CycSAT: SAT-Based Attack on Cyclic Logic Encryptions

Hai Zhou, Ruifeng Jiang, and Shuyu Kong
Northwestern University

ABSTRACT

Cyclic logic encryption is a newly proposed circuit obfuscation technique in hardware security. It was claimed to be SAT-unresolvable because feedback cycles were intentionally inserted under keys into the encryption. We show in the paper that even though feedback cycles introduce extra difficulty for an attacker, they can still be overcome with SAT-based techniques. Specifically, we propose CycSAT Algorithms based on SAT with different acyclic conditions that can efficiently decrypt cyclic encryptions. Experimental results have shown that our CycSAT is efficient and effective to decrypt cyclic encryptions, and we need to develop new encryptions with better security properties.

1. INTRODUCTION

Logic encryption (or logic locking) is a technique to manipulate a given combinational circuit with added key input to make sure that the encryption circuit will only function as the original one under a specific key value and it is difficult to figure out that value. It is an important problem for IP protection, IC production control, Trojan prevention, and many other applications in hardware security. Circuit camouflaging, where ambiguity is intentionally introduced in the layout to fool the reverse-engineer, can also be modeled as logic encryption with keys to encode the different possibilities [4], [6].

Even though there exist many different approaches for logic encryption [1], [2], [5], [7], [9], all of them are based on ad hoc approaches to insert extra gates with keys to the original circuit. Therefore, it should not be too great a surprise (even though it was actually a surprise to many people) that a SAT-based attack developed by Subramanyan et al. [13] can efficiently decrypt almost all of the encrypted circuits by them.

Immediately after Subramanyan et al. [13], some remedies were proposed to strengthen the existing logic encryption. Yasin et al. [15] proposed SARLock, which was inspired by the difficult case of the AND-tree discovered in [13], and ensures that each wrong key can only be excluded by one input. Xie and Srivastava [14] developed the Anti-SAT encryption, where one key has at most one wrong input, but one input may exclude many wrong outputs. However, all these remedies have extremely low error rate; both SARLock and Anti-SAT have 2^{-n} error rate (i.e. one input is wrong on each wrong key). Therefore, to protect against random guess attack, they have to be combined with traditional encryption methods.

The remedies such as SARLock and Anti-SAT combined with traditional encryptions make any SAT-based exact attack (i.e getting the correct key) exponentially expensive. However, it may be vulnerable to approximate attacks that can return a key with very low error rate. Double DIP [12] and AppSAT [10] are the first approaches for approximate attacks to logic encryption.

Different from previous approaches, Shamsi et al. [11] proposed to introduce feedback cycles into the encryption circuit to defeat the SAT-based attack. They called their approach cyclic obfuscation, which we prefer to call cyclic logic encryption in this paper, for the purpose of following the tradition [13]. To prevent structural analysis attack, their approach ensures that all the inserted cycles are irreducible and have more than one way to open. However, they simply claimed that the cyclic logic encryption is SAT-unresolvable because there are cycles in the circuit.

In this paper, we find out that it is a prejudice to assume that SAT can only handle circuits without cycle. Of course, cycles do introduce interesting phenomena and thus difficulties in the original SAT-based attack [13]. But they are not insurmontable.

We first discuss the interesting issues introduced by cycles to the SAT-based attack. A cycle in a circuit could be combinational, stateful, or oscillating. A cycle can only be either stateful or oscillating under certain input configurations. If it is not so under any input configuration, it is combinational. A combinational cycle is harmless to the original SAT-based attack. A oscillating cycle cannot be simulated by SAT so it is just stealthy. A stateful cycle may be satisfied by different values, so it may prevent the SAT-based attack from finishing.

Based on these understanding of cycles, we develop two algorithms based on SAT to decrypt the cyclic logic encryptions. The first algorithm, call Structural CycSAT Algorithm, assumes that there is at one correct key that will generate an acyclic circuit. Please note that this assumption is very reasonable and Shamsi et al. [11] satisfies this assumption. The algorithm first computes a formula to capture the condition that there is no structural cycle in the circuit under the key. Then it adds this constraint to the encryption circuit. The original SAT-based attack can finish the job on the constrained circuit.

The second algorithm is more complicated. It only uses a much weaker assumption that there is a correct key generating a combinational but maybe cyclic circuit. Similar to the first algorithm, it first computes a formula postulating that there is no sensitizable cycle in the circuit. Please note that in addition to the key input, this formula also depends on other signals. Then it constrains the encryption circuit by the formula, and runs the original SAT-based attack on the constrained circuit. However, when the iterations finish, it cannot just simply return the key generated by SAT-based attack. More iterations are needed to exclude both the stealthy oscillating cycles and the stateful cycles. The second algorithm is named Logic CycSAT Algorithm.

Since these two algorithms are similar to each other and the second one is much more complicated, we have only implemented the first algorithm. Experiments are conducted on purely cyclic encryptions of acyclic circuits, cyclic encryptions on top of acyclic circuits already encrypted by traditional encryptions. They demonstrate the effectiveness and efficiency of the CycSAT algorithm.

2. BACKGROUND AND PROBLEM FORMULATION

978-1-5386-3093-8/17/$31.00 ©2017 IEEE

2.A Conventional Logic Encryption and SAT-Based Attack

There exist many different ways [1], [2], [5], [7], [9] for logic encryption. However, they are based on the general idea of iteratively find a signal at random in the original circuit and insert a lock gate (mostly an XOR) with a key. Examples are given in Figure 1. Of course, they also try to prevent circuit analysis and simple testing-based attacks by carefully selecting the lock gate locations and doing resynthesis afterwards.

An attacker is generally assumed to have access to the encryption circuit either by reverse-engineering or through other ways. He is also assumed to have a blackbox access to the original circuit, for example, through product purchase on the market. Since almost all product ICs are sequential circuits, the combinational circuit assumption we use here assumes an access to the scan-chain.

With this attack model in mind, Subramanyan et al. [13] proposed a SAT-based attack that can effectively defeat almost all of the traditional logic encryption methods. We first give its pseudo-code here in Algorithm 1. The main step in

Algorithm 1 SAT Attack Algorithm

Input: An encryption circuit $g(x,k)$ and original boolean function $f(x)$.
Output: Correct key k^* such that $g(x,k^*) \equiv f(x)$.
1: **while** $\hat{x} = SAT(g(x,k) \neq g(x,k1))$ **do**
2: $\hat{y} = f(\hat{x})$;
3: $g(x,k) = g(x,k) \wedge (g(\hat{x},k) = \hat{y})$;
4: $g(x,k1) = g(x,k1) \wedge (g(\hat{x},k1) = \hat{y})$;
5: **end while**
6: $k^* = SAT(g(x,k))$;

the SAT-based attack is to use two copies of the encryption circuit with the same input but different keys under a given constraint to check whether it is still possible to generate different outputs. Such input patterns are called Differentiating Input Patterns (DIPs). Each DIP is then used to query the original circuit blackbox to get the correct output. The DIP with output is then used to further constrain the keys under consideration.

The idea of using DIP is to exclude at least one wrong key from consideration. However, the surprise is that many of the DIPs each may exclude a large number of wrong keys. That is the main reason for the effectiveness of the attack. Of course, if a DIP can only exclude a very small number of wrong keys, then the attack will take very long time to find the correct key. Yasin et al. [15] and Xie and Srivastava [14] explored this property to develop strengthening approaches. But since they both have an error rate of 2^{-n}, they can be defeated by approximate attacks such as Double DIP [12] and AppSAT [10].

2.B Cyclic Logic Encryption

Different from all previous logic encryption methods, Shamsi et al. [10] proposed a new way for logic encryption. In stead of using lock gates to change the signal values in the circuit, their approach suggested to introduce feedback cycles into the circuit. If cycles are broken in any different way, the circuit will (most possibly) be different from the original one. In order to protect from structural analysis attack, they also developed an algorithm that can ensure each cycle is irreducible and has multiple ways to open it.

A cycle is called reducible if it has only one entry point, i.e. the node with an edge coming from outside the cycle. A reducible cycle has a unique feedback edge, which can be safely removed. To ensure multiple ways to break a cycle, they add extra multiplexer on the cycle edges.

But their claim that cyclic encryption is "SAT-unresolvable" is only based on the simple reason that "the adversary cannot launch the existing SAT attacks, since the circuit can no longer be represented as a directed acyclic graph (DAG)." As we will discuss in the paper, the foundation of the claim is not solid.

2.C Problem Definition

In this paper, we adopt the same attack model as usual, that is, we have access to a cyclic logic encryption circuit, and also access to a blackbox original circuit. We also assume that the original circuit is combinational. The decryption problem of cyclic logic encryptions can be formulated as follows.

PROBLEM 1. *Given a black-box access to a Boolean function $f(x) : B^n \rightarrow B^p$ and its cyclic encryption $g(x,k)$ as a cyclic multilevel netlist with both original inputs $x \in B^n$, key inputs $k \in B^n$, and output $y \in B^p$, find a correct k^* such that $g(x,k^*) \equiv f(x)$.*

3. An Initial Study of SAT-Based Attack on Cyclic Encryptions

In this section, we will use a simple example to study the issues in applying the original SAT-based attack [13] on cyclic logic encryptions [11].

The first lesson we learn is that *it is a prejudice to assume that a SAT engine can only work on an acyclic combinational circuit.* When Shamsi et al. [10] proposed the cyclic obfuscation, they simply claimed that "the adversary cannot launch the existing SAT attacks, since the circuit can no longer be represented as a directed acyclic graph (DAG)." This convention was so strong that they did not even bother to try the SAT-based attack on any example in their paper. However, it may not be their fault if they had tried the open-source SAT-based attack from Subramanyan et al. [13] and found that it would not work. In the code, a topological sort is conducted on the circuit netlist before it is translated into a CNF formula, which will become an infinite loop when the circuit is cyclic! We feel that this unnecessary step was perhaps also due to the same prejudice.

Let us start our study with an encryption circuit shown in Figure 2. Here we have two inputs x_0, x_1, two keys k_0, k_1, and one output y. Note that k_0 with the MUX introduced a cycle in the circuit. However, the CNF of this cyclic circuit is a conjunction of the fellowing conditions, one for each gate:

$$\text{MUX}(k_0, y, x_0) = w, w \wedge x_1 = z, k_1 \oplus w = y.$$

You can check that any SAT engine will have no problem handling such kind of CNF.

First, we assume that the original circuit is $y = \text{NAND}(x_0, x_1)$, meaning that $(k_0, k_1) = (0, 1)$ is a correct key. Now let us see what will happen if we apply the original SAT-based attack from Subramanyan et al. [13] (minus the unnecessary topological sort). Remember that the SAT-based attack will find in each iteration an input called DIP that can distinguish the current key set. For our example in Figure 2, $(x_0, x_1) = (1, 0)$ could be the first DIP. Then the key set will be constrained by $(x_0, x_1, y) = (1, 0, 1)$. It forces $k_1 = 1$. The second iteration of SAT attack must fail to find another DIP, since $k_0 = 1$ cannot have any consistent assignment while $k_0 = 0$ can only

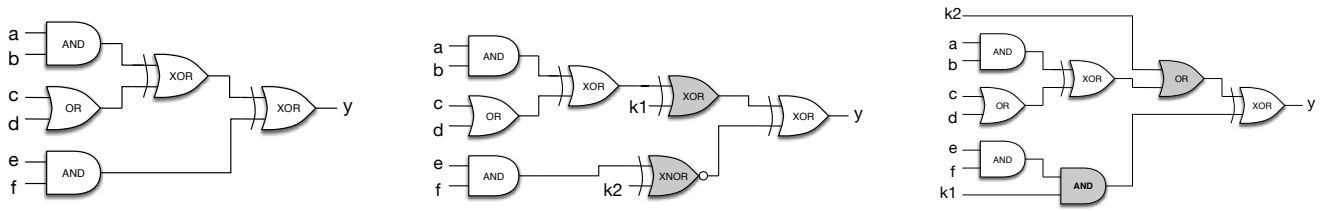

Figure 1: Traditional logic encryption example: gray gates are lock gates.

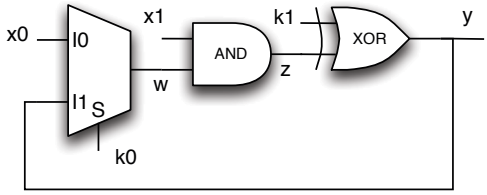

Figure 2: A cyclic logic encryption circuit.

provide a unique output. Therefore, the SAT attack will generate the key based on the only DIP $(1, 0, 1)$. The SAT engine may give us $(k_0, k_1) = (0, 1)$, the correct key, as a result. This simple analysis shows that *it is possible to directly run the original SAT-based attack on a cyclic encryption and get the correct key.*

However, there is a caveat in the above analysis, since the final result is dependent on the randomness of the SAT engine. With only one DIP $(1, 0, 1)$, it is also possible for the SAT engine to give $(k_0, k_1) = (1, 1)$ as the output, which give us a circuit shown in Figure 3. Please note that the output will oscillating between one and zero when $x_1 = 0$. This indicates that *the original SAT-based attack on a cyclic encryption may return a circuit with oscillation.*

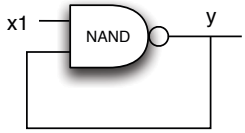

Figure 3: A circuit with oscillation.

But we may also run into different difficulties by running the SAT-based attack on cyclic encryptions. Still use the encryption circuit in Figure 2 as an example, but with a different original circuit $y = \text{AND}(x_0, x_1)$. Similarly, we could get $(x_0, x_1) = (1, 0)$ as the first DIP. Now the constraint $(x_0, x_1, y) = (1, 0, 0)$ will force $k_1 = 0$. Different from previous example, now we could get the second DIP $(0, 1)$, introducing a contraint $(x_0, x_1, y) = (0, 1, 0)$ on the key set. Then we could get a third DIP $(0, 1)$, which is the same as the second one. Thus, the SAT-based attack will get into an infinite loop, generating repeating DIPs. The reason is that after the first DIP sets $k_1 = 0$, we will have a circuit as shown in Figure 4, where no DIP can induce any constraint on k_0. With $x_1 = 1$, the output y can be either one or zero even when k_0 is fixed at one. *This seems to be a more serious problem since it prevents us from finishing the original SAT-based attack.*

Before we start to develop attack algorithms for cyclic logic encryption, let us summarize the situation for the original SAT-based attack. It is stated as the following lemma.

LEMMA 1. *If a cyclic encryption circuit is combinational under both original input and key input, then the original SAT-based attack can find the correct key. However, if it is stateful for any input and key configuration, SAT attack will get into infinite loop; if it is oscillating for any input and key configuration, SAT attack may return a wrong key.*

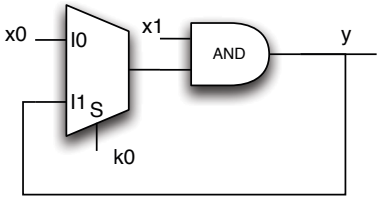

Figure 4: A circuit preventing SAT attack from finishing.

4. SAT ATTACK BASED ON STRUCTURAL ACYCLIC CONDITION

We present our first SAT-based attack on cyclic encryption based on structural acyclic condition. This attack needs to assume that there exist at least one correct key under which there is no structural cycle in the circuit. We believe that this is a reasonable assumption since the original combinational circuit is always acyclic, and a logic encryption algorithm usually gets its assurance of a correct key from its restoration back to the original circuit. In fact, Shamsi et al. [10], the only existing cyclic encryption method, does satisfy this assumption.

We will first show that the condition that "there is no structural cycle under k" can be postulated as a CNF of a size proportional to the size of the circuit. Then we can incorporate this CNF into the circuit CNF in the original SAT-based attack [13] to get an effective attack on cyclic encryptions.

It works as follows. First, we will select a small set of feedback signals whose break will make the encryption circuit acyclic. Denote the broken signals as $w_0, w_0', \ldots, w_m, w_m'$, where w_i' feeds to w_i before the break. Then, we construct the formula $F(w_i, j)$ to represent that "there is no structural path from signal w_i to signal j" iteratively in a topological order of j as follows. Initially, we have

$$F(w_i, w_i) = 0, \forall\, i \in 0..m.$$

And for any j in the topological order, we have the recursion

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

where $NK(j)$ represents the set of non-key fanins of signal j, and $bk(l, j)$ is the condition on key ensuring that l not affects j. For any gate that does not have any key input, we always have $bk(l, j) = 0$. Finally, the condition that "there is no structural cycle under k" can be formulated as

$$NC = \bigwedge\nolimits_{i=0}^{m} F(w_i, w_i').$$

To illustrate the method, let us apply the procedure to a more complex encryption circuit in Figure 5. There are more

than one possible feedback sets, for example, $\{v\}$ and $\{w, y\}$. Assume we select $\{v\}$. Now the F functions can be computed as follows.

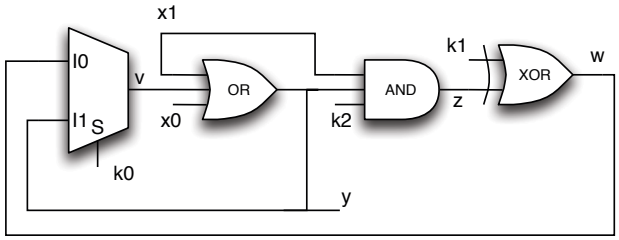

Figure 5: Another cyclic logic encryption circuit.

$$\begin{aligned} F(v,y) &= F(v,v) \vee bk(v,y) = 0 \\ F(v,z) &= F(v,y) \vee bk(y,z) = \neg k_2 \\ F(v,w) &= F(v,z) \vee bk(z,w) = \neg k_2 \\ F(v,v') &= (F(v,w) \vee bk(w,v')) \wedge (F(v,y) \vee bk(y,v')) \\ &= (\neg k_2 \vee k_0) \wedge \neg k_0 = \neg k_2 \wedge \neg k_0 \end{aligned}$$

The "no cycle" condition is $NC = \neg k_2 \wedge \neg k_0$. If NC is added as a constraint to the encryption circuit, we get a circuit $y = k_1 \vee x_0 \vee x_1$. Now let us run the original SAT-based attack on this circuit. It will find a DIP $(x_0, x_1) = (0, 0)$ to differentiate k_1, and to settle it based on the output of the original circuit. The pseudo-code is given in Algorithm 2.

Algorithm 2 Structural CycSAT Algorithm

Input: Cyclic encryption circuit $g(x, k)$ and original boolean function $f(x)$.
Output: Correct key k^* such that $g(x, k^*) \equiv f(x)$.
1: Find a set of feedback signals $(w_0, \ldots, w_m)$;
2: Compute "no structural path" formulas $F(w_0, w_0'), \ldots, F(w_m, w_m')$;
3: $NC = \bigwedge_{i=0}^{m} F(w_i, w_i')$;
4: $g(x, k) = g(x, k) \wedge NC(k)$;
5: $g(x, k1) = g(x, k1) \wedge NC(k1)$;
6: **while** $\hat{x} = SAT(g(x, k) \neq g(x, k1))$ **do**
7: $\quad \hat{y} = f(\hat{x})$;
8: $\quad g(x, k) = g(x, k) \wedge (g(\hat{x}, k) = \hat{y})$;
9: $\quad g(x, k1) = g(x, k1) \wedge (g(\hat{x}, k1) = \hat{y})$;
10: **end while**
11: $k^* = SAT(g(x, k))$;

At first look, the algorithm looks fine. However, a straightforward implementation may run into trouble. The main problem comes from the complexity of computing the "no structural path" condition. As we have shown in Equation (1), the formula of a signal can be computed from those of its fanins. However, if we try to compute the CNF of each of them by using the recursion, we may get an exponential-size formula.

Here is a simple example. Figure 6 shows an encryption circuit with a sequence of alternating MUX and OR gates, and one feedback edge from the end to the beginning. Now, let us try to compute all the formulas in CNF following the topological order based on the recursion. The output of the first MUX should have the formula $\neg k_0$, which is just one clause. Then the formula at the output of the first OR gate is $\neg k_0 \vee k_1$. With them, the formula at the output of the second MUX is $(\neg k_0)(\neg k_0 \vee k_1)$, of two clauses. In the similar way, the formula at the output of the n-th MUX will have 2^{n-1} clauses. Actually, such a CNF formula represents the all 2^{n-1} possible paths from the beginning to the curent gate, one by each clause.

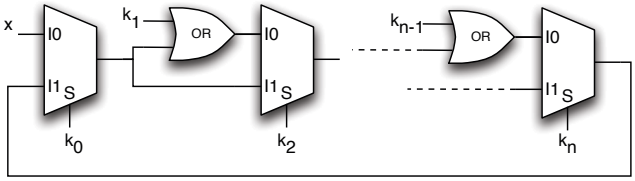

Figure 6: An cyclic encryption circuit whose CNF may have exponential clauses.

Fortunately, this exponential growth of clause number is not unavoidable. A better way is to introduce a new variable f_{ij} to represent the formula $F(w_i, j)$. Then what we need is to build a CNF to represent the relationship postulated in Equation (1):

$$f_{ij} = \bigwedge_{l \in NK(j)} f_{il} \vee bk(l, j),$$

where $bk(l, j)$ is a disjunction of one or more key literals. This can be translated into two implications. The first one $f_{ij} \Rightarrow \bigwedge_{l \in NK(j)} f_{il} \vee bk(l, j)$ is the same as the CNF

$$\bigwedge_{l \in NK(j)} (\neg f_{ij} \vee f_{il} \vee bk(l, j)).$$

However, the second one, $(\bigwedge_{l \in NK(j)} f_{il} \vee bk(l, j)) \Rightarrow f_{ij}$, equivalent to $f_{ij} \vee \bigvee_{l \in NK(j)} \neg f_{il} \wedge \neg bk(l, j)$, is actually a DNF. If we use the distributive law to directly translate a DNF into a CNF, the size could be exponential in term of $|NK(j)|$.

Once again, we can control the CNF size by introducing one auxiliary variable v_l for each $l \in NK(j)$. With them, the second implication can be translated into the following CNF:

$$(f_{ij} \vee \bigvee_{l \in NK(j)} v_l) \bigwedge_{l \in NK(j)} (\neg v_l \vee \neg f_{il})(\neg v_l \vee \neg bk(l, j)).$$

Combining the two implications gives the CNF for Equation (1).

This translation can be done for each formula $F(w_i, j)$. Combining them will give us a whole CNF for $NC(k)$. It is not hard to see that now the whole CNF representing that "there is no structural cycle under k" has a polynomial size. The result is stated in the following theorem.

THEOREM 2. *For any cyclic logic encryption circuit, the condition that there is no structural cycle under k can be computed in $O(mVE)$ time as a CNF formula of size $O(mVE)$, where V is the number of gates, E the total number of fanins for all gates, of the encryption circuit, and m is the size of the feedback set.*

5. SAT ATTACK BASED ON LOGIC ACYCLIC CONDITION

The attack in the previous section will be successful only if there is at least one correct key that renders an acyclic circuit. In this section, we are going to develop a more advanced attack that works even without that assumption. The situation without the assumption could happen if the original circuit is cyclic [3], [8], or the circuit designer intentionally adds combinational cycles before any cyclic encryption.

The approach is an extension of the algorithm from the previous section. It will first identify a small set of feedback signals $(w_0, \ldots, w_m)$ as usual. It will then generate a formula $F(w_i, w_i')$ for each $i \in 0..m$ similarly in a topological order. But now, the formula $F(w_i, j)$ postulates that "there is no sensitizable path from w_i to j."

The way to compute the formulas is the same as in previous section with only one exception that now the conditions

are now generalized from k to all signals. Specifically, the recursion for computing F is

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

where $Fanin(j)$ represents the set of all fanins of signal j, and $ns(l, j)$ is the general condition ensuring that j is not sensible to l. Finally, the condition that "there is no sensitizable cycle in circuit" can be formulated as

$$NC = \bigwedge_{i=0}^{m} F(w_i, w_i').$$

To illustrate the method, let us apply the procedure to the same encryption circuit in Figure 5. As mentioned before, there are more than one possible feedback sets, for example, $\{v\}$ and $\{w, y\}$. This time, let us assume $\{w, y\}$ is selected. The F functions can be computed as follows.

$$\begin{aligned}
F(y, v) &= F(y, y) \vee ns(y, v) = \neg k_0 \\
F(y, y') &= F(y, v) \vee ns(v, y') = \neg k_0 \vee x_0 \vee x_1 \\
F(w, v) &= F(w, w) \vee ns(w, v) = k_0 \\
F(w, y') &= F(w, v) \vee ns(v, y') = k_0 \vee x_0 \vee x_1 \\
F(w, z) &= F(w, y') \vee ns(y', z) \\
&= k_0 \vee x_0 \vee x_1 \vee \neg x_1 \vee \neg k_2 = 1 \\
F(w, w') &= F(w, z) \vee ns(z, w') = 1 \vee 0 = 1
\end{aligned}$$

The "no sensitizable cycle" condition is given by $NC = F(w, w') \wedge F(y, y') = \neg k_0 \vee x_0 \vee x_1$. Please note that this condition is much weaker than the "no structural cycle" condition $\neg k_2 \wedge \neg k_0$ from the previous section. It means that the "no sensitizable cycle" condition will allow much more possible solutions. For example $(k_0, k_1, k_2) = (0, 0, 1)$ gives such a circuit shown in Figure 7 that is not permitted by the "no strutural cycle" condition, since it does include a cycle. The complexity of the procedure is similar to the one for the Structural CycSAT algorithm.

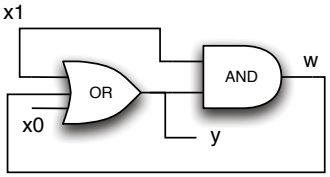

Figure 7: An original cyclic circuit that is the same as OR.

Now let us examine how the "no sensitizable cycle" condition $\neg k_0 \vee x_0 \vee x_1$ can be utilized by the SAT-based attack to find the correct original circuit in Figure 7. Constrained by this condition, the SAT needs to find a DIP to differentiate k on y. It has to fix $(x_0, x_1) = (0, 0)$, which then forces $k_0 = 0$ due to the constraint. Under this assignment, we have $z = 0$ and $y = k_1$. Thus the DIP $(0, 0)$ will make $k_1 = 0$. It can be checked that there is no more DIP. Thus, the SAT attack may return arbitrary k_0, k_2. We can check that any assignment to k_2 is correct. However, If we get $k_0 = 1$, we are in trouble since the circuit becomes stateful. This indicates that, different from the Structural CycSAT algorithm in the previous section, we cannot simply incorporate the acyclic condition into the original SAT attack to get the result.

A careful examination shows that, with consideration of both k and other signals, the "no sensitizable cycle" condition can recognize the combinational cycles in the circuit, such as the larger cycle (v, y, z, w) in Figure 5, thus be able to avoid unnecessary cycle break by $k_2 = 0$. In passing, note that the cycle behaves combinationally since x_1 will either dominate the AND or the OR gate. However, since the condition also incorporates input configurations that make cycles insensitizable, the SAT may return a key that gives sensitizable cycle outside the given input configurations. In the above example, $k_0 = 1$ is fine with $x_0 \vee x_1$. The sensitizable cycle happens only when x_0 and x_1 are zero.

Therefore, when we have extracted all DIPs under the "no sensitizable cycle" condition, we may not automatically get the correct key as in the Structural CycSAT Algorithm.

We need to develop a method to extract a correct key that guarantees "no sensitizable cycle" *for all possible inputs.* A new example in Figure 8 will be used to illustrate the issues and the method to handle them. For this circuit, the "no cycle" condition is $NC = (\neg k0 \vee \neg x \vee \neg b) \wedge (k_1 \vee \neg x \vee \neg a)$. You can check that there is no DIP. The SAT engine could return arbitrary k_0, k_1 with $x = 0$, which satisfies both NC and the circuit CNF. However, $k_0 = 1$ gives an oscillator when $x = 1$.

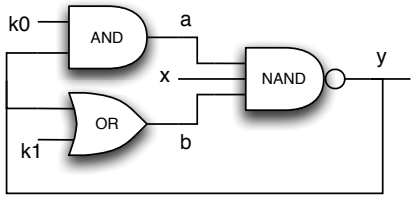

Figure 8: Yet another cyclic encryption circuit.

Our idea is to first check whether there is any input that could give sensitizable cycle in the circuit under the current constraints. Since a sensitizable cycle could be an oscillator, which does not have satisfiable assignment, we need to break each feedback signal w_i into two signals w_i, w_i'. Denote the constrained circuit $g(x, k)$ with feedback broken by $G(x, k)$, then the checking can be done by

$$SAT(\neg NC(x, k) \wedge G(x, k)).$$

If we have an assignment $\hat{x}$ on input, "no sensitizable cycle" condition NC on this specific input will be added:

$$g(x, k) = g(x, k) \wedge NC(\hat{x}, k)$$

This process will be repeated until there is no more input that can give sensitizable cycle in the constrained circuit. Then the constrained circuit $g(x, k)$ can be solved for the correct key.

For the encryption in Figure 8, we have $\neg NC = (k_0 \wedge x/\ b) \vee (k_1 \wedge x \wedge a)$. An assignment $x = 1$ satisfies $\neg NC$ and the circuit with broken y. The added constraint is $NC(x = 1) = (\neg k_0 \vee \neg b)/\ (k_1 \vee \neg a)$. You can check that there is no more assignment satisfying $\neg NC$ with the added contraint. Now, solving the constrained circuit will give us the correct $k_0 = 0$.

Please note that this procedure is similar to DIP generation in the original SAT-based attack, and we will call these input patterns SIPs (Sensitizable Input Patterns). We call this algorithm Logic CycSAT Algorithm, and its pseudo-code is given in Algorithm 3.

If we apply the Logic CycSAT Algorithm to the encryption circuit in Figure 5, we will find $(x_0, x_1) = (0, 0)$ as the first SIP. Applying this SIP to $NC = \neg k_0 \vee x_0 \vee x_1$, we get $k_0 = 0$, which will forbidden any more SIP. We will get $(k_0, k_1, k_2) = (0, \cdot, 1)$ as the correct key. You can check that it will give a circuit equivalent to the one in Figure 7.

6. Experimental Results

Algorithm 3 Logic CycSAT Algorithm

Input: Cyclic encryption circuit $g(x,k)$ and original boolean function $f(x)$.
Output: Correct key k^* such that $g(x,k^*) \equiv f(x)$.
1: Find a set of feedback signals $(w_0,\ldots,w_m)$;
2: Compute "no sensitizable path" formulas $F(w_0,w_0')$, $\ldots$, $F(w_m,w_m')$;
3: $NC(x,k) = \bigwedge_{i=0}^{m} F(w_i,w_i')$;
4: **while** $\hat{x} = SAT(NC(x,k) \wedge NC(x,k1) \wedge g(x,k) \neq g(x,k1))$ **do**
5: $\quad \hat{y} = f(\hat{x})$;
6: $\quad g(x,k) = g(x,k) \wedge (g(\hat{x},k) = \hat{y})$;
7: $\quad g(x,k1) = g(x,k1) \wedge (g(\hat{x},k1) = \hat{y})$;
8: **end while**
9: **while** $(\hat{x} = SAT(\neg NC(x,k) \wedge G(x,k)))$ **do**
10: $\quad g(x,k) = g(x,k) \wedge NC(\hat{x},k)$;
11: **end while**
12: $k^* = SAT(g(x,k))$;

6.A Benchmark Generation

Since the benchmarks from Shamsi et al. [10] are not available, we have to create our own benchmarks for testing the CycSAT algorithm. Our original acyclic circuit benchmarks include ISCAS'85 benchmarks and the combinational circuits from the Microelectronics Center of North Carolina (MCNC) as shown in table. Our traditionally encrypted circuit benchmarks are from Subramanyan et al. [13]. These benchmarks are encrypted with an encryption algorithm proposed by Dupuis et al. [2], which inserts AND and OR gates in order to minimize low-controllability locations in the circuits. With this technique, each original benchmark is encrypted into 4 circuits based on the area overhead for encryption (5%, 10%, 25% and 50%).

To create cycles in the circuits, we apply cyclic encryption following the cyclic obfuscation approach proposed by Shamsi et al. [11] on both original circuit benchmarks and traditionally encrypted circuit benchmarks. Shamsi's cyclic encryption aims to increase attack difficulty by creating irreducible loops with multiple removable edges in circuits. It is claimed that the complexity to attack a cyclic encrypted circuit with N loops of M removable edges is $2^{M\times N}$. To ensure the cycle we create is irreducible, we attempt to find a path in the circuit that has multiple entry points and create a feedback from the output of the last gate on this path to the input of the first gate with an extra multiplexer in between. The new MUX is controlled by an extra key bit so that if the correct key is provided, the feedback will be disconnected. If the selected path does not have multiple entry points, we will add MUXes along the path with extra key bits as the selecting bits and random wires not on this path as another inputs to the MUXes. Besides, if there is any gate on the path that has only one fanout, we will apply similar trick to connect its output to a random location. It turns out that all the edges on the path are removable. As can be seen, by creating one feedback loop, we may also introduce multiple unintentional cycles, further increasing attack complexity in the final encrypted circuits.

6.B CycSAT on Cyclic Encryption of Acyclic Circuits

We implement the proposed Structural CycSAT Algorithm in C++ . In this section, we compare the attacking capability and effectiveness between CycSAT and SAT-attack on the circuits with only cyclic encryption, without any traditional encryption.

Table 1: Original circuit benchmark information

ISCAS'85				MCNC			
Circuit	**#In**	**#Out**	**#Gates**	**Circuit**	**#In**	**#Out**	**Gates**
c432	36	7	160	apex2	39	3	610
c499	41	32	202	apex4	10	19	5360
c880	60	26	383	dalu	75	16	2298
c1355	41	32	546	des	256	245	6473
c1908	33	25	880	ex5	8	63	1055
c2670	157	64	1193	ex1010	10	10	5066
c3540	50	22	1669	i4	192	6	338
c5315	178	123	2307	i7	199	67	1315
c7552	207	108	3512	i8	133	81	2464
				i9	88	63	1035
				k2	46	45	1815
				seq	41	35	3519

As shown in Table 2, CycSAT successfully find correct keys for most benchmarks while original SAT-attack is often not terminating during the decryption(denoted as "-"' in the table). In all the benchmarks that SAT-attack fails, it keeps returning the same DIP and is trapped in the "while" loop in the algorithm. In benchmark **c2670** and **des**, SAT-attack returns the correct key. It is important to note that, even though not found in our experiment, the possibility that SAT-attack returns a wrong key transforming the circuit into an oscillator still in principle exists.

On the other hand, the fact that the original SAT-attack can solve the correct key of some cyclic encrypted circuits shows that cyclic encryption is not completely SAT attack resilient as claimed by Shamsi et al. [11] . The attack success relies on the randomness in the SAT engine as explained in Section 3. If the SAT engine happens to produce the desired DIP, the correct key will be generated. Otherwise, with the key constraint imposed by the undesired DIP, the circuit becomes either a state machine (as for all the failure benchmarks in Table 2) or an oscillator (rarely occurs but still possible).

Moreover, for the two benchmarks that both CycSAT and SAT-attack successfully decrypt, CycSAT takes fewer number of iterations and less CPU time, especially in benchmark **des**, which shows CycSAT's effectiveness in capturing the desired key constraint.

6.C CycSAT on Cyclic Encryption of Acyclic Encrypted Circuits

In this section, we explore the performance of CycSAT on double encrypted circuits, that is, traditionally encrypted then cyclic encrypted. All the benchmarks in our experiment have 10 percent traditional encryption area overhead. As shown in Table 3, we perform cyclic encryption to add different number of intentional feedbacks(N) with different lengths(M) for every traditionally encrypted benchmark.

We then apply CycSAT on the double encrypted circuits and obtain the attack performance as illustrated in Figure 9 and Figure 10. As expected, in most cases, it takes CycSAT more iterations and execution time to decrypt the benchmarks with more intentional feedbacks of longer lengths. However, this may not be true in some rare cases (e.g i9), where even though there are larger number of longer cycles, the decryption cost becomes smaller. According to our analysis, this can be due to the randomness of the SAT engine. Specifically, if the SAT engine fortunately returns a DIP that

Table 2: Cycsat vs Sat-attack on cyclic encryption of original circuits, with 10 intentional feedbacks of length 5

	cyclic encrypted					CycSAT		SAT-attack	
circuit	#primary inputs	#out	#gates	#key inputs	#cycles	#iterations	CPU time(s)	#iterations	CPU time(s)
apex2	39	3	695	85	40	14	0.204	-	-
apex4	10	19	5450	90	17	-	-	-	-
c432	36	7	226	66	22	-	-	-	-
c499	41	32	277	75	26	23	0.304	-	-
c880	60	26	458	75	24	25	0.208	-	-
c1355	41	32	613	67	38	-	-	-	-
c1908	33	29	953	73	30	11	0.268	-	-
c2670	233	140	1266	73	26	23	0.336	26	0.392
c3540	50	22	1745	76	38	19	0.576	-	-
c5315	178	123	2382	75	24	24	0.568	-	-
c7552	207	108	3583	71	29	26	0.852	-	-
dalu	75	16	2379	81	20	16	0.504	-	-
des	256	245	6551	78	20	18	0.908	53	2.272
ex5	8	63	1139	84	22	20	0.468	-	-
ex1010	10	10	5152	86	23	27	1.4	-	-
i4	192	6	428	90	20	29	0.172	-	-
i7	199	67	1404	89	22	40	0.688	-	-
i8	133	81	2550	86	26	-	-	-	-
i9	88	63	1121	86	26	36	0.536	-	-
k2	46	45	1895	80	18	16	0.276	-	-
seq	41	35	3605	86	20	25	0.88	-	-

Table 3: Cyclic encryption over acyclic encrypted circuits: 10% overhead traditional encryption + different cyclic encryption

	N=5, M=5			N=10, M=5			N=5, M=10			N=20, M=5		
circuit	#gate	#key	#cycle	#gate	#key	#cycle	#gate	#key	#cycle	#gate	#key	#cycle
apex2	718	108	12	762	152	32	768	158	12	851	241	54
apex4	5942	582	10	5987	627	20	5992	632	10	6077	717	40
c432	213	53	14	252	92	20	237	77	12	327	167	38
c499	262	60	16	299	97	24	305	103	22	377	175	76
c880	466	83	12	506	123	22	505	122	16	589	206	44
c1355	639	93	16	672	126	32	675	129	28	739	193	54
c1908	1008	128	28	1040	160	38	1041	161	28	1115	235	62
c2670	1372	179	16	1411	218	26	1417	224	14	1488	295	46
c3540	1879	210	20	1919	250	30	1918	249	26	1993	324	52
c5315	2595	288	12	2633	326	26	2635	328	12	2708	401	52
c7552	3912	409	10	3958	446	28	3965	453	12	4034	522	50
dalu	2576	278	10	2619	321	24	2621	323	18	2700	402	44
des	7190	717	10	7228	755	20	7234	761	10	7310	837	40
ex5	1203	148	14	1244	189	24	1251	196	14	1329	274	44
ex1010	5619	553	14	5663	597	24	5669	603	10	5750	684	44
i4	436	98	14	481	143	20	486	148	10	571	233	40
i7	1511	196	12	1556	241	22	1561	246	12	1646	331	40
i8	2768	304	16	2811	347	24	2814	350	18	2898	434	44
i9	1191	156	18	1236	201	32	1223	188	32	1326	291	52
k2	2039	224	10	2080	265	20	2091	276	12	2161	346	40
seq	3920	401	10	3963	444	20	3970	451	10	4052	533	42

excludes exponentially many wrong keys, the whole decryption process can dramatically speed up.

7. Conclusions

Cyclic logic encryption [11] is a newly proposed method different from existing logic encryption approaches. It intentionally introduces cycles into the encryption circuit to fool the attacker. In this paper, we examined the possibilities of SAT-based attacks on cyclic encryptions, and developed two efficient algorithms for that purpose. Based on theoretical and experimental studies, we found that the cyclic encryption can be effectively decrypted by the proposed CycSAT Algorithm. Therefore, it was not on a solid foundation to claim that cyclic encryption is SAT-unresolvable because SAT cannot handle a cyclic circuit. A promising direction for future work is to develop new cyclic encryption methods that will be more secure and robust. Our experiments have shown the vulnerability of the existing cyclic encryption method, and motivate us to develop new approaches in some new directions.

Acknowledgement

This work is partially supported by NSF under CNS-1441695, CCF-1533656, CNS-1651695, and by SRC under 2014-TS-2559.

8. References

[1] Baumgarten, A., Tyagi, A., and Zambreno, J. Preventing ic piracy using reconfigurable logic barriers. *IEEE Design and Test 27*, 1 (2010).

[2] Dupuis, S., Ba, P.-S., Natale, G. D., Flottes, M.-L., and Rouzeyre, B. A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans. In *IEEE International On-Line Testing Symposium* (2014).

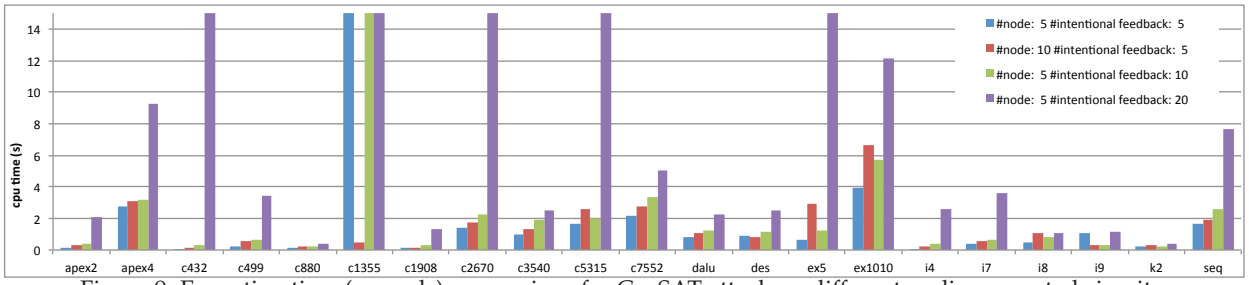

Figure 9: Execution time (seconds) comparison for CycSAT attack on different cyclic encrypted circuits.

Figure 10: Iteration number comparison for CycSAT attack on different cyclic encrypted circuits.

[3] EDWARDS, S. A. Making cyclic circuits acyclic. In *Proc. of the Design Automation Conf.* (2003).

[4] LI, M., SHAMSI, K., MEADE, T., ZHAO, Z., YU, B., JIN, Y., AND PAN, D. Provably secure camouflaging strategy for ic protection. In *Proc. Intl. Conf. on Computer-Aided Design* (Austin, TX, Nov. 2016).

[5] RAJENDRAN, J., PINO, Y., SINANOGLU, O., AND KARRI, R. Logic encryption: A fault analysis perspective. In *Proceedings of the Conference on Design, Automation and Test in Europe* (2012), EDA Consortium, pp. 953–958.

[6] RAJENDRAN, J., SAM, M., SINANOGLU, O., AND KARRI, R. Security analysis of integrated circuit camouflaging. In *CCS* (2013).

[7] RAJENDRAN, J., ZHANG, H., ZHANG, C., ROSE, G. S., PINO, Y., SINANOGLU, O., AND KARRI, R. Fault analysis-based logic encryption. *IEEE Transactions on Computers 64*, 2 (Feb. 2015).

[8] RIEDEL, M., AND BRUCK, J. The synthesis of cyclic combinational circuits. In *Proc. of the Design Automation Conf.* (2003).

[9] ROY, J. A., KOUSHANFAR, F., AND MARKOV, I. L. EPIC: ending piracy of integrated circuits. In *Proceedings of the conference on Design, automation and test in Europe* (2008), pp. 1069–1074.

[10] SHAMSI, K., LI, M., MEADE, T., ZHAO, Z., PAN, D., AND JIN, Y. AppSAT: Approximately deobfuscating integrated circuits. In *Proc. IEEE International Symposium on Hardware Oriented Security and Trust* (May 2017).

[11] SHAMSI, K., LI, M., MEADE, T., ZHAO, Z., PAN, D., AND JIN, Y. Cyclic obfuscation for creating SAT-unresolvable circuits. In *Proc. ACM Great Lakes Symposium on VLSI* (Banff, AB, Canad, May 2017).

[12] SHEN, Y., AND ZHOU, H. Double dip: Re-evaluating security of logic encryption algorithms. In *Proc. ACM Great Lakes Symposium on VLSI* (Banff, AB, Canada, May 2017).

[13] SUBRAMANYAN, P., RAY, S., AND MALIK, S. Evaluating the security of logic encryption algorithms. In *Proc. IEEE International Symposium on Hardware Oriented Security and Trust* (2015).

[14] XIE, Y., AND SRIVASTAVA, A. Mitigating SAT attack on logic locking. In *Conference on Cryptographic Hardware and Embedded Systems (CHES)* (2016).

[15] YASIN, M., MAZUMDAR, B., RAJENDRA, J. J. V., AND SINANOGLU, O. SARLock: SAT attack resistant logic locking. In *Proc. IEEE International Symposium on Hardware Oriented Security and Trust* (2016).