BeSAT: Behavioral SAT-based Attack on Cyclic Logic Encryption

Yuanqi Shen, You Li, Amin Rezaei, Shuyu Kong, David Dlott, Hai Zhou
Northwestern University
{yuanqishen2020, you.li}@u.northwestern.edu, me@aminrezaei.com,
{shuyukong2020, daviddlott2019}@u.northwestern.edu, haizhou@northwestern.edu

ABSTRACT

Cyclic logic encryption is newly proposed in the area of hardware security. It introduces feedback cycles into the circuit to defeat existing logic decryption techniques. To ensure that the circuit is acyclic under the correct key, CycSAT is developed to add the acyclic condition as a CNF formula to the SAT-based attack. However, we found that it is impossible to capture all cycles in any graph with any set of feedback signals as done in the CycSAT algorithm. In this paper, we propose a behavioral SAT-based attack called BeSAT. BeSAT observes the behavior of the encrypted circuit on top of the structural analysis, so the stateful and oscillatory keys missed by CycSAT can still be blocked. The experimental results show that BeSAT successfully overcomes the drawback of CycSAT.

1. INTRODUCTION

In rivalry global markets era, semiconductor industry faces serious challenges in Integrated Circuit (IC) design protection such as reverse engineering and inside foundry attack [1, 7, 9, 17]. The lack of efficient IC protection schemes may impose huge economic consequences to chip design companies. Accordingly, logic encryption is proposed as a protection technique to manipulate a given combinational circuit with additional key bits to make sure that the encrypted circuit functions correctly only under some specific values of those key bits.

Even though there are different encryption schemes [3, 5, 6, 8, 10], almost all of them are vulnerable to the well-known SAT-based attack [16], which can efficiently decrypt the traditional logic encryption using a few input-output observations of an activated IC. In order to make the SAT-based attack exponentially time consuming, some remedies like SARLock [19] and Anti-SAT [18] are introduced to be combined with traditional logic encryption. However, they are vulnerable to a divide and conquer attack called bit-flipping attack [14], and since these techniques have extremely low error rate, approximate attacks such as AppSAT [12] and Double DIP [15] can still decrypt the circuit by returning an almost correct key in polynomial time. To overcome these approximate attacks, the Error-Controlable Encryption (ECE) scientific benchmarks [13] are developed so that the approximate attacks perform no better than random guessing, and cyclic logic encryption [11] is proposed by adding cycles to the encryption scheme. In order to make it hard for the attacker to remove cycles, all the inserted cycles should be irreducible and have more than one way to open. However, since the circuit is still acyclic under the correct key value insertion, CycSAT [21] is proposed to incorporate a Conjunctive Normal Form (CNF) formula to the original SAT-based attack that captures the acyclic assumption.

Although CycSAT has been shown to successfully decrypt some cyclic benchmarks, the efficiency of CycSAT relies on its structurally-based nature. Precisely, it is not possible to structurally capture all cycles in any graph with any set of feedback signals. Thus in order to address the CycSAT shortcomings, we propose Behavioral SAT-based attack (BeSAT) that utilizes the behavioral analysis to conquer the missing cycle issue.

The paper is arranged as follows. Section 2 gives a descriptive preliminary about the novel concept of logic decryption. The limitation of CycSAT is then discussed in Section 3. In Section 4, we introduce statefulness and oscillation, and explain how the BeSAT algorithm can overcome these two issues. Moreover, the efficiency of BeSAT is shown with experimental results in Section 5. Finally, Section 6 concludes the paper.

2. PRELIMINARIES

In this section, firstly, we introduce the SAT-based attack [16]. Then we illustrate CycSAT [21], which adds the structural analysis on the top of the SAT-based attack.

2.1 SAT-based Attack

The model of the SAT-based attack assumes that the netlist of the encrypted circuit can be obtained, and attackers are able to buy an activated circuit from the market, so the correct output can be evaluated. The algorithm of the SAT-based attack is shown in Algorithm 1.

$c(x, k)$ represents the Conjunctive Normal Form (CNF) of the encrypted circuit with input x, key k. The SAT-based attack iteratively finds the Distinguishing Input Pattern (DIP) to differentiate two keys, since at least one of them generates the wrong output. When a DIP $\hat{x_i}$ is found, its corresponding output $\hat{y_i}$ can be evaluated by the activated circuit, and this input-output pair is used to constrain keys by adding

ASPDAC '19, January 21–24, 2019, Tokyo, Japan

DOI: https://doi.org/10.1145/3287624.3287670

Algorithm 1 SAT-based Attack

Input: The encrypted circuit $c(x, k)$ and an activated circuit $f(x)$.
Output: Correct key k^* such that $c(x, k^*) \equiv f(x)$.

```
while x̂ = SAT(c(x, k1) ≠ c(x, k2)) do
    ŷ = f(x̂);
    c(x, k1) = c(x, k1) ∧ (c(x̂, k1) = ŷ);
    c(x, k2) = c(x, k2) ∧ (c(x̂, k2) = ŷ);
end while
k* = SAT(c(x, k1));
```

$(c(\hat{x}_i, k_1) = \hat{y}_i) \wedge (c(\hat{x}_i, k_2) = \hat{y}_i)$ to the existing CNF. The CNF formula cannot be satisfied anymore when DIP cannot be found, and the algorithm terminates. Hence, a key that satisfies the current constraints will be returned and considered as the correct key.

The SAT-based attack can efficiently decrypt many existing encryption techniques using only a few iterations, since all of the wrong keys can be excluded by a small number of DIPs.

2.2 CycSAT

CycSAT holds the assumption that no structural cycle is present in the circuit under at least one correct keys. It postulates the condition that "there is no structural cycle under k" into a CNF of a size proportional to the size of the circuit. With this acyclic condition incorporated into the circuit CNF, CycSAT can leverage the original SAT-based attack to easily solve the correct key. The pseudo-code of CycSAT is given in Algorithm 2.

Algorithm 2 the CycSAT Algorithm

Input: The cyclic encryption circuit $c(x, k)$ and an original Boolean function $f(x)$.
Output: Correct key k^* such that $c(x, k^*) \equiv f(x)$.

```
Find a set of feedback signals (w0, ..., wm);
Compute "no structural path" formulas F(w0, w0'), ..., F(wm, wm');
NC = ⋀_{i=0}^{m} F(wi, wi');
c(x, k1) = c(x, k1) ∧ NC(k1);
c(x, k2) = c(x, k2) ∧ NC(k2);
while x̂ = SAT(c(x, k1) ≠ c(x, k2)) do
    ŷ = f(x̂);
    c(x, k1) = c(x, k1) ∧ (c(x̂, k1) = ŷ);
    c(x, k2) = c(x, k2) ∧ (c(x̂, k2) = ŷ);
end while
k* = SAT(c(x, k1));
```

In order to compute the acyclic condition as a CNF formula, CycSAT first breaks all the cycles in the encrypted cyclic circuit by breaking a set of feedback signals. Then for every broken feedback w, w', the formula $F(w_i, w_i')$ representing "there is no structural path from signal w to signal w'" is computed iteratively, following the topological order from w to w'. Finally, the condition that "there is no structural cycle under k" can be formulated by the conjunction of all such $F(w, w')$.

3. SHORTCOMINGS OF CYCSAT

By first glance, the way of constructing the acyclic condition in the CycSAT algorithm seems fine. However, since all the feedbacks are broken, only cycles with one feedback will be included in the acyclic constraint. Some cycles with more than one feedback edges can be missing in the analysis. We use the circuit shown in Figure 1 to illustrate this scenario and how problematic missing cycles can be. In this example, the original circuit is an inverter and the encrypted circuit has three key inputs. If the topological order for all the non-input signals in the encrypted circuit is (A, B, Out, C), B and C are feedback wires and will be broken in CycSAT as shown in Figure 2. Then CycSAT will formulate the acyclic condition NC as follows.

$$F(B, B') = F(B, A) \vee F(A, B') = \neg k_2 \vee \neg k_1$$
$$F(C, C') = F(C, B') \vee F(Out, C') = k_1 \vee k_3$$
$$NC = F(B, B') \wedge F(C, C') = (\neg k_2 \vee \neg k_1) \wedge (k_1 \vee k_3)$$

If we set $k_1 = 0$, $k_2 = 1$ and $k_3 = 1$, the above NC constraint is satisfied. However, under this key input assignment, there still exists a structural cycle (C, B, A, C), which contains the two feedback wires B and C.

Can we avoid missing cycles given all the feedbacks? Unfortunately, Chen [4] proved a strong claim as follows:

Theorem 3.1. *The edges in a directed graph cannot always be divided into two disjoint sets, such that any simple cycle is formed by two simple paths, one composed of edges from each set.*

This theorem implies that if more than one backward edges are involved in a cycle, they possibly do not form a simple path on this cycle, and the traversal in any topological order of vertices ignoring these backward edges does miss this cycle. This argument also reveals the unavoidable limitation of CycSAT and tells us it is impossible to efficiently capture the condition to break all possible cycles in the cyclic encryption circuit to prune out all the non-combinational keys. It is necessary to meanwhile examine the behavior of the cyclic circuits in order to solve the correct key as will be presented in Section 4.

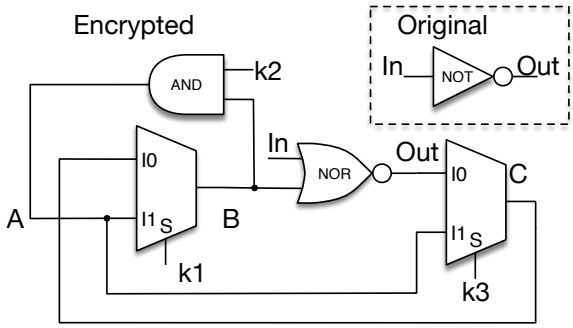

Figure 1: An example showing the limitation of CycSAT.

4. BESAT

The difficulty for CycSAT to capture all cycles in cyclic logic encryption indicates that only the structural analysis may not be enough. Could we propose a behavioral cyclic decryption technique to overcome the drawback of CycSAT? In this section, we first analyze the outcome of missing cycles in CycSAT, then introduce BeSAT, which includes a behavioral SAT-based technique to efficiently decrypt cyclic logic encryption.

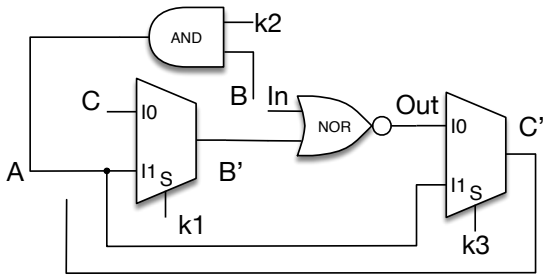

Figure 2: A circuit with the feedback broken.

4.1 Statefulness

Let us reconsider the example in Figure 1. After the structural analysis of CycSAT, a structural cycle (C, B, A, C) still exists. We notice that under the key value $k_1k_2k_3 = 011$ and the input value $In = 0$, Out cannot be uniquely determined. A input of the NOR gate is on the path of the cycle (C, B, A, C), so its value does not depend on In. As a result, the output of the circuit can vary under a fixed input and key pattern. We call this condition as statefulness, and the formal definition of statefulness is given as follows.

Definition 4.1. *Assume $c_1(x, k)$ and $c_2(x, k)$ are two copies of an encrypted circuit. Statefulness indicates $\exists$ an input $\hat{x}$ and a key $\hat{k}$, s.t. $c_1(\hat{x}, \hat{k}) \neq c_2(\hat{x}, \hat{k})$.*

we further investigate how statefulness can affect CycSAT to find the correct key value. Assume after computing the "no structural path" formula, the key assignment $k_1k_2k_3 = 011$ for both copies along with the DIP $In = 0$ will be found in the iteration i. Therefore, the correct output "1" can be evaluated by an activated IC, and the CNF formula $(c(0, k_1) = 1) \wedge (c(0, k_2) = 1)$ is utilized to constrain k_1 and k_2. However, in the iteration $i + 1$, the formula $(c(0, k_1) = 1) \wedge (c(0, k_2) = 1) \wedge c(0, k_1) \wedge c(0, k_2) \wedge (c(0, k_1) \neq c(0, k_2))$ can still be satisfied, for the reason that the formula $(c(0, 011) = 0) \wedge (C(0, 011) = 1)$ can be satisfied by assigning different values to the feedback. Therefore, the SAT solver can find the same assignment of all variables in the following iterations, and CycSAT cannot be terminated.

The above example shows that CycSAT may trap into the infinite loop when statefulness happens. The SAT solver can always find the same DIP, and none of wrong keys can be pruned. Hence, CycSAT performs as bad as the original SAT-based attack on the cyclic circuit if it cannot find all the cycles and create the NC formula accordingly. However, if we are able to detect statefulness, we can explicitly block the wrong key, so the SAT solver can continue to find a new DIP. To overcome this issue, we propose to record the DIP found by the SAT solver in each iteration, and detect if a repeated DIP exists. Hence, the following theorem can be proved.

Theorem 4.1. *If a repeated DIP is found in an iteration of CycSAT, at least one of the keys found by the SAT solver in this iteration causes statefulness.*

PROOF. In CycSAT, if a DIP, $\hat{x}$, is found, the correct output $\hat{y}$ of $\hat{x}$ will be evaluated by an activated IC, and the input-output pair $(\hat{x}, \hat{y})$ will be used to constrain k_1 and k_2 by adding $C(\hat{x}, k_1, \hat{y}) \wedge C(\hat{x}, k_2, \hat{y})$ to the existing CNF. If the same DIP $\hat{x}$ is found again by the SAT solver in the following iterations, it indicates that when the SAT solver tries to find the assignment of the CNF $c(\hat{x}, k_1, \hat{y}) \wedge c(\hat{x}, k_2, \hat{y}) \wedge c(x, k_1, y_1) \wedge c(x, k_2, y_2) \wedge (y_1 \neq y_2)$, at least one of the y_1 and y_2 is assigned a value not equal to $\hat{y}$, and either the CNF formula $c(\hat{x}, k_1, \hat{y}) \wedge c(\hat{x}, k_1, y_1) \wedge (y_1 \neq \hat{y})$ or $c(\hat{x}, k_2, \hat{y}) \wedge c(\hat{x}, k_2, y_2) \wedge (y_2 \neq \hat{y})$ can be satisfied. By the definition, the assignment of k_1 or k_2 is a stateful key since two different outputs can be evaluated under the same input and key pattern. □

Once we detect the stateful condition, we are able to directly prune out the stateful key. To find out whether the assignment of k_1 and k_2 should be pruned or not, we evaluate the correct output under the repeated DIP $\hat{x}$, and see if the assignment of y_1 and y_2 matches with the correct output. The key value that causes different outputs is a stateful key (i.e. $\hat{k}$), so we directly block it by adding the constraint $(k_1 \neq \hat{k}) \wedge (k_2 \neq \hat{k})$ into the SAT formula.

The behavioral analysis can be beneficial for CycSAT to overcome the missing cycle problem. In the example in Figure 1, under the key value $k_1k_2k_3 = 011$ and the input value $In = 0$, the stateful situation happens. Therefore, if the SAT solver detects $In = 0$ again, the stateful key $k_1k_2k_3 = 011$ can be directly pruned. As a consequence, the SAT solver has to find new DIPs.

It should be pointed out that more than one stateful key can be pruned if a repeated DIP is found. It is possible that assignments of both y_1 and y_2 are incorrect, and we can remove two stateful keys in one iteration.

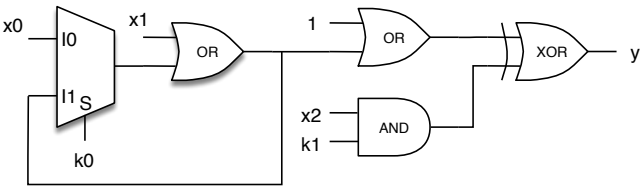

Figure 3: An example showing that the value of the feedback may vary but cannot affect the output.

The Definition 4.1 considers if the outputs vary under a fixed input and key value. It is also possible that with a input and key pattern, the outputs are unique but some internal wires can be assigned to different values. The example in Figure 3 reveals that, even the value of the feedback wire can vary under $k_0 = 1$, it cannot propagate to the output y because "1" is the controlling value of an OR gate. In this paper, we do not count this condition as statefulness.

4.2 Oscillation

However, the undetermined feedback may cause other problems. If we change the AND gate to a NAND gate in Figure 1, with the same assignment $k_1k_2k_3 = 011$ and $In = 0$, the value of the feedback inverts after the computation of the NAND gate, and the feedback connecting to the NOR gate oscillates between "0" and "1". consequently, the output oscillates.

Unfortunately, oscillation may not be that easy to be detected by the SAT engine. A fixed assignment is required for the SAT solver to provide to the CNF formula. However, if the values on some wires keep updating because of oscillation, the SAT solver is not able to find a unique assignment, and unsatisfiable will be returned. Therefore, the key

value causing oscillation meets all the requirements of the CNF constraint, and the oscillatory key can be returned. We define oscillation as follows.

Definition 4.2. *Oscillation indicates* $\exists$ *an input* $\hat{x}$ *and a key* $\hat{k}$*, s.t.* $SAT(c(\hat{x},\hat{k})) = unsatisfiable$*.*

To prevent oscillation, we investigate the ternary simulation combined with the Boolean satisfiability problem. Backes et al. have proposed a ternary-based SAT method [2], and the algorithm is as follows. First, all feedbacks are broken so that the circuit is acyclic, and dummy variables are introduced at each cut location and assigned to $\perp$, which indicates that the value is unknown. Then, the circuit is expressed in terms of the ternary-based logic, and these ternary values are further encoded as dual-rail binary values: zero is encoded to "00", one is encoded to "11", and the unknown is either "01" or "10". If the ternary value of all inputs of a gate is known, we can apply ternary calculus to compute the value of the outputs. For example, for a 2-input AND gate f with inputs a and b, its ternary formula is:

$$f_0 = a_0 b_0 + a_1 b_0 \bar{b_1},$$
$$f_1 = a_1 b_1 + a_0 b_1 \bar{b_0}.$$

In the case, each of the signals has one additional rail. Thus, armed with ternary calculus, we can do the binary computation as usual, and obtain the ternary value for any signal in the circuit.

To make sure the circuit is combinational, two constraints should be satisfied: variables computed at the cut location should agree with the dummy variables, and the dummy variables should not be $\perp$ when the fixed point is reached. To check if these two conditions are satisfied, two types of function blocks have been inserted into the circuit, which are the equivalence checker and the $\perp$ detector. The equivalence checker equals to one if the dummy variables match with the variables at the cut location, and the $\perp$ detector is simply an XOR gate to check if "01" or "10" still exists. The SAT solver is to find if there is an assignment for the following conditions: the equivalence checking is one for all cut locations, and there is at least one pair of dummy variables assigned to "01" or "10". If these conditions can be satisfied, the circuit is considered to be non-combinational.

It is worth noting that the requirement for the circuit to be combinational is arguably stringent, since it is possible that some internal wires in the circuit oscillate without affecting any primary output. However, two counterpoints can be offered. First, such oscillation is a waste of power, and so is unlikely to be present under the entire set of correct keys. Second, Riedel et al. also mention the same question, noting that the ternary SAT algorithm is easily adapted to match such a weaker constraint if so desired [2].

Can the ternary-based SAT method be helpful to detect and prevent oscillation? The following theorem illustrates that the ternary-based SAT method is a strong technique to check if there exists oscillation in the circuit.

Theorem 4.2. *If the ternary-based SAT method returns unsatisfiable on a circuit* $c(x,k)$*,* $\nexists$ *an input* $\hat{x}$ *and a key* $\hat{k}$ *s.t.* $SAT(c(\hat{x},\hat{k})) = unsatisfiable$*.*

PROOF. Since the circuit becomes acyclic after breaking all the feedbacks, all signals in the circuit $c(x,k)$ can only be determined by primary inputs, key inputs and dummy variables. Since the two rails of the primary inputs and key inputs are stick together, if some signals in the circuit are assigned to $\perp$, at least one of the dummy variables must be $\perp$, which can be captured by the $\perp$ detector. In this case, the ternary-based SAT method will return satisfiable. So if the ternary-based SAT method returns unsatisfiable on a circuit $c(x,k)$, it guarantees that the values of every cycle in $c(x,k)$ can be determined only by the input x and key k, which are assigned with Boolean variables. As a result, oscillation will not happen. □

We adopt the ternary-based SAT method and check if the returned key from the SAT solver will cause the circuit non-combinational. Since there is only one unique key that can be embedded in the circuit, we perform the ternary-based SAT method on the circuit $c(x,\hat{k})$ with a fixed key value $\hat{k}$. If the ternary-based SAT method returns satisfiable, the circuit is non-combinational and $\hat{k}$ should be pruned. We repeatedly find the candidate of the correct key, and use the ternary-based SAT method to check oscillation until it returns unsatisfiable.

4.3 BeSAT Algorithm

The algorithm of BeSAT is then can be developed as shown in Algorithm 3. First, we generate the "no structural path" formula, and its CNF formula is added into the existing constraint. Then, we conduct the behavioral analysis. For each iteration, BeSAT checks if the SAT solver finds the repeated DIP, and directly block stateful keys that prevent the algorithm from termination. When the SAT solver cannot find any DIP, we consider all the key values that satisfy the current CNF constraint as candidates of the correct key, and we use the ternary-based SAT method to verify. Once the ternary-based SAT method returns unsatisfiable, the key is returned and considered to be correct.

Theorem 4.3. *When BeSAT terminates and returns a key value, the returned key is guaranteed to be correct.*

PROOF. Since wrong keys that lead to an acyclic circuit can be pruned by simply adding key constraints, it is equivalent to prove that if the stateful and oscillatory keys are missed by the "no structural path" formula, they can still be detected and pruned by BeSAT.

Assume that when the algorithm terminates, there is a missing stateful key. By the definition of the statefulness, the SAT solver can still find a DIP so that k_1 and k_2 are both assigned to the stateful key, and diverse outputs can be produced. Therefore, the algorithm cannot be terminated, which contradicts to our assumption. So when the SAT solver cannot find any DIP, all the stateful keys can be pruned.

The ternary-based SAT method then tries to find a key so that the circuit cannot oscillate for all possible inputs. It adopts a stringent definition that no $\perp$ values can persist in the circuit. As Theorem 4.2 proved, if the ternary-based SAT method returns unsatisfiable, it is guaranteed that there is no oscillation in the circuit under this key. □

5. EXPERIMENTAL RESULTS

In this section, we evaluate the correctness and the efficiency of BeSAT. Our experiment is conducted on a machine with 2.4 GHz Intel Core i5, running Linux with memory 5.4

Table 1: BeSAT, CycSAT and the SAT-based attack on cyclic encryption of original circuits, with 10 intentional feedbacks of cycle lengths 10.

	cyclic encrypted circuits					CycSAT		SAT−based attack		BeSAT	
circuit	#primary inputs	#key inputs	#outputs	#gates	#cycles	key	#iterations	key	#iterations	key	#iterations
apex2	39	20	3	600	20	yes	4	no	–	yes	4
apex4	10	20	19	5380	20	yes	3	no	–	yes	3
c432	36	20	7	180	20	no	–	no	–	yes	17
c499	41	20	32	222	20	yes	3	yes	8	yes	4
c880	60	20	26	403	20	no	–	no	–	yes	19
c1355	41	20	32	566	20	yes	25	no	–	yes	28
c1908	33	20	25	900	20	yes	34	yes	167	yes	8
c2670	233	20	140	140	20	no	–	no	–	yes	32
c3540	50	20	22	1689	20	no	–	no	–	yes	31
c5315	178	20	123	2327	20	no	–	yes	15	yes	14
c7552	207	20	108	3532	20	no	–	no	–	yes	5
dalu	75	20	16	2318	20	yes	16	yes	9	yes	12
des	256	20	245	6493	20	yes	5	no	–	yes	5
ex5	8	20	63	1075	20	yes	11	no	–	yes	11
ex1010	10	20	10	5086	20	yes	3	no	–	yes	3
i4	192	20	6	358	20	yes	3	no	–	yes	3
i7	199	20	67	1335	20	yes	7	yes	6	yes	7
i8	133	20	81	2484	20	yes	8	yes	7	yes	8
i9	88	20	63	1055	20	yes	3	yes	11	yes	4
k2	46	20	45	1835	20	no	–	no	–	yes	37
seq	41	20	35	3559	20	yes	5	no	–	yes	5

GiB. We perform BeSAT on two types of benchmarks: benchmarks encrypted with only the cyclic logic encryption, and benchmarks encrypted with both the traditional logic encryption and the cyclic encryption. The cyclic encrypted circuits are from Zhou et al. [21], and original acyclic benchmarks are from the ISCAS'85 and the Microelectronics Center of North Carolina (MCNC). The method proposed by Shamsi et al. [11] is adopted as the cyclic logic encryption technique, which creates irreducible loops with multiple removable edges in circuits so that the attack difficulty is increased. It can be proved that with M edges and N loops in the cyclic encryption circuit, the attack complexity can be exponential as $2^{M \times N}$ [11]. Extra multiplexers are also inserted into the encrypted circuit at the feedbacks of cycles to ensure the cycles are irreducible.

We use the traditional logic encryption technique proposed by Dupuis et al. [5], which inserts AND and OR gates into the circuit so that the number of low-controllability locations is minimized. The benchmarks in the experiment have 10 percent area overhead of the traditional logic encryption, and we select two advanced logic decryption techniques, the SAT-based attack and CycSAT, as comparison. Although there are many other encryption techniques such as Anti-SAT [18] and SFLL [20], they are not related to cyclic logic encryption, so we do not include them in the experiment. Meanwhile, We did not discuss removal or side channel attacks since they have been shown to be ineffective on cyclic logic encryptions by Shamsi et al. in [12].

We evaluate BeSAT on encrypted benchmarks with 10 intentional feedbacks of cycle lengths 10. The result is illustrated in Table 1. The "key" indicates whether the key values are correct or not, and "-" means the logic decryption techniques do not terminate because of infinite or exponential loops. If the process of the decryption can be complete, the numbers of iterations that each logic decryption technique takes are shown in the table. We notice that the original SAT-based attack has the worst performance among these three techniques, since most of the encrypted benchmarks cannot be decrypted because of the infinite loop. Meanwhile, the CycSAT can solve the correct key for most of benchmarks in a few iterations, but there are still 33% benchmarks cannot be decrypted. However, the BeSAT is able to successfully attack all of benchmarks, and it takes no more than 37 iterations to finish. It should be mentioned that the SAT-based attack may perform better than CycSAT or BeSAT due to the diverse selection of DIPs by a SAT solver. Meanwhile, benchmarks such as C5315 can be defeated by SAT-based attack because the inserted cycles do not produce non-combinational behavior. However, when cycles are carefully inserted and the hard cycle shown in Figure 1 are added, neither CycSAT nor the SAT-based attack can defeat it, only BeSAT has a chance.

To further investigate if the traditional logic encryption increases the attack complexity for BeSAT, we perform BeSAT and CycSAT on benchmarks encrypted with both cyclic and traditional encryption. The result is shown in Figure 4 and 5. We set 13000 iterations as the threshold to illustrate that the decryption cannot be finished within 24 hours. Overall, BeSAT and CycSAT take similar numbers of iterations to complete. However, BeSAT can decrypt more cyclic encrypted benchmarks than CycSAT.

6. CONCLUSION

Recently, cyclic logic encryption [11] is proposed as a newly encryption technique to defeat many existing attacks, since it introduces dummy cycles into the circuit, which can lead to statefulness and oscillation. Even though CycSAT [21] is proposed to carefully compute the formula that there is no structural path under a key, we noticed that it is impossible to capture all cycles in any circuit with any set of feedback signals in such a way. In order to overcome the drawback of CycSAT, we propose BeSAT, which brings in the behavioral analysis to overcome the challenge that CycSAT may miss cycles. The experimental result indicates BeSAT

Algorithm 3 the BeSAT Algorithm

Input: The cyclic encryption circuit $c(x, k)$ and an original Boolean function $f(x)$.
Output: Correct key k^* such that $c(x, k^*) \equiv f(x)$.

Find a set of feedback signals $(w_0, \ldots, w_m)$;
Compute "no structural path" formulas $F(w_0, w_0'), \ldots, F(w_m, w_m')$;
$NC = \bigwedge_{i=0}^{m} F(w_i, w_i')$;
$c(x, k_1) = c(x, k_1) \wedge NC(k_1)$;
$c(x, k_2) = c(x, k_2) \wedge NC(k_2)$;
$DIP = \{\}$;
while $\hat{x}, \hat{k_1}, \hat{k_2} = SAT(c(x, k_1) \neq c(x, k_2))$ **do**
  $\hat{y} = f(\hat{x})$;
  if $\hat{x}$ in DIP **then**
    if $c(\hat{x}, \hat{k_1}) \neq \hat{y}$ **then**
      $c(x, k_1) = c(x, k_1) \wedge (k_1 \neq \hat{k_1})$;
      $c(x, k_2) = c(x, k_2) \wedge (k_2 \neq \hat{k_1})$;
    if $C(\hat{x}, \hat{k_2}) \neq \hat{y}$ **then**
      $c(x, k_1) = c(x, k_1) \wedge (k_1 \neq \hat{k_2})$;
      $c(x, k_2) = c(x, k_2) \wedge (k_2 \neq \hat{k_2})$;
  $DIP.add(\hat{x})$;
  $c(x, k_1) = c(x, k_1) \wedge (c(\hat{x}, k_1) = \hat{y})$;
  $c(x, k_2) = c(x, k_2) \wedge (c(\hat{x}, k_2) = \hat{y})$;
end while
while $\hat{kc} = SAT(c(x, k_1))$ **do**
  if $ternary\_SAT(c(x, k), \hat{kc}) == SAT$ **then**
    $c(x, k_1) = c(x, k_1) \wedge (k_1 \neq \hat{kc})$;
  else
    $k^* = \hat{kc}$, break;
end while

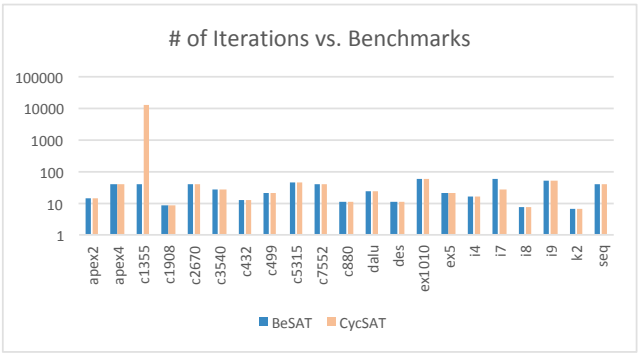

Figure 4: CycSAT vs. BeSAT on benchmarks encrypted with 5 intentional feedbacks of cycle lengths 5.

can efficiently solve the correct key value of cyclic encrypted benchmarks.

Acknowledgments

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

7. REFERENCES

[1] M. Abramovici and P. Bradley. Integrated circuit security: new threats and solutions. In *Workshop on Cyber Security and Information Intelligence Research*, page 55, 2009.
[2] J. Backes, B. Fett, and M. D. Riedel. The analysis of cyclic circuits with boolean satisfiability. In *ICCAD*, pages 143–148, 2008.

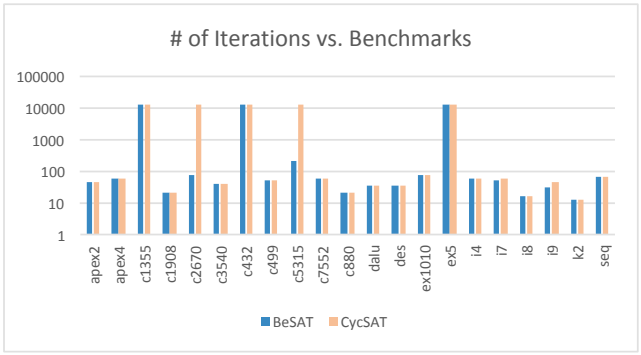

Figure 5: CycSAT vs. BeSAT on benchmarks encrypted with 20 intentional feedbacks of cycle lengths 5.

[3] A. Baumgarten, A. Tyagi, and J. Zambreno. Preventing ic piracy using reconfigurable logic barriers. *IEEE Design and Test*, 27(1), 2010.
[4] R. Chen and H. Zhou. Statistical timing verification for transparently latched circuits. *IEEE TCAD*, 25(9), 2006.
[5] S. Dupuis, P. S. Ba, G. D. Natale, M. L. Flottes, and B. Rouzeyre. A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans. In *IEEE International On-Line Testing Symposium*, pages 49–54, 2014.
[6] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri. Security analysis of logic obfuscation. In *DAC*, pages 83–89, 2012.
[7] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri. Security analysis of integrated circuit camouflaging. In *CCS*, pages 709–720, 2013.
[8] J. Rajendran, H. Zhang, C. Zhang, G. S. Rose, Y. Pino, O. Sinanoglu, and R. Karri. Fault analysis-based logic encryption. *IEEE Transactions on Computers*, 64(2), 2015.
[9] A. Rezaei, Y. Shen, S. Kong, J. Gu, and H. Zhou. Cyclic locking and memristor-based obfuscation against cycsat and inside foundry attacks. In *DATE*, pages 85–90, 2018.
[10] J. A. Roy, F. Koushanfar, and I. L. Markov. EPIC: Ending piracy of integrated circuits. In *DATE*, pages 1069–1074, 2008.
[11] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Pan, and Y. Jin. Cyclic obfuscation for creating SAT-unresolvable circuits. In *GLSVLSI*, pages 173–178, 2017.
[12] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin. AppSAT: Approximately deobfuscating integrated circuits. In *HOST*, pages 95–100, 2017.
[13] Y. Shen, A. Rezaei, and H. Zhou. A comparative investigation of approximate attacks on logic encryptions. In *ASP-DAC*, pages 271–276, 2018.
[14] Y. Shen, A. Rezaei, and H. Zhou. Sat-based bit-flipping attack on logic encryptions. In *DATE*, pages 629–632, 2018.
[15] Y. Shen and H. Zhou. Double dip: Re-evaluating security of logic encryption algorithms. In *GLSVLSI*, pages 179–184, 2017.
[16] P. Subramanyan, S. Ray, and S. Malik. Evaluating the security of logic encryption algorithms. In *HOST*, pages 137–143, 2015.
[17] R. Torrance and D. James. The state-of-the-art in semiconductor reverse engineering. In *DAC*, pages 333–338, 2011.
[18] Y. Xie and A. Srivastava. Mitigating SAT attack on logic locking. In *CHES*, pages 127–146, 2016.
[19] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu. SARLock: SAT attack resistant logic locking. In *HOST*, pages 236–241, 2016.
[20] M. Yasin, A. Sengupta, M. Nabeel, M. Ashraf, J. J. Rajendran, and O. Sinanoglu. Provably-secure logic locking: From theory to practice. In *CCS*, pages 1601–1618, 2017.
[21] H. Zhou, R. Jiang, and S. Kong. Cycsat: Sat-based attack on cyclic logic encryptions. In *ICCAD*, pages 49–56, 2017.