 Circuit Obfuscation

IC camouflaging. the goal in IC camouflaging is to prevent recovery of the netlist from layer-by-layer images of the IC. There exists an array of industrial fabrication technologies for creating indistinguishable layout structures [4, 20]. Some notable techniques include, using dummy vias, altering doping patterns, or using filler cells. Dummy vias are metal-to-metal or metal-to-diffusion contacts that appear to be connected from the top view while they are not in fact conducting. Modifying doping patterns allows altering the type of transistors or junctions while conventional IC imaging techniques cannot detect the difference [8]. Dummy cells can also be inserted in the empty areas of the chip and connected to active logic. With these techniques, camouflaging logic units can be built such that their overall Boolean function cannot be resolved to a specific function. Inserting these camouflaged units with different strategies creates obscurity during the reverse engineering process. This can be seen in Fig. 1a.

Logic Locking. Figure 1b shows a circuit obfuscated with logic locking. Logic locking was originally presented for hiding design details from the foundry and allowing for authenticated usage of the IC [14]. The first logic locking scheme was based on XOR/XNORing randomly selected wires in the circuit with key inputs [14]. Later proposals included more complex strategies for improving security. These techniques include inserting XOR/XNOR or MUX gates at locations that maximize properties such as output hamming-distance [9], or the size of cliques in the interference graph of key inputs [10]. Interconnect obfuscation schemes were also proposed such as shuffling wires in bus architectures [15], combinational logic [21, 3], or most recently, inserting dummy wires into the circuit through chip-level modification [16]. All of these schemes avoid creating logical cycles in the circuit.

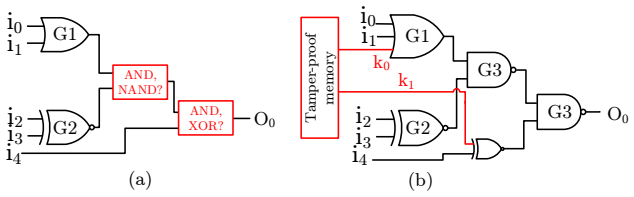

Figure 1: (a) IC camouflaging by replacing gates with camouflaged gates. (b) Logic locking with tamper-proof key inputs.

2.2 SAT Attacks

The SAT attack assumes the oracle-guided attack threat-model where the attacker has input-output access to an operational chip (as black-box) and a netlist of the obfuscated chip layout. The first step in the SAT attack is to model the obfuscated circuit with a Boolean function, C_{enc}, from input space I, and an obfuscation-secret (key)[1] space, K, to outputs ($C_{enc} : K \times I \rightarrow O$). For logic locking this is a direct transformation. Most camouflaging scheme can also be represented in such a form with polynomial overhead. For instance, a camouflaged gate can be modeled as a key-controlled MUX selecting among a set of possible functionalities of the camouflaged gate.

The attack begins by satisfying a mitter SAT problem, $C_{enc}(i, k_1) \neq C_{enc}(i, k_2)$ with some i, k_1, k_2. The solution input, i, is called a discriminating input pattern (DIP) since it differentiates k_1 and k_2. i is queried on the black-box C_o and the output y_i is obtained. The input-output relation, $(C_{enc}(i, k_1) = y_i) \wedge (C_{enc}(i, k_2) = y_i)$, is appended to the mitter SAT problem as a new constraint. The process continues until no more DIPs can be found. At this point satisfying the constraints will return a key, $k_* \in K_*$, which agrees with the black-box on all the input-output observations and cannot be differentiated by a query. El massed [5] showed that any key from K_* is necessarily a correct key ($\forall i \in I, C_{enc}(i, k_*) = C_o(i)$). The SAT attack is successful in deobfuscating almost all traditional logic locking schemes [18], and gate-level IC camouflaging schemes [5]. Furthermore, Shamsi et al. [17] extended the SAT attack to include approximation which allows the SAT attack to approximate the original circuit defeating the recently proposed SAT-resilient obfuscation schemes [24, 26, 7, 23].

3. SAT-RESILIENT OBFUSCATION

3.1 Security Criteria

Defining general and meaningful security criteria for circuit obfuscation is an elusive task. However, in the context of SAT attacks on combinational logic locking and camouflaging schemes the following criteria can be defined: 1) High query complexity: The query complexity measure, QC, of an obfuscated circuit, C_{enc}, with black-box access to the original circuit, C_o, is the minimum number of queries required to resolve the key. 2) High corruptibility: the corruptibility measure of the obfuscation, Cr, captures the effect of the key on the output. This can be defined as the disagreement probability: $\mathsf{Cr} = \mathsf{Pr}_{i \in I,\, k \in K}[C_{enc}(i, k) \neq C_o(i)]$. 3) As for approximation resiliency which is a stricter criteria there should not exist an algorithm that learns an ϵ-approximation of the original circuit with a success rate of $1 - \delta$ where ϵ and δ are small factors [17].

3.2 Existing Defenses Against SAT Attacks

A number of defenses against the SAT attack have been proposed in literature [24, 26, 7, 23]. These methods all rely on limiting the number of incorrect keys that each DIP can exclude. All these methods require tree structures that will output 1 for only a single input pattern and output 0 for all others. Therefore these schemes have a very low output corruptibility [23] and are hence all combined with high-corruptibility obfuscation schemes such as XOR/XNOR locking etc. The low-corruptibility schemes are generally vulnerable to "find-and-remove" style attacks that search the circuit structure or its function for the tree-like blocks and remove

[1]For most logic locking schemes the obfuscation secret or key is the physical key. For IC camouflaging the key is an abstract variable that decides the function of camouflaged logic. We will use key to refer to both.

them as was shown in [25]. Furthermore, with the approximate SAT attack the high-corruptibility schemes can be attacked independent of the low-corruptibility obfuscation [17].

3.3 Obfuscating Topology

It is easy to see that a fully programmable function is difficult to deobfuscate using the SAT attack. A function $\mathcal{X}_n : \{0,1\}^n \times K \rightarrow \{0,1\}$, which we define to be a function that implements all 2^{2^n} different Boolean functions from $\{0,1\}^n$ to $\{0,1\}$ by configuring $k \in K$, is secure against SAT attacks. Each query will reveal a single entry in the truth-table and hence $\mathsf{QC}(\mathcal{X}_n) = |I| = 2^n$. Furthermore, corruptibility is high since the probability of predicting the correct output given an incorrect key is no better than random guessing. Approximation resiliency is also high since the rate of discovering the truth-table is no faster than linear with respect to the queries.

It is well known that modern programmable logic such as FPGAs are practically fully programmable functions. An FPGA achieves this large expressiveness through combining small LUTs with 6 or less inputs and *programmable interconnects* as seen in Fig. 2a . With programmable interconnections, the small units can be arranged in different topologies to implement a large number of possible functionalities. A sea of AND/OR/NOT gates, plus a sea of configurable wires is capable of implementing a prohibitively large number of functionalities[2].

With respect to SAT attacks, one difficulty in deobfuscating programmable interconnects is to model the fabric with an acyclic Boolean circuit of keys and inputs. When reverse engineering a sea of programmable wires, any terminal on a logic unit can potentially be connected to any other logic unit. Therefore, in the graph model of such a circuit there are many edges and potentially many loops. However, existing SAT attacks require an directed acyclic graph (DAG) to begin the attack. It seems that at least additional abstract key-bits are required to represent the cyclic topology as a DAG. Cyclic obfuscation exploits this idea to create a highly complex circuit structure in ASICs without incurring the high overheads of fully programmable logic (30X area, 3X delay, and 10X power). Note that if such overhead is acceptable simply using programmable devices and using the configuration bitstream as the key is a secure defense against SAT based deobfuscation.

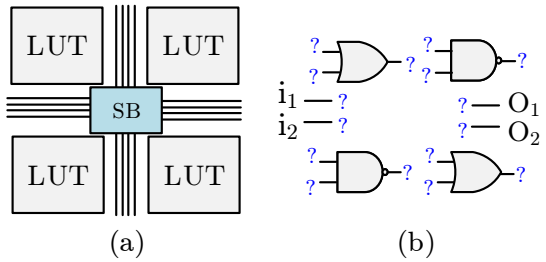

Figure 2: (a) FPGA programmable logic with configurable interconnects. (b) A circuit with known components but unknown interconnects is greatly unintelligible.

4. SECURE AND LIGHT-WEIGHT CYCLIC-OBFUSCATION

While it is possible to flood the circuit with dummy interconnections to achieve a highly complex and expressive cyclic structure, our light-weight cyclic obfuscation scheme relies on using the minimum number of dummy edges to create loops that are difficult to remove.

We will model circuits with a directed graph. Consider the original Boolean circuit C_o as seen in Fig. 3. In the graph representation of C_o, input wires are represented by nodes with no incoming edges. Similarly, output wires are represented by nodes with no outgoing edges. Each gate is represented by a node where the gate inputs are incoming edges and the gate output is represented by outward edges. The fanout cone of a node u includes every node v reachable from u, and the fanin cone of u includes every node v that can reach u.

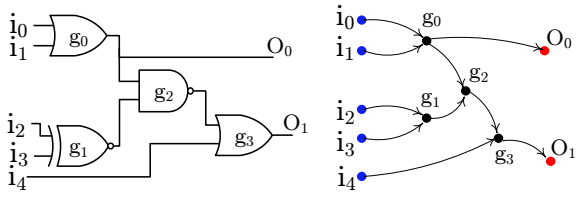

Figure 3: Directed graph model of Boolean circuit.

4.1 Creating Hard Loops

We can begin adding edges to the circuit DAG to create logical loops. In order for an edge to create a logical loop it has to connect a node in the fanout cone of u to a node in the fanin cone of u which will create a loop with u being on the loop. The goal of the defender is to first ensure that there are more than one ways to open the loop, all of which are feasible from the attacker perspective. Furthermore, the defender wants to maximize the number of possible ways to open this loop. These require two conditions that are discussed herein.

CONDITION 1. *Any created loop has to be non-reducible.*

Given a general cyclic graph the problem of finding edges in the graph to remove to obtain an acyclic graph is known as the *feedback arc set* problem [2] which is NP-complete. However, the NP-complete hardness of this problem may not necessarily be used to ensure the difficulty of recovering the original acyclic circuit in our case. The inputs and outputs of a circuit graph are known as sources and sinks to the attacker which help infer a general direction in the graph and potentially remove edges that oppose this direction. More precisely, a cyclic circuit graph is a *flow-graph* [6].

Flow-graphs and loops are studied in depth in the context of programs [6]. Loops in flow-graphs can be *reducible* or non-reducible. If a flow-graph has only reducible loops, the depth-first-search (DFS) traversal of this graph is unique. This unique DFS tree will allow all reducible loops to be opened by removing a unique set of edges which can be found efficiently. Since this would be greatly detrimental to the security of cyclic obfuscation any loop created during the obfuscation has to be non-reducible. A sufficient condition for a loop to be non-reducible is for it to have multiple entry points. An entry point in a loop is an edge arriving on one of the vertices in the loop from a vertex outside the loop.

Example: Consider graph in Fig. 4a. In this graph if we add edge e_3, the loop $\{e_1, e_2, e_3\}$ is reducible since it has a single entry through g_1 and in the DFS traversal of this graph e_3 will always be a backward edge. On the other hand in graph Fig. 4b, the loop $\{e_1, e_2, e_3\}$ is non-reducible since if we enter the loop from g_1, e_2 will be a backward edge, but if we enter the loop from g_2, a different edge, e_1, will be opposing the flow.

[2]Note that having g gates and infinite interconnect resources may not implement all 2^{2^n} Boolean functions but for sufficiently large values of g implements a prohibitively large number of functions for an attacker to prune.

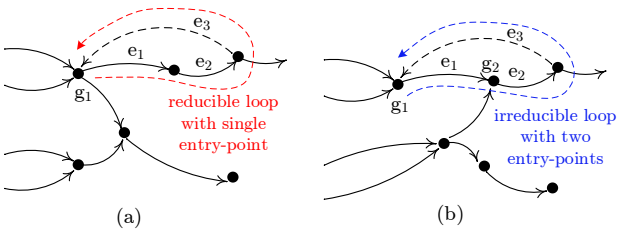

Figure 4: Reducible and non-reducible loops.

CONDITION 2. *At least $n \geq 2$ edges in a loop have to be "removable". This will result in a loop complexity of 2^n.*

An edge is removable if 1) the attacker is able to modify the key to remove that edge from the graph; 2) removing the edge through a key value, should not create gates with no inputs or no outputs. Otherwise, even though the edge removal is controlled by the key, its removal is not possible without introducing errors into the circuit.

Example: In the graph in Fig. 4b, if the removal of edge e_1 is controlled by a key-bit it would still not be a removable edge, since removing e_1 will leave the gate, g_1, with no outputs.

If an edge is removable, then it creates a dilemma for the attacker during reverse engineering. If a loop has n removable edges, there are $\binom{n}{m}$ ways to open it by removing m edges. Thus in total there are $\sum_{m=1}^{n}\binom{n}{m} = 2^n - 1$ ways to open a loop with n removable edges.

4.2 Dummy Logic as Extra Nodes

Thus far we only discussed adding edges (wires) to the design to create obscurity. However, dummy gates can also be added to the circuit as extra nodes and included in loops. This will have the following benefits: 1) inserting dummy gates in realistic designs can have a virtually zero area overhead, since they can be inserted in empty areas that would be otherwise filled with filler cells. Therefore, dummy gates can be easily used to increase the length of a given loop at no cost. 2) As was discussed, for an edge to be removable, its removal should not result in a gate with empty fanout or fanin. However, if the attacker is faced with the possibility of dummy cells, this condition can be relaxed, since if the gate is dummy it can have an empty fanout or fanin.

4.3 Light-Weight Implementation Algorithm

We will first discuss a logic locking implementation of the obfuscation scheme. For logic locking, from a graph level, we first need to identify a path (v_1, v_2) and make sure that at least one node between v_1 and v_2 has more than one incoming edges to ensure the non-reducibility of the loop. If this is the case then we can feed v_2 back into v_1. We then have to ensure that edges on this loop are removable.

To create removable feedback wires we can either use a gate to nullify the effect of a a wire on a gate as seen in Fig. 5a, or use a MUX gate to select between two wires as seen in Fig. 5b. To make edges along the path (v_1,v_2) removable, depending on the fanout count of the nodes on the path we can use one or two MUX gates to implement this as is discussed in the bellow example. a general algorithm for performing the above cyclic logic locking is presented in Algorithm 1.

Example: Per Fig. 6, we first feedback the wire w_3 into w_0 and then we need to make the edges on the path (w_1, w_3) removable. For wire w_1, the wire already connects to more than one location. Therefore, simply including the MUX gate, M_1, in this path will allow the key-bit k_1 to open this edge. When k_1 opens the loop, the input of gate g_2 is fed from a randomly selected wire in the circuit r_0. For the wire w_2 however, since it is driving the gate g_2 only, two MUXs are used to make this edge removable. First, M_2 selects between w_2 and the random wire r_1. Second, M_3 is used to open a path from w_2 to a randomly selected location r_2. MUXs M_2 and M_3 are controlled by the same key-bit k_2 which will remove the edges from the loop and redirect them to other locations.

Algorithm 1 Obfuscate circuit C_o with N loops of length M (attack complexity is $(2^M)^N$).

```
1: function CyclicObfuscate(C_o, N, M)
2:     while loops found < N do
3:         repeat
4:             u ← random gate from C_o
5:             start DFS at u to find path of length M
6:         until path of length M found
7:         feed v back to u
8:         for each g on path (u, v) do
9:             if fanoutSize(g) = 1 then
10:                make g removable with two MUXs
11:            else
12:                make g removable with single MUX
13:            end if
14:        end for
15:    end while
16: end function
```

For IC camouflaging a similar procedure can be used, except that MUXs in IC camouflaging can be implemented with dummy vias with virtually zero gate area overhead as seen in Fig. 7. Even for the case of logic locking one-time-programmable contacts, such as anti-fuse vias, high R_{on}/R_{off} non-volatile memory devices, or other metal-to-metal programmable switches can be used to implement a light-weight and non-volatile MUX element for cyclic logic locking. We note that cyclic obfuscation heavily relied on the security of such silicon level techniques. We allow the attacker to tell the difference between true vias and potentially-true vias, however, the attacker should not be able to tell apart true and fake vias. For instance, doping based programmable/camouflaged connections can be revealed with selective etching, or capacitive imaging [19] and should not be used, whereas Mg/MgO connections [3] are more difficult to differentiate at least with current scanning electron microscopy techniques.

5. EXPERIMENTATION RESULTS

We implemented cyclic obfuscation in both IC camouflaging and logic locking. We begin with cyclic logic locking. The benchmark netlists were analyzed and modified using a C++ framework. Synthesis and delay and area characterization were performed using Design Compiler (DC) with the Nangate OpenCell 15nm library [1]. It is important to inform the synthesis tool that the loops created in the circuit are not part of the critical path. Otherwise the synthesis tool will try to fix design-rule violations for such paths which would create large unnecessary overheads. The results for cyclic logic locking on the c432 ISCAS benchmark circuit are shown in Table 3. Since c432 is the smallest benchmark, overhead trade-offs are more significant. Larger circuits will have negligible overheads.

We then applied the cyclic logic locking to a larger set of ISCAS and MCNC benchmarks as shown in Table 2, for $N = 6$ (loop count) and $M = 6$ (loop length). The same benchmark circuits were obfuscated with a traditional random XOR/XNOR obfuscation with the same number of key-bits (72). The area and delay results from synthesis in the 15nm OpenCell library are shown in Figs. 8 and 9 which shows the feasibility of a low overhead implementation of cyclic logic locking in comparison with traditional methods. Note that SAT resilient proposals [24, 26, 7, 23] are not

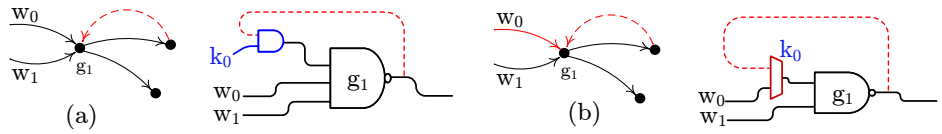

Figure 5: Add-edge operations: (a) Nullify with key-gate. (b) Choose with key-controlled MUX.

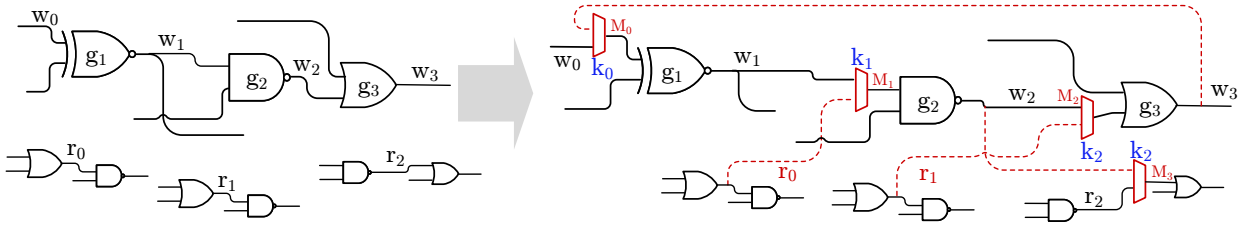

Figure 6: Cyclic logic locking example ensuring the removabillity of edges on path.

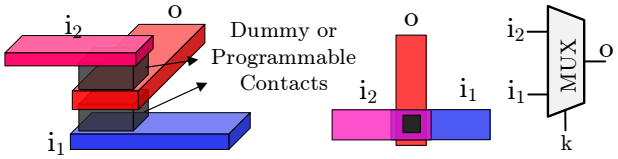

Figure 7: 2-to-1 MUX with dummy contacts or one-time-programmable vias.

ISCAS				MCNC			
circuit	#inputs	#outputs	#gates	circuit	#inputs	#outputs	#gates
c432	36	7	160	apex2	39	3	610
c499	41	32	202	ex5	8	63	1055
c880	60	26	383	i9	88	63	1315
c1355	41	32	546	i7	199	67	1581
c1908	33	25	880	k2	46	45	1815
c2670	157	64	1193	ex1010	10	10	5066
c3540	50	22	1669	des	256	245	6437
c5315	178	123	2307				
c7552	207	108	3512				

Table 2: Benchmark circuits adopted from [18]. Gate count is in terms of number primitive gates.

included in the comparison since they all utilize XOR/XNOR obfuscation internally to increase corruptibility.

As for cyclic IC camouflaging, we created a set of dummy vias and MUXs using Cadence Virtuoso and utilized them in Encounter as part of a physical-only library. As seen in Figure 10c the dummy vias are automatically placed and routed using Encounter. Our layout experiments with three benchmark circuits, c432, c1908 and c3540 showed that with a core utilization of 70%, the cyclic camouflaged circuit does not increase the core area and only contributes to additional congestion and wire-length. The maximum wire-length increase was for 8 cycles of 6 edges long inserted in the c1908 benchmark circuit with 32% wire-length overhead and all designs converged without design-rule violations. Accurate delay and power overhead analysis require capacitance details from the camouflaged vias, however, by assuming capacitance values of two crossing wires for the fake vias, delay overhead was a maximum of 6.2% and power overhead was a maximum of 5% for 8 cycles of length 8 across the benchmarks. Extensive layout implementation and characterization is a topic of our future work.

6. CONCLUSION

In this paper a novel approach towards thwarting SAT attacks was proposed based on creating interconnection cyclic obscurity. The method is applicable to both logic locking and IC camouflaging and as was demonstrated in this paper can be performed with very low performance and area overhead. While the current SAT attacks are incapable of reverse engineering cyclic circuits, investigating novel oracle-guided attacks that are able to model such circuits is essential to further evaluating the security of cyclic obfuscation. Extensive layout level characterization and implementation utilizing dummy gates is also a significant future direction.

7. REFERENCES

[1] Nangate freepdk15 open cell library. http://www.nangate.com/?page_id=2328.

[2] P. Charbit, S. Thomassé, and A. Yeo. The minimum feedback arc set problem is np-hard for tournaments. *Combinatorics, Probability and Computing*, 16(01):1–4, 2007.

[3] S. Chen, J. Chen, D. Forte, J. Di, M. Tehranipoor, and L. Wang. Chip-level anti-reverse engineering using transformable interconnects. In *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pages 109–114. IEEE, 2015.

[4] R. P. Cocchi, J. P. Baukus, L. W. Chow, and B. J. Wang. Circuit camouflage integration for hardware ip protection. In *Proc. Design Automation Conf.*, pages 1–5. IEEE, 2014.

[5] M. El Massad, S. Garg, and M. V. Tripunitara. Integrated circuit (ic) decamouflaging: Reverse engineering camouflaged ics within minutes. In *NDSS*, 2015.

[6] M. S. Hecht and J. D. Ullman. Characterizations of reducible flow graphs. *Journal of the ACM (JACM)*, 21(3):367–375, 1974.

[7] M. Li, K. Shamsi, T. Meade, Z. Zhao, B. Yu, Y. Jin, and D. Z. Pan. Provably secure camouflaging strategy for ic protection. In *Proc. Int. Conf. on Computer Aided Design*, page 28. ACM, 2016.

[8] S. E. Quadir, J. Chen, D. Forte, N. Asadizanjani, S. Shahbazmohamadi, L. Wang, J. Chandy, and M. Tehranipoor. A survey on chip to system reverse engineering. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(1):6, 2016.

[9] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri. Logic encryption: A fault analysis perspective. In *Proc. Design, Automation and Test in Eurpoe*, pages 953–958, 2012.

[10] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri. Security analysis of logic obfuscation. In *DAC*, pages 83–89, 2012.

[11] J. Rajendran, O. Sinanoglu, and R. Karri. Vlsi testing based security metric for ic camouflaging. In *Proc. IEEE Int. Test Conf.*, pages 1–4. IEEE, 2013.

[12] J. Rajendran, O. Sinanoglu, and R. Karri. Regaining trust in vlsi design: Design-for-trust techniques. *Proceedings of the IEEE*, 102(8):1266–1282, 2014.

[13] J. Rajendran, H. Zhang, C. Zhang, G. S. Rose, Y. Pino, O. Sinanoglu, and R. Karri. Fault analysis-based logic encryption. *IEEE Trans. on Computers*, 64(2):410–424, 2015.

[14] J. A. Roy, F. Koushanfar, and I. L. Markov. Epic: Ending piracy of integrated circuits. In *Proc. Design, Automation and Test in Eurpoe*, DATE '08, pages 1069–1074, 2008.

Table 3: Loop count and length versus loop complexity (number of possible ways to open loops), and delay/area overhead% for the c432 circuit. loop complexity can be a measure of security.

cycle length	3			4			6			8			12		
cycle count	area	delay	comp	area	delay	comp	area	delay	comp	area	delay	comp	area	delay	comp
2	12.21	2.88	64	24.59	5.08	256	19.07	11.47	4096	18.12	11.95	10^4	29.74	13.19	10^7
3	17.93	6.07	512	17.74	7.64	4096	24.02	9.31	10^5	46.88	43.79	10^7	50.89	16.04	10^{10}
4	20.21	4.42	4096	23.64	7.41	10^4	28.02	18.34	10^7	46.50	16.07	10^9	52.41	14.22	10^{14}
6	23.45	8.02	10^5	34.31	-3.40	10^7	39.45	22.58	10^{10}	76.60	46.30	10^{14}	102.51	44.64	10^{21}
8	48.22	36.40	10^7	50.50	38.27	10^9	54.50	14.47	10^{14}	91.85	30.31	10^{19}	111.85	36.81	10^{28}

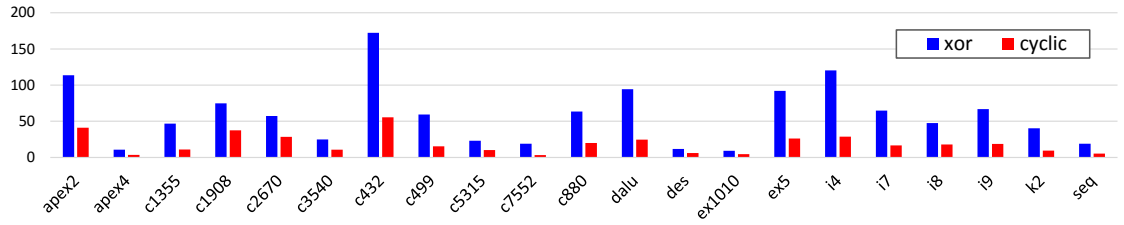

Figure 8: Area overhead of cyclic logic locking with 6 loops of length 6 versus XOR/XNOR locking with the same number of key-bits(72).

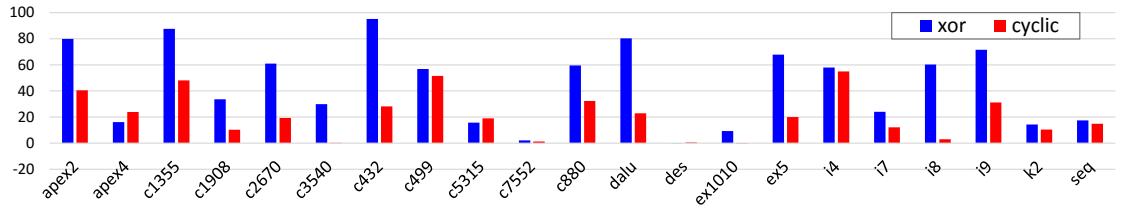

Figure 9: Delay overhead ($M = N = 6$).

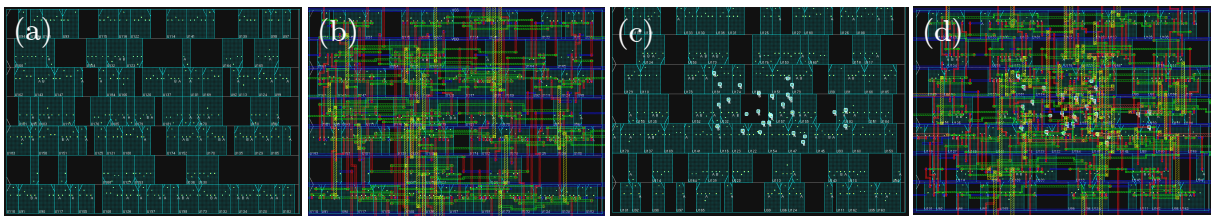

Figure 10: Layout-level cyclic camouflaging with 4 loops of length 4 on the small c432 circuit. Figure (a) shows the original circuit and Figure (b) shows the original routing. In Figure (c) the placement of the dummy vias are shown. As seen from Figure (d) the routed camouflaged circuit only contributes to wire overhead.

[15] J. A. Roy, F. Koushanfar, and I. L. Markov. Protecting bus-based hardware ip by secret sharing. In *Proceedings of the 45th annual Design Automation Conference*, pages 846–851. ACM, 2008.

[16] B. Shakya, N. Asadizanjani, D. Forte, and M. Tehranipoor. Chip editor: leveraging circuit edit for logic obfuscation and trusted fabrication. In *Proceedings of the 35th International Conference on Computer-Aided Design*, page 30. ACM, 2016.

[17] K. Shamsi, M. Li, T. Meade, Z. Zhao, Y. Jin, and D. Z. Pan. Appsat: Approximately deobfuscating integrated circuits. In *Proc. IEEE Symp. Hardware-Oriented Security and Trust*. IEEE, 2017.

[18] P. Subramanyan, S. Ray, and S. Malik. Evaluating the security of logic encryption algorithms. In *Proc. IEEE Symp. Hardware-Oriented Security and Trust*, pages 137–143. IEEE, 2015.

[19] T. Sugawara, D. Suzuki, R. Fujii, S. Tawa, R. Hori, M. Shiozaki, and T. Fujino. Reversing stealthy dopant-level circuits. In *Cryptographic Hardware and Embedded Systems*, pages 112–126. Springer, 2014.

[20] A. Vijayakumar, V. C. Patil, D. E. Holcomb, C. Paar, and S. Kundu. Physical design obfuscation of hardware: A comprehensive investigation of device and logic-level techniques. *IEEE Transactions on Information Forensics and Security*, 12(1):64–77, 2017.

[21] T. F. Wu, K. Ganesan, A. Hu, H.-S. P. Wong, S. Wong, and S. Mitra. Tpad: Hardware trojan prevention and detection for trusted integrated circuits, 2015.

[22] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor. Hardware trojans: Lessons learned after one decade of research. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 22(1):6, 2016.

[23] Y. Xie and A. Srivastava. Mitigating sat attack on logic locking. http://eprint.iacr.org/2016/590.pdf.

[24] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu. Sarlock: Sat attack resistant logic locking. In *Proc. IEEE Symp. Hardware-Oriented Security and Trust*, pages 236–241, 2016.

[25] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran. Security analysis of anti-sat. https://eprint.iacr.org/2016/896.pdf.

[26] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran. Camoperturb: secure ic camouflaging for minterm protection. In *Proc. Int. Conf. on Computer Aided Design*, page 29. ACM, 2016.