IcySAT: Improved SAT-based Attacks on Cyclic Locked Circuits

Kaveh Shamsi*, David Z. Pan†, and Yier Jin*
*Department of Electrical and Computer Engineering, University of Florida
†Department of Electrical and Computer Engineering, University of Texas at Austin
kshamsi@ufl.edu, dpan@ece.utexas.edu, yier.jin@ece.ufl.edu

***Abstract*—“Cyclic” circuit locking/camouflaging is a recently proposed direction in logic obfuscation for thwarting foundry and end-user reverse engineering. As opposed to traditional schemes, these techniques create cycles in the obfuscated circuit in a way that confuses the attacker but does not disrupt the combinational nature of the circuit. While these schemes can thwart the baseline SAT-based attack, the CycSAT attack was proposed recently to break these schemes through a preprocessing step that builds a Boolean condition to avoid cyclic solutions/keys during the attack. However, follow-up work has suggested that extracting these conditions requires enumerating all cycles in the circuit, or that instead of relying on these conditions preemptively, cyclic solutions must be banned individually on the fly. In this paper we present new algorithms for performing SAT-based attacks on cyclic circuits. We first propose an algorithm that can produce non-cyclic conditions in polynomial time with respect to the size of the circuit, avoiding the potentially exponential runtime of explicit key-banning or cycle enumeration. We then take a deeper look at the problem, discussing some of the fundamental limitations of extracting precise non-cyclic conditions and propose a more complex but complete procedure for cyclic deobfuscation. We evaluate our attacks on densely cyclic obfuscated benchmark circuits.**

I. Introduction

A major part of today’s semiconductor industry operates under the fabless business model in which one party performs the design and verification of the IC while another party, typically in another country, manufactures the design. This means that the physical design of an IC including all non-programmable features of the chip are revealed to a potentially untrusted foundry. This has raised several security concerns including the threat of malicious modification of the design, Intellectual Property (IP) theft, and overproduction. Not only is malicious modification of large designs easier than anticipated, due to scaling and process variation, it is becoming more and more difficult to detect such modifications post fabrication. In addition to the threat from an untrusted foundry, end-users can reverse engineer fabricated ICs through depackaging, delayering and imaging the IC in an increasingly automated fashion.

Logic locking, IC camouflaging, and split-manufacturing are three main techniques for increasing supply chain security by partially hiding the design from the foundry or end-users. Locking is based on inserting programmability into the circuit such that post fabrication a configuration of the programmable elements with a secret bitstream/key is necessary to operate the IC correctly. This hides from the foundry the complete functionality of the circuit and can protect against end-user attackers as well, if the programmable device technology is tamper-resistant. IC camouflaging is based on inserting nano-device structures in the circuit that are difficult to recognize by end-user attackers through microscopy-based reverse engineering while providing no protection against an untrusted foundry. Split-manufacturing is based on fabricating upper metal layers or another part of the design in a trusted foundry, hiding partly design information from the untrusted foundry.

All three of the above design-hiding techniques are difficult to secure with low overhead when an attacker has access to a functional/unlocked IC. The attacker can use this functional IC as an oracle to extract correct input-output pairs which can be used to resolve the ambiguity in the netlist. The most powerful and systematic of such “oracle-guided” attacks are the SAT attacks [1], [2] which use SAT solver calls to adaptively find input patterns to query on the oracle and incrementally deobfuscate the netlist.

There has been several constructions for locking/camouflaging that thwart the SAT attack by exponentially increasing the minimum number of queries required to learn the circuit [3], [4]. This is done by ensuring that each query can help disqualify only an exponentially small portion of the hypothesis space of the attacker. However, these schemes inevitably result in a low error-rate meaning that the attacker can approximate the functionality of the original circuit with approximation attacks [5], [6].

Another branch of research on SAT-resilient locking/camouflaging was initiated by Shamsi et al. [7] by deliberately adding dummy cycles to combinational directed-acyclic-graph (DAG)-circuits for the purpose of obfuscation. These added cycles can cause problems for the baseline SAT attack. However, Zhou et al. [8] proposed a novel preprocessing technique that can allow the SAT attack to deobfuscate cyclic circuits. The preprocessing step is based on extracting a condition from the obfuscated circuit that ensures that the attack search only operates within the subset of the solution space that produces acyclic circuits. This condition was called the non-cyclic (NC) condition.

It was later observed that the method described in [8] for extracting NC conditions may require a traversal of all cycles in the circuit [9], [10] to construct with effective precision. This limitation was exploited in several cyclic protection schemes to thwart oracle-guided attacks [9], [10]. The BeSAT attack was then proposed in [11] which operates by detecting cyclic solutions as they arrive during the attack and explicitly banning them from the SAT solver.

Cyclic deobfuscation is not only important to the analysis of cyclic locking/camouflaging schemes, it is critical to analyzing the security of any scenario where an attacker wants to fully reconstruct unknown circuit interconnects through input-output queries. For instance, the security of split-manufacturing was assessed by formulating a cyclic deobfuscation problem in [12] but solved using the existing inefficient cycle enumeration approach. In this paper we improve upon the state-of-art in cyclic deobfuscation. We specifically deliver the following contributions:

- We first show how explicitly banning cyclic keys as used in BeSAT can easily end up performing exponential computation on small and simple cyclic circuits.
- We present a technique for extracting non-cyclic conditions with time and space complexity $O(|W|.S)$ where W is a feedback-arc-set, and S is the size of the obfuscated circuit. We show how this technique avoids the potentially exponential runtime of BeSAT and cycle enumeration by performing the attack on densely cyclic benchmark circuits.
- We present a novel study of the deeper limitations of non-cyclic conditions including the impreciseness of sensitivity analysis which is a main part of these attacks, plus the case where the original circuit itself is cyclic (cyclic+cyclic obfuscation). To address these limitations, we present a more complex but complete cyclic deobfuscation procedure based on circuit unrolling.

The paper is organized as follows Section II presents preliminaries, Section III presents our efficient NC condition extraction algorithm. Section IV presents discussions and algorithms beyond NC conditions. Section V presents experimental results, and Section VI concludes the paper.

II. Preliminaries

A. Oracle-Guided Attacks

Logic locking can be modeled as inserting additional programmable inputs called key-inputs into the circuit. Formally this can be modeled as taking the original circuit $c_o : I \rightarrow O$ where $I = \{0,1\}^n$, $O = \{0,1\}^m$ and transforming it to a locked/obfuscated circuit $c_e : I \times K \rightarrow O$ where $K = \{0,1\}^l$ is the key-space and there exists a correct key $k_* \in K_* \subset K$ for which $c_e(i, k_*) = c_o(i)$ for all $i \in I$. IC camouflaging can be modeled with this formalization as well by encoding all the ambiguity in the netlist with additional key-variables. It can be shown that for digital ambiguity this should be possible with a polynomial number of key-variables. Split-manufacturing can also be modeled this way by replacing the missing interconnections of n nets with a network of n-input multiplexer (MUX) gates that are controlled by $\log(n)$ number of unknown key-bits [12].

With respect to the above formalism, the oracle-guided attack model assumes the attacker has white-box access to the obfuscated circuit c_e, and black-box access to the oracle circuit c_o that can be queried with arbitrary x to obtain $c_o(x)$. The SAT attack [1], [2] operates under this model. It begins by forming a mitter circuit/formula $M \equiv (c_e(x,k_1) \neq c_e(x,k_2))$. The formula is satisfied with $\hat{x}$, $\hat{k_1}$, $\hat{k_2}$ using a SAT solver. $\hat{x}$ which is referred to as a discriminating-input-pattern (DIP) is queried on the oracle and the observation from the output is added back to the SAT solver as a new constraint. The procedure stops when the mitter+constraints can no longer be satisfied in which case the constraints should return a correct key. The baseline SAT attack can be seen in Algorithm 1.

Algorithm 1 With oracle access to c_o and the circuit c_e return a correct key $k_* \in K_*$.

```
1: function OGSATAttack($c_e$, $c_o$ as black-box)
2:     $F \leftarrow true$, $M \leftarrow c_e(x,k_1) \neq c_e(x,k_2)$
3:     while $F \wedge M$ is solvable do
4:         Solve $F \wedge M$ with $\hat{x}$, $\hat{k_1}$, $\hat{k_2}$
5:         $\hat{y} \leftarrow c_o(\hat{x})$
6:         $F \leftarrow F \wedge (c_e(\hat{x},k_1) = \hat{y}) \wedge (c_e(\hat{x},k_2) = \hat{y})$
7:     satisfy $F$ with $\hat{k}_1$ and $\hat{k}_2$
8: return $\hat{k}_1$ as a correct key $k_*$
```

B. Baseline SAT Attacks on Cyclic Circuits

An example of cyclic locking can be seen in Figure 1. If the baseline SAT attack is run on this circuit pair, the cyclic relation $c_e(x,k)$ will be directly added to the SAT solver. If the attack is lucky, the SAT solver will never land on a cyclic key and the SAT attack will conclude correctly. With high probability however, the attack can run into issues. For instance, if $(\hat{k_2}, \hat{k_3}) = (1,1)$, the input x_0 gets disconnected from the output, and the output becomes dependent on the initial state of the wire e. At this point the solver can keep satisfying the mitter condition M in an infinite loop just by changing the initial state of e rather than changing the input x_0 and learning new information about the key.

C. The Cyclic SAT Attack (CycSAT)

The "Structural"-CycSAT attack [8] which we will simply call CycSAT here, is for deobfuscating cyclic circuits where the original circuit itself is acyclic. The attack is based on placing a condition on

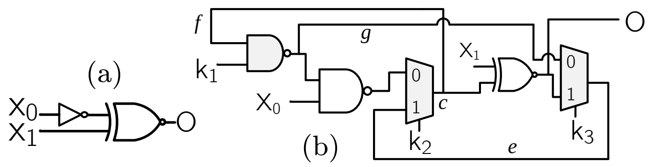

Fig. 1: (a) The original acyclic circuit. (b) Cyclic locking with three added key bits.

the key, banning the solver from landing on cyclic keys. Extracting this condition is the critical part of the attack. In the CycSAT attack, first a feedback-arc-set $W = \{w_0, w_1, ..., w_p\}$ is discovered which can be done with a simple depth-first-search (DFS) starting from inputs. A feedback-arc-set is any set of edges in a directed cyclic graph that if opened will turn the graph acyclic. These wires are then opened each into an input w_i and an output w_i'. The NC condition is defined as $\bigwedge_{i\in 0..p} F(w_i', w_i)$ where $F(w_i', w_i)$ is the condition on the key such that there is no structural path from w_i leading to w_i'. The approach for constructing $F(w_i', w_i)$ was to recursively compute $F(w_i', w_i) = \bigwedge_{l\in \mathrm{NK}(w_i')} F(w_i', l) \vee bk(l, w_i)$ where $\mathrm{NK}(w_i')$ is the non-key fanins of the wire w_i' and $bk(l, w_i)$ is the condition on the key that will block w_i from affecting l.

D. Behavioral Cyclic SAT Attack (BeSAT)

A critical challenge when extracting the NC condition is that when computing $F(w_i', w_i)$, the logic cone that connects w_i and w_i' may include other feedback edges and worst it may itself be cyclic. When these edges are encountered during the traversal, if we assume that they are open, this loosens the NC condition and certain cycles may not be blocked by it [9]. For instance in Figure 1b during the backward traversal for feedback c, if we assume that e is already open, the NC condition will allow the cyclic solution $(k_2, k_3) = (1,1)$.

BeSAT [11] overcomes this issue by analyzing in each iteration of the SAT attack if the keys $\hat{k}_1$ and $\hat{k}_2$ create cyclic circuits. This is done by first checking if the output of two copies of the circuit with the same inputs and keys, $c_{e1}(x, \hat{k}_1)$ and $c_{e2}(x, \hat{k}_1)$, can be made different. If true the circuit has internal floating nodes and is cyclic. Second, a ternary simulation test from Malik in [13] is used to test for cyclicness of the keys. If a key is deemed cyclic, it is explicitly banned from the SAT solver by adding the condition $k \neq \hat{k}_{cyc}$.

III. Improved Cyclic SAT Attacks (IcySAT-I)

In this section we focus on the case where the original circuit is not cyclic, i.e. feedback paths can be opened by setting the correct key.

A. On the Limitations of BeSAT

We begin first by showing how BeSAT's approach of explicit key-banning can result in exponential runtime and exponential size formulae given small cyclic circuits. A simple way to cause BeSAT (or any attack that is based on individual key exclusion) to perform exponential work, is to integrate some comparator logic into the key. For instance, take a feedback MUX gate $m(k_0, fb, x_1)$ that will select the feedback signal fb if $k_0 = 0$. The cyclic solution for a circuit with one such feedback multiplexer occurs only when $k_0 = 0$ and hence BeSAT can find the correct key in one iteration. However, if we replace k_0 in the MUX gate with $t_0 = P(k_v, k_v^*)$ where k_v is a v-element vector and P is a comparator so that t_0 is only 1 when $k_v = k_v^*$, now there are $2^v - 1$ possible key patterns that result in a cyclic circuit and the SAT solver in the worst case will pick all of these cyclic keys before landing on the acyclic one k_v^*, even though k_v^* is hard-coded into the circuit's logic.

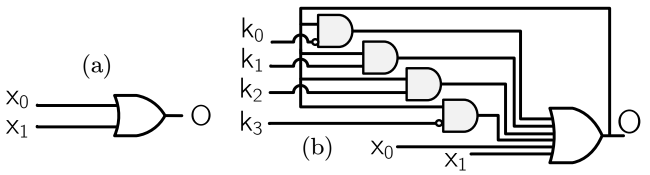

Fig. 2: (a) The original acyclic circuit. (b) Cyclicly locked circuit which will take BeSAT 2^4-1 iterations to resolve in the worst-case even though the non-cyclic key can be inferred from the structure.

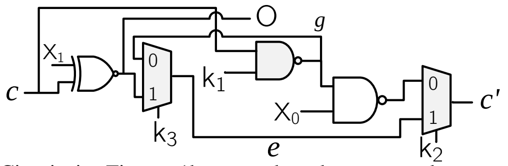

Fig. 3: Circuit in Figure 1b opened and rearranged to analyze the dependence of the feedback edge c.

The above example relies on using a key-comparator. This however is insecure from the perspective of functional security. Functional secrecy in logic locking/camouflaging demands that the obfuscated circuit's functionality be difficult to learn rather than the key per se. From this standpoint, any logic cone that operates solely on key-inputs can be removed from the circuit and replaced with a single new key-variable at the tip of the cone. The circuit locking shown in Figure 2 however, shows how the same concept of "exponentially many cyclic keys" can be implemented without key-exclusive logic that will spell trouble for the BeSAT attack. In the circuit in Figure 2b each of the feedback paths need to be broken up by setting the corresponding keys to a particular value. Note that while BeSAT can take up to 2^4-1 key-banning iterations to find the non-cyclic key, the original CycSAT attack, as part of computing the NC condition will in fact discover the particular key pattern that results in an acyclic circuit, which happens to also be the overall correct key.

While the two examples discussed above were contrived to create many non-cyclic keys, if we expand our scenario to a case with more feedback arcs, the possibility of exponential key-banning increases naturally. When studying a particular opened feedback wire w_i, we are interested in the logic cone that connects w_i to w_i' which we can represent with $w_i' = g_i(w_i, x, k)$ where g_i is a function of primary inputs x and key inputs k as well as w_i. It is easy to see that the worst-case key-banning iteration complexity of BeSAT is equal to the number of vectors of k for which w_i' is dependent on w_i which is an exponential function of the key-length for randomly selected g_i. Note that the DIP-complexity of BeSAT will be similar to CycSAT and baseline SAT attack counterparts as demonstrated in [11].

B. On the Limitations of CycSAT

We now return our focus to the original (structural)-CycSAT attack and its limitation. Consider the circuit in Figure 1b. The first step in the CycSAT attack is to find a feedback-arc-set. One such set is $\{c\}$. Next is to extract a condition so that c does not affect c'. The circuit rearranged with opened c can be seen in Figure 3. As can be seen in this case, the cone $c' = g_c(c, x, k)$ is acyclic. Hence, CycSAT can easily extract a non-cyclic condition from this cone by traversing from c' to c through different paths in the circuit.

Now imagine if we pick the feedback-arc-set $\{c, e\}$. This is similar to the previous case except now when analyzing $F(c', c)$, the cone $c' = g_c(c, x, k)$ includes the edge e which is also a member of the feedback-arc-set. We are now faced with a dilemma. When constructing $F(c', c)$ should we open the edge e into e and e' and use $F(e', e)$? If yes, what should be used as $F(e', e)$ when

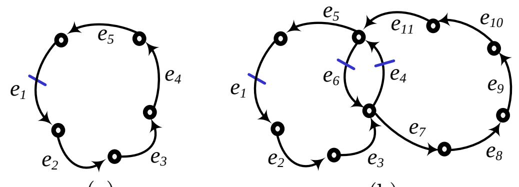

Fig. 4: (a) A single cycle. (b) Two intertwined loops show the impossibility of avoiding interdependence among feedback-arcs.

building $F(c', c)$? Furthermore, whatever dilemma we are facing when building $F(c', c)$ with respect to $F(e', e)$, we will face the same question when later building $F(e', e)$, since $F(c', c)$ depends on $F(e', e)$ and $F(e', e)$ depends on $F(c', c)$.

The above interdependency between the feedback-arc break conditions can be shown on the graph in Figure 4a. Here we assume there is a single cycle in the circuit. If we break this circuit at e_1 (the feedback-arc-set $\{e_1\}$) we can compute the NC condition by traversing the cycle easily. However, since in the CycSAT attack any feedback-arc-set can be selected, we can add e_3 to the feedback-arc-set, in which case there will be an interdependency between $F(e_1', e_1)$ and $F(e_3', e_3)$. This in fact can be exacerbated to an n-way interdependency when using a feedback-arc-set with n edges on a single cycle.

One may be tempted to think that picking a minimum feedback-arc-set or a more judiciously selected arc-set can alleviate the interpendence problem. However, as soon as the graph becomes slightly complex, it is easy to see that no selection of edges will help avoid this problem. Take the graph in Figure 4b. Here if we break either one of the outer rings, we still need to break e_6 or e_4 to break all cycles in which case there will be interdependency among them. This simple example is in fact a counter-example that can help prove the following theorem:

THEOREM 1. *It is not always possible in a directed graph $G(E, V)$ to find a feedback-arc-set W where for all $w \in W$ all paths in $G'(E - \{w\}, V)$ from $w.v_1$ to $w.v_2$ avoid all edges in $W - \{w\}$.*

C. Polynomial-Time Construction of Non-Cyclic Conditions

We begin presenting our algorithm by focusing first on the simple case of extracting NC conditions for an edge $w_i' = g_i(w_i, x, k)$ when g_i is acyclic and void of other feedback arcs. For this task, CycSAT proposes recursively computing the following:

$$F(w_i', w_i) = \bigwedge_{l \in \mathrm{NK}(w_i')} F(w_i', l) \vee bk(l, w_i)$$

where $bk(l, w_i)$ is the condition on the key that l does not depend on w_i. A direct implementation of this recursive procedure will end up traversing all paths in the cone g_i starting from the tip of the cone emitting clauses on k for each of the traversed paths. This can result in a blow-up of added clauses which was discussed in [8] and the proposed solution was to add intermediate variables to reduce the clause count. While it was proven in [8] how generating a polynomial condition is possible, a systematic approach for how this is integrated into the algorithm was not provided.

In this paper, we create the independence condition for cone g_i by building a circuit nc_i that represents the $F(w_i', w_i)$ condition and can be converted to CNF. Each wire in nc_i correspond to a wire of g_i. The gates in nc_i are constructed by transforming the corresponding gates in g_i based on dependency rules in addition to a partition on the inputs of the gates specifying which inputs are "blockers", and which are "blockees". For instance, for an AND gate $\mathrm{AND}(k_0, x_0)$ in g_i, if

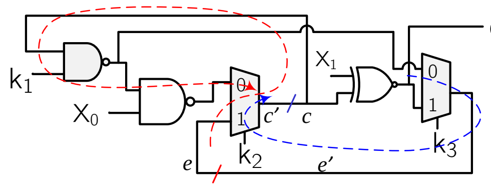

Fig. 5: Circuit in Figure 1b when building the NC condition for feedback set $\{c, e\}$ with the assumption that other feedback arcs are closed.

k_0 is a blocker and x_0 is a blockee, the corresponding (super)-gate in nc_i, representing the condition on the blocker k_0 that the output of the gate does not depend on the blockee input x_0, is $\text{OR}(\text{NOT}(k_0), x_0)$. The output of this new gate will be 1 (meaning it does not depend on w_i) if either the blocker k_0 is 0 so that it kills x_0, or that x_0 is 1 (non-w_i-dependent) in the first place. If all inputs of a gate are blockers the gate is not changed. If all inputs are blockees, the tranformation produces an AND of inputs (if one of them is 0/w_i-dependent then the output is 0/w_i-dependent). These transformations can be listed for the various n-input gates and used to build nc_i from g_i. This approach corresponds to a dynamic-programming (DP)-based implementation of the recursive procedure in [8] with a time/formula-size complexity of $O(|g_i|)$ rather than $O(\#\text{paths}(g_i))$. Figure 6b shows an example. Line 13 in Algorithm 2 performs this transformation.

The next challenge is dealing with the feedback-arc interdependency and the cyclicness of some g_is. We overcome this issue by taking the conservative approach and assuming that no other feedback-arcs are open when analyzing a particular arc. When building the independence condition $F(w'_i, w_i)$ given the cone g_i, under the assumption that other feedback-arcs are *not* open, there are a few possibilities: 1) The cone g_i contains no other feedback-arcs and is hence acyclic, in which case we can easily extract nc_i. 2) The cone g_i includes other feedback-arcs but is not cyclic, in which case we proceed similar to case 1. 3) The cone g_i includes other feedback-arcs and is cyclic itself. In this case we similarly extract an nc_i circuit from g_i even though g_i and hence nc_i are cyclic.

While the first two cases are intuitive, it seems strange that in the third case we allow the DP-based independence procedure to operate with a cyclic circuit. The empirical idea however, is that the extracted cyclic nc_i represents the condition that w_i will not affect w'_i *through a cyclic flow-graph*. This condition is tighter than the NC condition extracted under the assumption that other feedback-edges in g_i are simply open. This difference in practice is sufficient for extracting correct keys in cases where CycSAT can fail including the examples in [11], [9], all of the cyclicly locked circuits in our experimentation, and split-manufacturing cases. Figure 5 shows an example. Here the feedback-arc-set is $\{c, e\}$. When constructing the non-cyclic condition for edge c, we are in case 2, where we arrive at the other edge e in the feedback set, however, the cone g_c is acyclic. In this case rather than assuming e will be open, we continue exploring through e along the blue line, blocking this path in the process which helps avoid $(k_2, k_3) = (1, 1)$.

We then perform the independence analysis on edge e. In this case the cone g_e is cyclic if we assume that c is not open and hence this falls in case 3. Here if k_2 is 1, e will have to be blocked at k_3 or k_1. If k_2 is 0, e will be blocked immediately. Hence the NC condition extracted from analyzing e does not disrupt the condition already extracted from c and the attack returns a correct key.

Our first attack called IcySAT-I, follows the above procedure to extract NC conditions by converting g_is to nc_is and conjoining them.

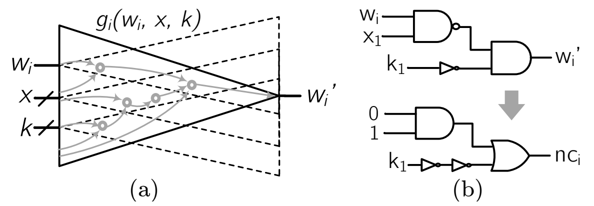

Fig. 6: (a) Fanin cone of feedback-arc w'_i partitioned according to w_i, x and k's fanout sets. (b) Transforming a cone g_i to a circuit nc_i representing the condition on the blocker k_1 for independence of w'_i and w_i. nc_i will be one when w'_i is independent of w_i.

Algorithm 2 Given w a feedback wire construct the circuit that represents the $F(w', w)$ condition on the key.

```
1:  function IcySAT-I_NCcompute($w$ a feedback arc)
2:      $F(k) \leftarrow$ NewCircuit(); $w' \leftarrow F$.NewWire(); $M \leftarrow$ Map()
3:      for $g \in$ BackwardBFSGates($w$) do
4:          for $u \in$ GateInputs($g$) do
5:              if $u \in$ TransFanout($w$) then
6:                  $M[u] \leftarrow F$.NewWire(); Blockee.add($u$)
7:              else if $u$ is key then
8:                  $M[u] \leftarrow$ MitterKey($u$); Blocker.add($u$)
9:              else if $u \in$ KeyExclusiveLogic() then
10:                 $M[u] \leftarrow F$.NewWire(); Blocker.add($u$)
11:             else if $u = w$ then $M[u] \leftarrow F$.Const0(); Blockee.add($u$)
12:             else $M[u] \leftarrow F$.Const1(); Blockee.add($u$)
13:         $F(k)$.AddTransformedGate($g$, Blocker, Blockee, $M$)
14: return $F(k)$ as non-cyclic condition circuit for $w$
```

As per Figure 6a the wires in the cone are partitioned to blocker, blockee based on their membership in the fanout sets of x, k, and w_i. w_i itself is tainted with 0 and its propagation to the output should be blocked by demanding that the output be 1/non-w_i-dependent. Figure 6b shows an example of this. Algorithm 2 details the procedure.

IV. Cyclic Targets and Beyond (IcySAT-II)

IcySAT-I can fail in cases where the original circuit itself is cyclic. Additionally the kind of dependency analysis performed in IcySAT-I, ternary simulation, and CycSAT, are fundamentally imprecise. We discuss these issues herein.

A. On Logic-CycSAT

In IcySAT-I and the structural-CycSAT attack, the NC condition for an edge is a condition on the key only. In IcySAT-I any wire in the cone $g_i(w_i, x, k)$ that is solely controlled by x was simply assumed to not be related to w_i when building nc_i (circuit for $F(w'_i, w_i)$). Per Figure 6b and Line 12 of Algorithm 2, we set x_1 and any non-key-controlled wire outside of the fanout set of w_i to 1/non-w_i-dependent. When the original circuit is cyclic-yet-combinational, typically, the cycles in the original circuit are structural-only and break up under all possible inputs. In this case, for the feedback-arc-set W, for all $w_i \in W$, $w'_i = g_i(w_i, x, k_*)$ will not depend on w_i for any input x under the correct key k_*. In this case the x-controlled variables in the fanin cone of w'_i (e.g x_1 in Figure 6b) become important as they themselves help break up certain cycles in the circuit.

Logic-CycSAT proposed in [8], suggested handling this case by defining the NC condition over both the keys and inputs for breaking cyclic+cyclic locking. The idea is to extract $NC(x, k)$ which allows x to participate in breaking cycles along with k. IcySAT-I can simply be slightly modified to extract such a condition as well. Rather than setting all wires in the cone g_i that are exclusively in the fanout of

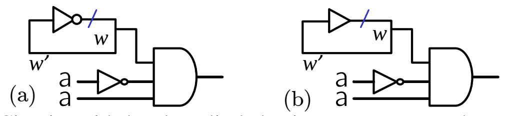

Fig. 7: Circuits with local cyclic behavior, not propagated to the output. (a) Internal oscillation. (b) Internal statefulness.

inputs x to 1, these wires can be included as blockers which will help extract $NC(x,k)$ in polynomial time.

In [8] it is suggested that $NC(x,k)$ will constrain x, limiting the attack to particular input patterns. Hence subsequent to extracting $NC(x,k)$, a second phase follows to find input patterns that are outside of $NC(x,k)$ and patch the NC condition for said input patterns. However, fundamentally, $NC(x,k)$ if extracted properly, captures a condition on inputs and keys for which the feedback arcs are broken. Hence such a condition should not constrain x at all. This is because there will exist at least one key $\hat{k} \in K_*$ which satisfies $NC(x,\hat{k})$ regardless of x as the circuit should naturally break all of its cycles and the second phase must be unnecessary.

The above approach can work in cases where the feedbacks in the original circuit are always broken by the inputs. However, we can have a cyclic circuit that has internal floating or even oscillating nodes, where these nodes despite having local sequential/cyclic behavior, do not affect the output under any input pattern. Take the circuit in Figure 7b. Signal w is clearly a cyclic/floating net. If we cut this net and form an NC condition to make w' not depend on w, this NC condition will simply be unsatisfiable as there is no way to block w from affecting w'. The case in Figure 7a is even worst as the inverter will cause an internal oscillation and will yield an unsatisfiable formula as soon as it is included in the cone g_i of any other feedback wire as well. Note however that such oscillations will waste power, heat up the device, and add noise to other signals and therefore are not a great defense strategy.

We can modify the IcySAT-I NC condition to handle the above cases. We can demand that each feedback wire either not affect itself, *or* not affect the outputs. This can be done by building $nc_i = g_i \vee go_i$ where go_i is a multi-output dependency circuit that tracks the effect of w_i on the outputs. Attacks using such complex independence conditions are fragile however due to the impreciseness of the procedure as we discuss in the following subsection.

B. Impreciseness of Independence Conditions

An important part of the cyclic SAT attacks discussed so far is encoding the dependence of a function on a particular variable into a formula that can be appended to the SAT solver without exponential work. Namely, for the function $g_i(w_i, x, k)$ we were interested in extracting a condition on x, or k, or both to ensure that w_i does not affect the output of g_i. This was done using *taint-propagation*. The idea is that we taint the input w_i (i.e. in the case of IcySAT-I we tie it to 0/w_i-dependent). The circuit nc_i then represents the propagation of this taint through g_i to w'_i using *propagation rules*. An example of such rules is that we assume that if a gate has two inputs that are dependent on w_i, the output of the gate must also be dependent on w_i. BeSAT uses ternary simulation in a similar way. The broken feedback signals are set to X (tainted as unknown), then the X taints are propagated through the circuit with propagation rules, one of which is that if a gate has two inputs that are X, the output is definitely going to be X and unknown as well.

It is trivial to show that in fact this form of taint-propagation is not accurate. Take Figure 7a. The AND gate has two inputs $\bar{a}$ and a. Both of these inputs depend on a, hence both the IcySAT-I approach and ternary simulation will suggest that the output of this gate is also going to be dependent on a. However, we know that in reality the output of the gate is going to be constant and 0.

It is easy to show that this inaccuracy does not only occur when the output of a gate is constant. Given an OR gate with input $f_1(a,x)$, and $f_2(a,x)$ where both f_1 and f_2 are dependent on a, the condition for the output of the gate to not depend on a is $f_1(0,x) \vee f_2(0,x) = f_1(1,x) \vee f_2(1,x)$ for all x. Since the condition has to hold true for all x, the two sides can be thought of as vectors and since the vectors can all be arbitrary, it is easy to see how there are an exponential number of vectors that satisfy this condition without the output of the OR being a constant 0 or 1.

The taint-propagation approach in fact *under-approximates* independence. In other words, if taint-propagation asserts that that the output of a function is not dependent on a particular input, then the function is definitely independent of that variable. If however, taint-propagation concludes that the output does depend on an input variable, it is possible for this to be false. These techniques were originally studied in the context of logic synthesis and analysis of cyclic circuits [13] where this under-approximation is not critical. However, when it comes to cyclic deobfuscation, it is easy for the defender to deliberately use these cases to hinder cyclic attacks. The defender can use simple reconvergent paths such as the ones discussed earlier in Figure 7, to create a scenario where taint-propagation-based NC condition extraction ends up excluding the correct key or generating an unsatisfiable formula.

Is there a way to improve the precision of taint-propagation? There may be some possible solutions. First consider the case for $g_i(w_i, x, k)$ where we want to extract a condition on x and k to block w_i. A simple formulation for this is $g_i(0,x,k) = g_i(1,x,k)$ which can be represented with a CNF. If g_i is acyclic, satisfying this formula will only return x and k for which w_i is killed. However, if g_i is cyclic, this approach can fail as the condition can be satisfied using the internal floating variables.

If we are looking for conditions on a subset of inputs of g_i to block w_i, the formula representing the precise independence can become a Quantified-Boolean-Formula (QBF) which requires a multitude of SAT queries to solve. E.g. the condition on k that g_i does not depend on w_i for all possible values of x is $\forall x \; g_i(0,x,k) = g_i(1,x,k)$ where x has to be removed through a universal quantifier.

Another approach for capturing true dependency is to use a Binary-Deicion-Diagram (BDD) representation. In the case of Figure 7 if we can compute a manageable-sized BDD (not always possible) of the output node, it will reveal that the variable a does not affect the output. Quantification on BDDs can help as a systematic technique for building more precise dependency conditions.

C. A Complete Cyclic Deobfuscation Procedure

The issues discussed above can render IcySAT-I useless against smarter defenses. We now discuss IcySAT-II as an attack that is a complete procedure for cyclic deobfuscation. This means that regardless of whether the original circuit is cyclic, or if the feedback dependencies are reconvergent, or if the circuit contains internal oscillations that do not leak to the output, this procedure will terminate in finite time with a guaranteed correct key.

The idea behind the attack is simple. The feedback arcs in the circuit are first broken up. This transforms the circuit $c_e(x,k) \rightarrow o$ to $c_e^b(w,x,k) \rightarrow (w',o)$. w' are new outputs and w are new inputs. c_e^b is now an acyclic circuit. Now we take c_e^b and treat it as a sequential circuit that operates on $|w|$ registers. Since the behavior of a cyclic circuit is somewhat similar to that of a sequential circuit, we can "unroll" this circuit t times to represent its operation over t time frames. Unrolling is done by replicating the circuit t times, then taking the w' wires in time frame i and feeding them to the w signals in time frame $i+1$ for all $i \in 0...t-1$. The x and k wires in all frames are connected to the same original inputs and keys. The w signals in the first frame are treated as new primary inputs and only the output signals in the last frame are used as primary outputs obtaining $c_e^u(w, x, k) \to o$. A $|w|$ number of new imaginary primary inputs are also added to the oracle, although they do not affect its output. i.e. $c_o(x) \to o$ becomes $c_o^+(w, x) \to o$. Now we can run any oracle-guided attack on the new obfuscated/original circuit pair $(c_e^u(w, x, k), c_o^+(w, x))$ to find a correct key.

In essence we are modeling the cyclic circuit as a sequential circuit with flip-flops on each feedback arc. Stating that the w signals not affect the output of the oracle means we are demanding that the initial state in this new sequential circuit not affect its output. This is equivalent to the definition of a cyclic-yet-combinational circuit for which its behavior is independent of the choice of the initial value of feedback arcs, i.e. the outputs settle eventually.

An important question that follows is how many rounds should the circuit be unrolled? To see this consider the loop in Figure 2a. One choice of feedbacks is to cut only one edge e_1 in this loop to break the loop. If this edge is the only broken edge, it is possible to see how unrolling the circuit twice (two copies of c_e) will allow the broken input signal e_1 to reach its counterpart signal e_1'. If however, two edges are broken, we need to unroll the circuit one more time to make sure that the signal e_1 can reach signal e_1' within the unrolling. If any signal w_i cannot reach its counterpart in the unrolled circuit, the unrolled model is incomplete, as the full effect of w_i is not modeled. Hence it is possible to prove that we need to unroll at least N rounds where N is the largest number of feedback edges that will be encountered between any broken feedback w_i and w_i'. Since calculating this requires solving the NP-complete problem of longest-simple-path, in our experiments we simply unroll up to $|W|$ rounds where W is the feedback-arc-set.

V. Experiments

We implemented the proposed attacks in a C++ framework using the Glucose SAT solver. The experiments presented here were run on a 96 core AMD EPYC server with 256GB of memory running at 1.9GHz. The attack tests were run in parallel. The number of simultaneous processes was limited to 90 to avoid resource sharing from skewing the data. All recovered keys were verified with equivalence checking.

We first perform tests on the c432 ISCAS benchmark to show the limitation of BeSAT. We implement BeSAT by checking whether a key is cyclic and banning it by adding a single clause to the solver. The circuit is locked with CrossLock from [14] which inserts $N \times M$ cross-bars into the circuit represented by N M-input key-controlled MUXs. The crossbars are inserted randomly and creating multiple entangled cycles in the obfuscated circuit is highly likely. The results are shown in Table I. As can be seen, for small key lengths BeSAT performs similar to IcySAT-I, however, as the number of key-bits increases the larger width of the multiplexers means that there will be many cyclic keys. BeSAT ends up trying to ban such patterns one by one as opposed to IcySAT-I which scales nicely since the non-cyclicness is being captured by a concise formula.

We then tested IcySAT-I on a set of ISCAS and MCNC benchmark circuits adopted from [1]. The circuits are locked with the original cyclic locking technique from Shamsi et al. in [7] which finds N paths of length L, feeds back their head to their tail, and inserts additional MUXs to make the edges appear removable to the attacker. We apply this technique across the benchmarks with loops of length 5, where the number of loops is selected to ensure reaching the overhead category (5%, 10%, etc.). These results are presented in Table II which show deobfuscation times within minutes for circuits with many loops.

The next tests perform the unrolling attack (IcySAT-II) on circuits that are cyclified before obfuscation. The cyclification is performed by finding AND/OR gates in the circuit, and selecting a feedback wire from their transitive fanout. Then the feedback wire is mixed with the target gate in a way similar to that in Figure 7 where the dependency analysis of IcySAT-I will under-approximate the non-cyclic condition and cause the attack to fail. 8 such edges are added to each of the circuits. In addition we inserted a few inverter loops to test the ability of the unrolling attack in deobfuscating cyclic circuits that have internal oscillations/statefulness that do not leak to the output.

To discover the number of frames that must be unrolled we go through the feedback arcs one by one and look at the logic that connects w_i to w_i'. Ideally we must find the simple path in this logic that meets the most number of other feedback wires ($w_j,\ j \neq i$). Since this problem is NP-complete, we simply resort to counting the total number of feedback wires in $g_i(w_i, x, k)$. This can end up unrolling more than necessary but will not disrupt the correctness of the attack. Note that while the minimum feedback-arc-set is an NP-complete problem, trying to at least reduce the size of the feedback-arc-set can be a good way to reduce the complexity of IcySAT-II. Subsequent to unrolling, we use ABC to simplify the circuit (several rounds of rewriting, refactoring, and SAT-sweeping) since there are many equivalent nodes in the unrolled circuit. This was able to reduce the size of the unrolled circuit by 5% to 50% across the benchmark circuits. We then perform the conventional SAT attack to recover the key. The results of this attack are shown in Table III which is consistent with the expectation that IcySAT-II should be more complex than IcySAT-I.

TABLE I: IcySAT-I compared to BeSAT for CrossLock on c432.

CB dim	#keys	**IcySAT-I** DIP-iter	**BeSAT** key-ban iter	**BeSAT** DIP-iter	**IcySAT-I** time(s)	**BeSAT** time(s)
4x4	12	4	91	5	0.06	0.06
6x6	18	8	130	7	0.10	0.09
8x8	32	13	3883223	15	0.09	3379
10x10	40	24	103946	24	0.15	1131
12x12	48	29	-	-	0.25	-
14x14	56	26	-	-	0.9	-

VI. Conclusion

In this paper we presented an efficient algorithm for generating NC conditions for cyclic deobfuscation avoiding the exponential complexity of existing techniques. We further expanded upon the fundamental limitations of NC conditions, presenting a complete procedure for cyclic deobfuscation. Faster cyclic deobfuscation can help better understand the security of a myriad of interconnect-oriented hardware protection schemes.

Acknowledgements

The work is partially supported by the Defense Advanced Research Projects Agency (DARPA) and the National Science Foundation (NSF-1812071).

References

[1] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust*, pp. 137–143, 2015.

TABLE II: IcySAT-I results. Benchmarks are locked by adding as many loops of length 5 to reach the overhead category. Dashed cells reached the time-out limit which is 1 hour. All recovered keys were verified with cyclicness testing and equivalence checking.

overhead				5 %			10 %			15 %			25 %		
bench	#in	#out	#gate	#key	#DIP	time(s)	#key	#DIP	time(s)	#key	#DIP	time(s)	#key	#DIP	time(s)
c432	36	7	160	9	5	0.06	16	8	0.09	25	11	0.08	39	12	0.22
c499	41	32	202	15	7	0.10	23	13	0.12	31	21	0.17	49	20	0.25
i4	192	6	338	19	17	0.11	39	40	0.22	58	34	0.18	85	81	1.94
c880	60	26	383	17	6	0.21	34	12	0.23	51	22	0.32	81	20	0.98
c1355	41	32	546	22	15	0.24	45	16	0.46	67	24	0.86	107	39	5.10
apex2	39	3	610	39	27	0.26	69	37	0.79	98	60	2.67	155	75	8.67
c1908	33	25	880	45	21	0.59	79	23	1.24	124	34	4.52	202	41	1586
i9	88	63	1035	56	13	0.23	103	21	0.60	148	18	0.66	235	25	3.92
ex5	8	63	1055	58	27	0.38	105	31	0.51	151	41	0.82	254	62	11.40
c2670	157	64	1193	55	26	0.47	106	36	1.23	157	80	6.73	262	108	153
i7	199	67	1315	67	24	0.56	134	42	0.88	190	38	0.59	312	39	1.86
c3540	50	22	1669	79	20	3.13	150	34	12.24	222	47	45.38	357	63	406
k2	46	45	1815	95	43	1.67	179	53	5.98	258	60	16.95	411	76	357
dalu	75	16	2298	112	38	2.83	214	70	80.08	326	94	690	531	-	-
c5315	178	123	2307	101	35	2.56	205	65	15.06	302	87	34.55	498	122	1617
i8	133	81	2464	122	30	0.88	237	60	2.58	349	52	6.24	504	88	191
c7552	206	107	3512	159	54	5.60	315	93	96.83	460	-	-	644	-	-
seq	41	35	3519	176	86	7.10	352	133	265	521	-	-	722	-	-
ex1010	10	10	5066	259	107	20.74	505	188	2301	742	-	-	742	-	-
apex4	10	19	5360	262	87	15.97	527	126	747	789	-	-	807	-	-
des	256	245	6473	302	65	11.52	600	73	373	638	70	586	638	72	780

TABLE III: IcySAT-II results. Circuits are first cyclified with 8 feedback edges. Then loops of length 5 are created until the overhead number is reached. Dashed cells reached the time-out limit which is 1 hour. All recovered keys were verified with cyclicness testing and equivalence checking.

overhead				5 %			10 %			15 %			25 %		
bench	#in	#out	#gate	#key	#DIP	time(s)	#key	#DIP	time(s)	#key	#DIP	time(s)	#key	#DIP	time(s)
c432	36	7	160	9	8	0.90	17	10	0.81	25	14	1.61	41	26	28.51
c499	41	32	202	15	9	2.99	22	11	4.46	30	14	3.04	45	34	540
i4	192	6	338	19	15	0.29	38	38	0.80	58	36	0.54	83	66	21.01
c880	60	26	383	16	9	1.09	35	14	3.79	53	19	6.77	89	19	71.24
c1355	41	32	546	21	17	7.28	44	38	31.95	65	29	67.35	104	29	527
apex2	39	3	610	38	29	2.04	66	48	15.57	95	59	45.30	150	-	-
c1908	33	25	880	49	15	76.79	84	27	273	125	32	379	204	-	-
i9	88	63	1035	59	25	6.04	107	19	8.58	152	17	11.68	246	21	152
ex5	8	63	1055	57	26	0.83	105	46	30.51	151	40	72.04	251	53	926
c2670	157	64	1193	51	39	7.65	101	65	66.49	156	137	1083	260	-	-
i7	199	67	1315	67	13	0.76	134	21	1.66	191	38	1.55	316	31	185
c3540	50	22	1669	75	27	386	148	41	2675	228	-	-	360	-	-
k2	46	45	1815	90	31	16.25	174	48	3432	256	-	-	405	-	-
dalu	75	16	2298	115	47	762	217	-	-	333	-	-	547	-	-
c5315	178	123	2307	106	35	6.71	206	48	894	298	52	3452	499	-	-
i8	133	81	2464	121	38	2.54	233	42	608	345	-	-	521	-	-
c7552	206	107	3512	162	50	607	324	-	-	472	-	-	596	-	-
seq	41	35	3519	179	70	605	354	132	20375	518	-	-	674	-	-
ex1010	10	10	5066	251	113	3270	492	-	-	719	-	-	719	-	-
apex4	10	19	5360	259	99	92.68	522	-	-	771	-	-	771	-	-
des	256	245	6473	312	62	1643	613	-	-	706	-	-	706	-	-

[2] M. El Massad, S. Garg, and M. V. Tripunitara, "Integrated circuit (IC) decamouflaging: Reverse engineering camouflaged ICs within minutes.," in *Proc. Network and Distributed System Security Symp.*, 2015.

[3] K. Shamsi, T. Meade, M. Li, D. Z. Pan, and Y. Jin, "On the approximation resiliency of logic locking and ic camouflaging schemes," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 2, pp. 347–359, 2019.

[4] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proc. ACM Conf. on Computer & Communications Security*, pp. 1601–1618, ACM, 2017.

[5] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," in *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust*, 2017.

[6] Y. Shen and H. Zhou, "Double DIP: Re-evaluating security of logic encryption algorithms," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pp. 179–184, ACM, 2017.

[7] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "Cyclic obfuscation for creating sat-unresolvable circuits," in *Proc. IEEE Great Lakes Symp. on VLSI*, 2017.

[8] H. Zhou, R. Jiang, and S. Kong, "CycSAT: SAT-based attack on cyclic logic encryptions," in *Proc. Int. Conf. on Computer Aided Design*, pp. 49–56, IEEE, 2017.

[9] S. Roshanisefat, H. Mardani Kamali, and A. Sasan, "Srclock: Sat-resistant cyclic logic locking for protecting the hardware," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, pp. 153–158, ACM, 2018.

[10] 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, Automation and Test in Europe*, pp. 85–90, IEEE, 2018.

[11] Y. Shen, Y. Li, A. Rezaei, S. Kong, D. Dlott, and H. Zhou, "Besat: behavioral sat-based attack on cyclic logic encryption," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pp. 657–662, ACM, 2019.

[12] S. Chen and R. Vemuri, "On the effectiveness of the satisfiability attack on split manufactured circuits," in *2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 83–88, IEEE, 2018.

[13] S. Malik, "Analysis of cyclic combinational circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 7, pp. 950–956, 1994.

[14] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, "Cross-lock: Dense layout-level interconnect locking using cross-bar architectures," in *Proc. IEEE Great Lakes Symp. on VLSI*, 2018.