A Humble Theory and Application for Logic Encryption

Hai Zhou
Northwestern University

ABSTRACT

Logic encryption is an important hardware security technique that introduces keys to modify a given combinational circuit in order to lock the functionality from unauthorized uses. Traditional methods are all ad hoc approaches based on inserting lock gates with keys on randomly selected signals in the original circuit. Thus, it was not a surprise that a SAT-based attack developed by Subramanyan et al. successfully defeated almost all of them. New approaches such as SARLock and Anti-SAT were then developed to defend against SAT-based attack. However, they are still vulnerable with extremely low error rates.

In this paper, we present a theory on logic encryption that provides a complete understanding on the design space and the trade-off between error rate and attack complexity. An efficient general design scheme is derived from the theory and some specific designs are also suggested. We also suggest a method to insert one-way function to burden the SAT engine, and a method to obfuscate the whole design. In addition, as an application of the theory, we also develop a scientific encryption benchmark for approximate attacks. We test our encryption designs and obfuscation techniques by the SAT-based attack, and the results have verified the robustness of our approach.

CCS CONCEPTS

• **Security and privacy → Hardware reverse engineering**; Tamper-proof and tamper-resistant designs; • **Hardware → Combinational circuits**;

KEYWORDS

logic encryption, obfuscation, SAT attack, theory

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 [13, 16].

Even though there exist many different approaches for logic encryption [3, 9, 15, 17, 18], 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. [21] can efficiently decrypt almost all of the encrypted circuits by them.

Immediately after Subramanyan et al. [21], some remedies were proposed to strengthen the existing logic encryption. Yasin et al. [25] proposed SARLock, which was inspired by the difficult case of the AND-tree discovered in [21], and ensures that each wrong key can only be excluded by one input. Xie and Srivastava [24] developed the Anti-SAT encryption, where one key has at most one wrong input, but one input may exclude many wrong keys. 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 [20] and AppSAT [19] are the first approaches for approximate attacks to logic encryption.

In this paper, we would like to change the status quo of logic encryption by developing a theory for it. We start with a basic understanding that there are two entangled goals in logic encryption: locking and obfuscation. *Locking is a logical request to make sure that the correct behavior only happens when a correct key is applied and the correct key cannot be easily figured out by studying the* logic *of the encryption circuit. Obfuscation is a structural request to make sure that the correct circuit cannot be revealed by any* structural *analysis of the encryption circuit.*

We suspect that the entangled goals of locking and obfuscation might be one cause of current ad hoc approaches in

Conference'17, July 2017, Washington, DC, USA
2017. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

logic encryption. Therefore, we want to separate the two concerns starting from the beginning. By investigating the logic of all possible encryptions for a given function, we develop a theory for logic encryption that captures the whole design space and the relation between a design and its attack complexity. With error rate as a design goal, we also establish the relation between error rate and attack complexity. Both minimal and average attack complexities are considered in our work, and what we discover is that there is a contention between attack complexity and error numbers and their product cannot be larger than $n2^n$, even for average attack complexity.

Our theory has been initially investigated without any concern on the size of the encryption circuits. Fortunately, when the size constraint is considered, we find that the above bound can still be realized by *linear-sized* encryption circuit. The benefit comes from the modulating of the keys by the normal inputs with XOR gates. Based on this, we propose a general logic encryption scheme.

In order to further thwart SAT-based logic analysis attacks, we also propose to insert Goldreich's one-way function [11] in the scheme to increase the running time for SAT engines.

We should caution the readers not to worry about the easiness of structural attack to our proposed logic encryption scheme, since it represents only the logic functionality of any equivalent circuits. Circuit obfuscation will always be applied to these logic to hide any structural vulnerability.

Circuit (or program) obfuscation has been a practice in software engineering for a long time. However, its theoretical study only started relatively recently [2]. Even with a recent breakthrough on indistinguishability obfuscation [10], it is still too expensive to be deployed in our logic encryption. We will show that resynthesis is complete for *the best-possible obfuscation* [12], and propose to do resynthesis on our logic encryption to protect it from structural analysis attacks.

It is very difficult to measure the performance of an approximate attack, since the error rate cannot be measured on the number of key bits that the same as the correct one, and exactly measuring of error rate requests to simulate all input combinations. As an application of our theory, we develop a suite of scientific benchmarks for approximate attacks. On these benchmarks, different key has different error rate, which can be simply calculated based on the key.

We also test our logic encryption designs and their obfuscations with Goldreich's one-way function by the SAT-based attack [21]. The results confirms our theory and the robustness of our approach.

2 BACKGROUND AND PROBLEM FORMULATION

2.1 Conventional Logic Encryption and SAT-Based Attack

There exist many different ways [3, 9, 15, 17, 18] for logic encryption. However, they are all 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. [21] 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

Require: An encryption circuit $g(x, k)$ and original boolean function $f(x)$.
Ensure: Correct key k^* such that $g(x, k^*) \equiv f(x)$.

```
while x̂ = SAT(g(x,k) ≠ g(x,k1)) do
    ŷ = f(x̂);
    g(x,k) = g(x,k) ∧ (g(x̂,k) = ŷ);
    g(x,k1) = g(x,k1) ∧ (g(x̂,k1) = ŷ);
end while
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. [25] and Xie and Srivastava [24] explored this property to develop strengthening approaches.

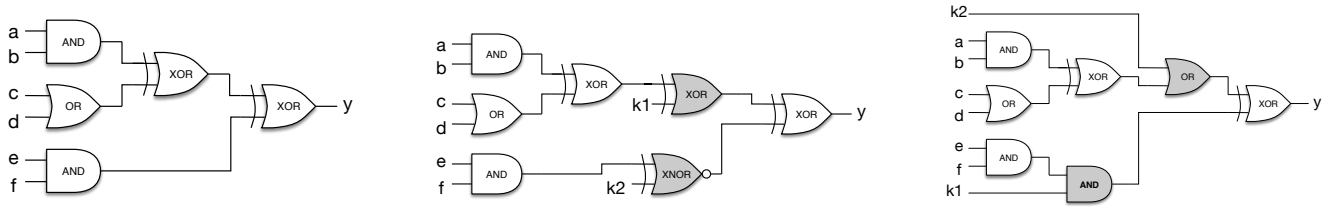

Figure 1: Traditional logic encryption example: gray gates are lock gates.

But since they both have an error rate of 2^{-n}, they can be effectively defeated by approximate attacks such as Double DIP [20] and AppSAT [19].

2.2 Problem Definition

The logic encryption problem can be formulated as follows. Given a Boolean function $f : B^n \rightarrow B$ represented by a multi-level netlist of Boolean gates, find a Boolean function $g : B^{2n} \rightarrow B$ also as a multi-level netlist, such that

(1) There is a random Boolean n-vector k^* such that $g(x, k^*) \equiv f(x)$;
(2) The netlist size of g is only a polynomial of that of f;
(3) With the netlist of g and a black-box access to f, it is difficult to "discover" f.

The first two requirements are straightforward: the first one means there is at least one correct key to decrypt the logic; the second one only allows a polynomial blow-up of the circuit size. For performance critical applications, we may also request that the netlist depth of g be bounded. However, the last requirement is tricky and requests a thorough discussion.

There are two different aspects that need to be discussed in the third requirement: what it means to "discover" f, and how to measure the difficulty. The conventional definition for "discovering" f is to find k^* or any k such that $g(x, k) \equiv h(x)$. The interesting result of Subramanyan and Malik [21] was to find that almost all existing logic encryption algorithms before them can be effectively decrypted by a SAT-based attack. The effectiveness is measured by the number of inputs that are queried on the black-box of f, which is bounded by a polynomial of n. In addition, it also depends on the efficiency of solving a SAT problem on all these inputs for a key.

Immediately after the SAT-based attack by Subramanyan and Malik [21], there come a few remedies for it. SARLock [25] utilizes the AND-tree structure discovered in Subramanyan and Malik [21] to thwart the SAT-based attack by ensuring that the number of inputs to be queried is at least exponential. Anti-SAT [24] is another systematic way to design logic encryptions that will also request the SAT-based attack to query exponential number of inputs. One drawback for both SARLock and Anti-SAT approaches is that when applying an arbitrary key k, the error rate is extremely low. Here the error rate is defined as the percentage of inputs x that will generate wrong output, meaning, $g(x, k) \neq f(x)$. *For both SARLock and Anti-SAT, there is exact one input with wrong output for every $k \neq k^*$.*

The extreme low error rates in SARLock and Anti-SAT entice a new type of attacks: approximation attacks. A random guess k on the key will give a circuit $g(x, k)$ that is "almost correct" where a wrong result will happen on at most one input. The justification for such approximation attacks is that the attacker is very sure that the unauthorized circuit can be sold on the market with an extremely low probability of being caught. Furthermore, any catch of a discrepancy on a given input will help the attacker to correct the key.

Another argument for approximate attacks–assuming we are in the shoes of an attacker–is that, under a wrong key with extremely low error rate, the attacker now not only pirates on your intellectual property but also places in it a perfect Trojan horse. Remember that an extremely low error rate means that it will be extremely difficult to detect the Trojan. In this sense, we may argue that approximate attacks are even more malicious than the exact attacks, thus should be seriously prevented in logic encryption.

Obviously, Yasin et al. [25] and Xie and Srivastava [24], the designers of SARLock and Anti-SAT respectively, were already aware of the approximate attacks. Therefore, they both proposed in their work to combined the new encryption approaches with existing approaches, just in order to thwart the approximation attacks [19, 20].

With this perspective, we believe that it is necessary to include approximate attack as successfully "discovering" f in our problem formulation. When we argue that a logic encryption is difficult to decrypt, it is not sufficient to consider only existing attack methods such as SAT-based attack and 2DIP attack, since who knows what new attack method will be invented tomorrow. Following the tradition-honored practice in cryptography, we would better consider probabilistic learning as the most general form for "discovering" f. Therefore, we should consider decryption as finding a key k such that $g(x, k)$ is Probabilistically Approximately Correct (PAC) [22] for $f(x)$.

3 A THEORY OF LOGIC ENCRYPTION

In this section, we will consider a general theory for logic encryption, where the efficiency will be temporally ignored. In other word, we will consider neither the netlist size constraint in the second requirement in the problem formulation nor the runtime for constructing the encryption.

Since each iteration of the SAT-based attack is to find an input that can differentiate two keys on $g(x,k)$ and then to constrain the keys by the evaluation of that input on the original circuit $f(x)$, we start our investigation by a Shannon decomposition of the encryption circuit $g(x,k)$ on the input x. It gives us the following equation:

$$g(x,k) = \bigvee_{i=0}^{2^n-1} m_i(x) \wedge g(i,k),$$

where $m_i(x)$ is the ith minterm of x, and $g(i,k)$ is Boolean function of only k.

Under the requirement of $g(x,k^*) = f(x)$, we must have $g(i,k^*) = f(i)$ for all $i \in 0..2^n-1$. In terms of minterms of k, it means that $m_{k^*}(k) \Rightarrow (g(i,k) = f(i))$ for all $i \in 0..2^n-1$. In the same vein, if we want a wrong key w to generate a wrong output on the input i, we must have $m_w(k) \Rightarrow (g(i,k) \neq f(i))$. In other words, if we want to have a mismatch on input i for key j, we include the minterm $m_j(k)$ in $g(i,k)$ if $f(i) = 0$. We will do the opposite if $f(i) = 1$.

We define the error number for a given key j as the number of mismatches on all inputs. It can be easily designed by including (when $f(i) = 0$)) or excluding (when $f(i) = 1$) $m_j(k)$ from $g(i,k)$ on any desired number of inputs. For example, if we want the error number to be 2^n for any wrong key, the logic encryption must be

$$g(x,k) = f(x)m_{k^*}(k) \vee \neg f(x)\neg m_{k^*}(k).$$

However, such a design can be easily decrypted by the SAT-based attack: one iteration will find the key k^*.

Besides error number, the other important aspect of logic encryption design is the difficulty for attacks. We will first consider exact attacks, especially the SAT-based attack. In the above framework, a query on input i will add the contraint $g(i,k) = f(i)$ to the key constraint. If the conjunction of existing contraints implies $g(j,k) = f(j)$ for any input j not queried yet, it makes the query unnecessary. Therefore, the number of queries needed for exact attack is dependent on the logic relations among $g(i,k) = f(i)$ for all i and also the order on which each input i is queried. Since any SAT engine has a built-in randomness and we are generally blind to it, we have to use the average number of queries as the measure, which may be complicated to calculate.

We first note that each function $g(i,k) = f(i)$ could be an arbitrary function with n inputs and one output. We denote the set of all such functions as $[B^n \to B]$. The total number of functions in this set is 2^{2^n}. In order to study their logic relations, we will look at the complete lattice given by $([B^n \to B], \vee, \wedge)$. For uniformity, we will view each function as a subset of minterms of k. As an illustration, the complete lattice of all possible functions $[B^2 \to B]$ is shown in Figure 2. The top element is $TRUE$ and the bottom element $FALSE$. The row above $FALSE$ has all the minterms, and the row below $TRUE$ has the negations of all the minterms. The partial order on the lattice is the implication relation $\Rightarrow$.

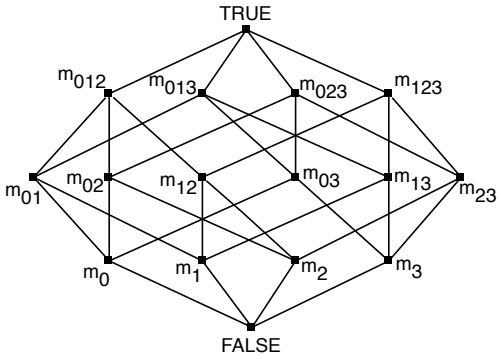

Figure 2: The complete lattice of all possible 2-input 1-output boolean functions, edges representing implications.

Now we will map the 2^n number of functions $g(i,k) = f(i)$ to the complete lattice. Some of these functions may collide with each other, thus the total number of elements mapped on the lattice may be smaller. For example, in the above encryption with 2^n as error number, there is only one mapped element $m_{k^*}(k)$, which explains why decrypting it requests only one query. As another example, consider the AND-tree discovered by Subramanyan et al. [21]:

$$g(x,k) = \bigwedge_{i=0}^{n-1} x_i \oplus k_i, \quad f(x) = \bigwedge_{i=0}^{n-1} x_i.$$

The Shannon decomposition is

$$g(x,k) = \bigvee_{i=1}^{2^n-1} m_i(x) m_{\bar{i}}(k),$$

where $\bar{i}$ is one's complement of i. The functions $m_{\bar{i}}(k) = f(i)$ map to a minterm $m_0(k)$, and negations $\neg m_i(k)$ for all other $i \neq 0 \in 0..2^n-1$. These negations are each independent of the other, but they are all implied by $m_0(k)$. The minimal number of queries for decryption is one, if the first input queried happens to be $i = 2^n - 1$. The maximal number of queries is 2^n-1, while any number in between is possible. Of course, a careful analysis can show that the average number of queries needed is still exponential, which explains why its decryption is expensive.

The general case will have 2^n mapped functions in the lattice and there may exist complicated implication relations among different conjunctions of them. We have the following lemma.

Lemma 3.1. *Given a logic encryption $g(x,k)$ for a function $f(x)$, an DIP set D can exactly decrypt $g(x,k)$ if and only if the infimum of $\{g(i,k) = f(i) | i \in D\}$ is also the infimum of the functions $g(i,k) = f(i)$ for all $i \in 0..2^n - 1$ in the complete lattice $([B^n \to B], \vee, \wedge)$, in other words*

$$\wedge_{i \in D} g(i,k) = f(i) \Rightarrow \wedge_{i \in 0..2^n} g(i,k) = f(i).$$

If we have a subset of functions $g(i,k) = f(i)$ formed a chain in the complete lattice, then the bottom element will imply all other elements. It means that the minimal number of queries needed for them is just one. However, the average number of inputs to query before finding the bottom one is $\Theta(\log m)$, where m is the number of elements in the chain.

On the other hand, if we have a subset of m functions that none of them implies the other, what is the average number of queries before we find the infimum? This is complicated. In general, it depends on the difference between the common minterms of the whole set and those of an arbitrary subset. Any function that excludes at least one minterm that is common in all other functions must be queried. We can use this property to make sure that m is the minimum number of queries needed. For example, this is the case for the $2^n - 1$ negations $\neg m_i(k)$ in the AND-tree.

Besides the attack complexity, another design criteria of encryption circuit is the error rate. For any wrong key, the error rate is the ratio of inputs generating wrong outputs. In other words, the error rate is the error number divided by the total number of possible inputs (2^n). All existing remedies [24, 25] against the SAT-based attack have extremely low error rate: 2^{-n}. Therefore, an interesting problem to investigate is whether there could be logic encryptions that have both exponential SAT attack complexity and substantial error rates.

In our system, the error number for k is given by

$$error(k) \triangleq \sum_{i=0}^{2^n-1} g(i,k) \neq f(i).$$

We can also symmetrically define the error number for each input i as

$$error(i) \triangleq \sum_{k=0}^{2^n-1} g(i,k) \neq f(i),$$

which can be measured as the number of minterms of k not included in function $g(i,k) = f(i)$. Even though requesting $error(k)$ to be exponential for every wrong k is different from requesting $error(i)$ to be exponential for every input i. It can be seen that they are closely related since the sum is the same, that is,

$$\sum_{i=0}^{2^n-1} error(i) = \sum_{k=0}^{2^n-1} error(k).$$

More importantly, we can also view the logic encryption design as to determine the following error matrix.

$(g(x,k) \neq f(x)) =$

$$\begin{array}{c} \left[\begin{array}{ccccccc} 1 & \dots & 0 & \dots & 0 & \dots & 1 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \dots & g(i,j) \neq f(i) & \dots & 0 & \dots & 1 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & \dots & 1 & \dots & 0 & \dots & 0 \end{array}\right] \begin{array}{c} \text{x=0} \\ \vdots \\ \text{i} \\ \vdots \\ 2^n-1 \end{array} \\ \begin{array}{ccccccc} \text{k=0} & \dots & \text{j} & \dots & \text{k}^* & \dots & 2^n-1 \end{array} \end{array}$$

Each row represents a given x value from 0 to $2^n - 1$, and each column represents a given k value also from 0 to $2^n - 1$. The element at (i,j) represents the value of $g(i,j) \neq f(i)$, that is, an error at $g(i,j)$. The error number for k is the number of ones in column k. The minimal error number is the minimal number of ones in any column with one. On this error matrix, the decryption problem can be formulated as a cover problem: a subset of rows (inputs x) are sufficient if and only if they cover all the columns with ones.

Such a unate covering problem is well studied in logic synthesis [8]. However, our goal is quite different. We want to design a matrix such that the minimal number of ones for any column with one is large, while the minimal (or average) number of rows needed to cover them is also large. However, the following lemma captures a natural contention between these two goals. It also explains why all the existing SAT-resilient approaches have extremely small error rates.

Lemma 3.2. *In any given encryption $g(x,k)$ for any function $f(x)$, if the minimal error number for any wrong key is M, the minimal attack complexity is N, then $MN \leq 2^n$.*

Proof. Assume that we have a minimal cover of size N. Each row in the cover must have at least one column that is uniquely covered by this row. Otherwise, this row can be removed without affecting the coverness, contradicting with the assumption that the cover is minimal. On this unique column, there must be at least M ones. Furthermore, no pair of the ones on the N unique columns can be in the same row. Otherwise, we can add the row with the pair and delete two corresponding rows from the cover, contradicting with the assumption of minimal cover. We have at least N unique columns, each has at least M ones, which cannot share any common rows. The total number of rows is 2^n. Therefore, we have $MN \leq 2^n$. □

Now let us investigate whether we can escape from this contention if the average attack complexity is considered instead of the minimal attack complexity. Unfortunately, we cannot do much, because of the following lemma.

Lemma 3.3. *In any given encryption $g(x,k)$ for any function $f(x)$, if the minimal error number for any wrong key is M, then $n2^n/M$ random DIP queries will decrypt it with a probability at least $1-(2/e)^n$.*

Proof. We will consider a sequence of N independent random selection of rows and to calculate the probability that they are still not a cover. Please note that a DIP selection in SAT-based attack is dependent on existing selections and also no repeated selection is allowed. Therefore, such a probability for independent random selection is an upper bound of the probability for dependent DIP selections.

Now consider each column with one in it. It must have at least M ones. Therefore, an independent random selection of a row will not cover it with a probability at most $1-M/2^n$. A sequence of N selections will not cover it with a probability at most $(1-M/2^n)^N$. There are at most 2^n-1 such columns, thus, such selections will not form a cover with a probability at most $2^n(1-M/2^n)^N$. It can be shown that $(1-M/2^n)^{2^n/M}$ is monotonically increasing and converges to e^{-1}. Therefore, if $N = n2^n/M$, then

$$2^n(1-M/2^n)^N \le 2^n e^{-n} = (2/e)^n.$$

□

Fortunately, we can show that there does exist an encryption with both high attack complexity and high error numbers.

Theorem 3.4. *For any given $f(x)$, there exists a logic encryption $g(x,k)$ such that both the minimal attack complexity and the minimal error number are $2^{n/2}$.*

Proof. This is a constructional proof. Given any $f(x)$, we will construct $g(x,k)$ as follows:

$$g(i,k) = (m_{k*} \vee \bigvee_{j=i}^{i+N} m_j \mod 2^n) \equiv f(i),$$

where $N = 2^n - 2^{n/2}$ It is easy to check that $error(k) = 2^{n/2}$ for any wrong k. We can also show that the minimum number of queries to attack $g(x,k)$ is $2^{n/2}$. This is because that every query can remove at most $2^{n/2}$ minterms, and to remove 2^n-1 minterms needs at least $2^{n/2}$ queries. □

The above theorem is not very practical since the provided $g(x,k)$ may have exponential size. The more important problem is to find such an encryption with small size. Based on our theory, each function $g(i,k) = f(i)$ has to distinguish (thus to exclude) $2^{n/2}$ minterms. The request of small size on g forbids a different block for different $g(i,k) = f(i)$, otherwise there will be exponential number of blocks. Without loss of generality, let $g(0,k) = (m_{k*} + h(k)) \equiv f(0)$ be the block shared by every $g(i,k)$ for $i \in 0..2^n-1$. A good (or perhaps the best) way to get distinguished minterms from each $g(i,k)$ is to modulate k in $h(k)$ bit-wisely by i, that is, to make

$$g(i,k) = (m_{k^*} \vee h(k \oplus i)) \equiv f(i).$$

In this case, we can have a general design as shown in Figure 3, which is also given by the following formula,

$$g(x,k) = (m_{k^*}(k) \vee h(k \oplus x)) \equiv f(x).$$

Its correctness is stated in the following theorem.

Theorem 3.5. *If function $h : B^n \to B$ has an on-set of size M, then the logic encryption given in Figure 3 will have an error number of M for every wrong key, and a minimal attack complexity of $2^n/M$.*

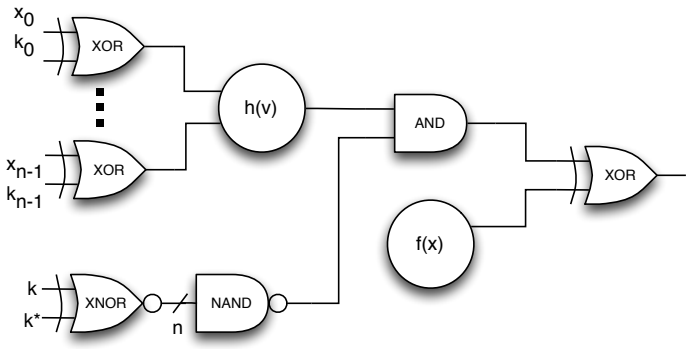

Figure 3: A general logic encryption scheme where h(v) has an on-set of size M.

A simple design following the general scheme is to have $h(v) = \bigwedge_{i=0}^{n/2-1} v_{2i} \oplus v_{2i+1}$, as shown in Figure 4. It can be seen that the on-set of the h function in this design is $2^{n/2}$. Therefore, it has $2^{n/2}$ as both the minimal attack complexity and the error number for every wrong key.

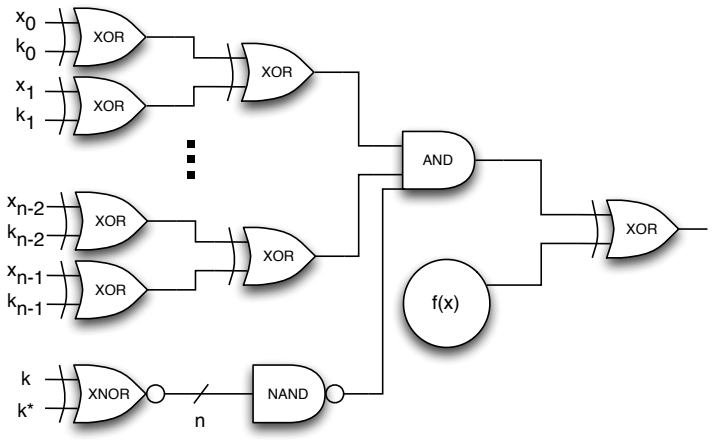

Figure 4: A simple logic encryption design with both exponential attack complexity and error number.

4 INSERTION OF ONE-WAY FUNCTION

Increasing the number of necessary iterations in the SAT-based attack is just one way to increase the attack complexity. Another way is to increase the complexity of SAT instances in the DIP finding. For this purpose, we need to look for hard instances for SAT problem and integrate them into the logic encryption circuit. Cryptography is one of the promising areas to look for hard SAT instances.

Similar ideas have been proposed and discussed by Yasin et al. [26] and cited by Xie and Srivastava [24]. However, they only proposed to use AES as the hard instance. It is well known that AES is a complicated algorithm, with at least 10 cycles of repetition even for the smallest 128-bit key configuration. To be used in the encryption circuit, which is a combinational circuit, the AES has to be unrolled to make it combinational, which will definitely increase its size. The result in [26] only showed the execution time of the attack, but not the circuit size of inserted AES. But it can be estimated that AES will introduce substantial overhead on the circuit size.

We advocate here to use Goldreich's candidate one-way functions based on expander graphs [11] as the hard instances inserted in logic encryptions. The benefits include:

- The Goldreich one-way functions are naturally combinational thus no unrolling is needed;
- They are simple to implement and have only linear sizes in terms of the input size;
- Their one-wayness has been thoroughly investigated and experimentally confirmed with SAT engines [7].

Goldreich's one-way functions are easy to construct. There are two parameters to select: a connection degree d and a predicate P on d inputs. For any n-bit inputs, the one way function will compute each bit of its output by applying P on a random selection of d input bits. There are some criteria to follow: P should not be linear sum or degenerate on the inputs; if the connection between inputs and outputs is treated as a bipartite graph, it has to be an expander. The connection degree d can be very small, in $O(\log n)$ or even $O(1)$.

Cook et al. [7] had a thorough study on Goldreich's one-way functions. Based on previous study, they even suggested a simple predicate

$$P(x_0, \ldots, x_{d-1}) = x_0 \oplus x_1 \ldots \oplus (x_{d-2} \wedge x_{d-1}).$$

They have conducted experiments with SAT engines on functions thus constructed. Even with $d = 5$, they observed an exponential increase of running time as a function of the input length n. Their experiments also indicate that the MiniSat engine will take more than 10 seconds if the input length is 140. That provides a strong evidence for us to suggest such functions to be inserted in logic encryption.

The next question then is where and how to insert Goldreich's functions. There are two possible locations for inserting the function in our general logic encryption scheme shown in Figure 3. One is before the h function, and the other is before the key input to the circuit.

The advantage of the first location is that the function is mixed up well with the rest of the encryption circuit. Its drawback is that, combined with this random function, it is hard to calculate the error rate and attack complexity of the encryption. However, if we assume that these random one-way functions have low collision rate (a collision means two different input mapped to the same output), the disturbance to the error rate and attack complexity will be tiny.

On the other hand, even though the second location (before the key input) will not affect the h function or its on-set, it only give us the benefit of accurately calculating the attack complexity. The error rate with respect to the output of the one-way function is known, but it is still unknown with respect to the input of the one-way function. The drawback of the second location is its not mixing up with other part of encryption circuit. Even though we will definitely do obfuscation by resynthesis of the whole encryption circuit, as discussed in the next section, thus can mix the one-way function with other part of the encryption, its benefit may not justify its adaptation.

Therefore, we suggest to use the general logic encryption with one-way function as shown in Figure 5. Here, the G function is Goldreich's one-way function. The simplest design could use $d = 5$ with

$$P(x_0, \ldots, x_4) = x_0 \oplus x_1 \oplus x_2 \oplus (x_3 \wedge x_4).$$

Please note that we do not need to have a big one-way function to make the SAT engine infeasible to solve one iteration in the attack. Remember that with one more iteration in the SAT-based attack, one instance of the encryption circuit will be added with one DIP into the CNF. Therefore, even though one instance of the one-way function may not be too difficult to solve, its effect will accumulate with each more iteration, and make it getting slower and slower. Remember that our design requests at least exponential number of iterations to decrypt.

5 CIRCUIT OBFUSCATION BY RESYNTHESIS

As we already mentioned in Section 1, there are two tangled goals in logic encryption: locking and obfuscation. Locking is a logical request to make sure that the *almost correct* behavior cannot happen when a wrong key is applied and an almost correct circuit cannot be easily figured out by studying the *logic* of the encryption circuit. Obfuscation is a structural request to make sure that the correct circuit cannot be revealed by any *structural* analysis of the encryption circuit.

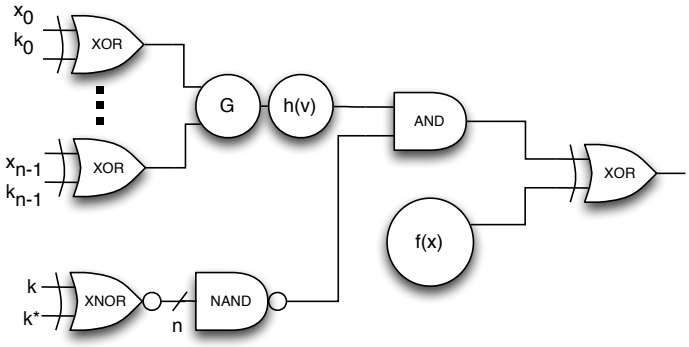

Figure 5: A general logic encryption with one-way function, where G is Goldreich's one-way function, and h(v) has an on-set of size M.

The theory we developed in Sections 3 and 4 achieves the goal of locking without being bothered by obfuscation. In other words, the encryption circuit we designed in these sections are *just an equivalent function of the obfuscated circuit that will be ultimately used.* Obviously, the design in Figure 5 has many vulnerabilities in terms of structural analysis attacks, including subcircuit identification and removal, or general learning. For example, the secret constant k^* can be easily discovered in the circuit structure. Even this is hidden, identifying $f(x)$ XORing other part of the circuit also makes it vulnerable. Therefore, circuit obfuscation is necessary to hide the secret k^* and vulnerable structure of the logic encryption.

Program obfuscation as a practice has long been applied in software engineering to protect the Intellectual Properties of software. However, the theoretical study of program/circuit obfuscation has only been investigated recently, starting with Barak et al. [2].

Barak et al. [2] started with an ambitious definition of obfuscation, that is, a "black-box" obfuscation is one that only leaks the same information as black-box invocations of the program. They proved that such a black-box obfuscation does NOT exist on general programs or circuits. Obviously, a black-box obfuscation protects both the structure and logic of a circuit. Therefore, their result actually shows that *it is impossible to achieve both logical locking and structural obscurity in obfuscation.* This confirms our choice to separate logical locking from structural obfuscation in our theory for logic encryption.

Because of the impossibility result of the black-box obfuscation, Barak et al. [2] have proposed a weaker form of obfuscation, *indistinguishability obfuscation.* An indistinguishability obfuscation of a circuit is such that it is computationally indistinguishable from the obfuscation of any other equivalent circuit. If an indistinguishability obfuscation is applied to our logic encryption in Sections 3 and 4, then it is guaranteed that the obfuscated circuit is the same secure as the obfuscated circuit started with any equivalent circuit.

Actually, we can prove that any logic encryption is logically almost equivalent to the general scheme in Figure 3. The more precise statement is given in the following lemma. Even though we cannot prove that every encryption should have the XOR of x and k, as discussed in Section 3, it may be the best choice for achieving the linear-size encryption.

LEMMA 5.1. *Any logic encryption $g(x, k)$ for any function $f(x)$ is logically equivalent to the circuit shown in Figure 6.*

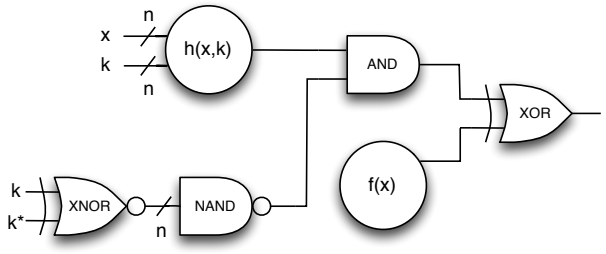

Figure 6: Every possible logic encryption is equivalent to this circuit.

PROOF. Based on the theory in Section 3, the proof is simple. For any logic encryption $g(x, k)$, we can do the Shannon decomposition on x. We then study the structure of $g(x, k)$ based on whether $g(x, k) \neq f(x)$, which gives us the XORing of $f(x)$ with the remaining of the circuit. Since $g(x, k^*) = f(x)$, we have to make the subcircuit XORing with $f(x)$ being zero if $k = k^*$, which gives the ANDing of the $k \neq k^*$ part. Now, the remaining part is identified as $h(x, k)$ shown in the figure. □

With the above lemma, it seems that if an indistinguishability obfuscation is applied on the general logic encryption circuit shown in Figure 3 or Figure 5, we can safely go back to sleep. However, there are a few caveats.

The first is about the obscurity capability of indistinguishability obfuscation. The definition of indistinguishability obfuscation does not specify what information can be hidden by the obfuscation. It only guarantees that it is indistinguishable from obfuscation of any equivalent circuit. Therefore, if every possible equivalent circuit is vulnerable, then even an indistinguishability obfuscation cannot make a circuit secure. However, based on the above lemma, if it is the case, then it also means that logic encryption is impossible, and nobody can help.

The second is about the feasibility of indistinguishability obfuscation. Even though Barak et al. [2] gave the definition and showed some difference between indistinguishability

obfuscation and black-box one, it was not known whether indistinguishability obfuscation is feasible for general program/circuit. This status quo has been only changed very recently in 2013 by Garg et al. [10]. As a big breakthrough in cryptography, they discovered a candidate distinguishability obfuscation for all circuits!

Even though the candidate has provable promising cryptographic properties, its construction is currently still very expensive. Apon et al. [1] had investigated the implementation of the candidate, and found that the obfuscation was "still far from being deployable, with the most complex functionality we are able to obfuscate being a 16-bit point function." Given that their implementation was a program while logic encryption requires a combinational circuit, what we can expect is that the indistinguishability obfuscation is even more expensive to be deployed for logic encryption. Of course, it is a promising research direction to find an efficient trade-off between implementation complexity and the cryptographic guarantees.

Shortly after Barak et al. [2], Goldwasser and Rothblum [12] had proposed an alternative definition of obfuscation: *best-possible obfuscation.* Intuitively, a best-possible obfuscation of a circuit can be as leaky as any circuit that is equivalent to the given circuit. They also proved that any best-possible obfuscater is also an indistinguishability obfuscater, and for efficient obfuscation, they are equivalent. For logic encryption, it is required that the obfuscation is still an efficient combinational circuit. Thus, an indistinguishability obfuscater is also a best-possible obfuscater.

Zhou [27] has developed a sequential circuit locking and obfuscation technique based on the best-possible obfuscation. He applied structural transformation including retiming, resynthesis, sweep, and conditional stuttering to lock and then obfuscate a given sequential circuit. He also proved that those operations are complete for best-possible obfuscation.

In logic encryption, the obfuscation starts with a combinational circuit and ends up with another combinational circuit. Therefore, we can adopt part of Zhou's approach in the following lemma.

Lemma 5.2. *For (combinational) logic encryption, a best-possible obfuscation can be done by a sequence of resynthesis.*

Of course, the lemma only provides a feasibility guarantee of obfuscation by resynthesis. Specific transformations are needed in order to hide sensitive information and vulnerable structures.

One of the vulnerabilities in our general encryption scheme in Figure 3 is the key checking $k = k^*$ and then disabling the flip of $f(x)$. It has the same functionality and structure as the "login" process of many software systems. Embedded in this structure is a point-function, a function that will produce zero only for one point in the domain. Obfuscation of point-function has been thoroughly investigated in cryptography research [4–6, 14, 23]. Actually, a point-function is used in the proof of the impossibility result of Barak et al. [2].

The basic idea of point-function obfuscation is to use a random oracle O to produce the output $O(k)$ on input k and to compare it with the stored $O(k^*)$. The security is guaranteed by the impossibility to get k^* from knowing $O(k^*)$. In practice, the random oracle is usually substituted by a cryptographic hash function such as MD5 or SHA. Therefore, we suggest to deploy a one-way hash function in our logic encryption scheme. If we want to combine it with the one-way function insertion discussed in Section 4, we have to make it more difficult.

We also want to mix up the logic $f(x)$ with other part of the encryption, especially with k. Thus, we connect the output of the key checking $O(k) \neq O(k^*)$ to XOR gates with a randomly selected bits of x before they feed into f. Figure 7 shows the suggested circuit before further resynthesis operations are applied, where the gray XOR indicates that only a random subset of bits are modified.

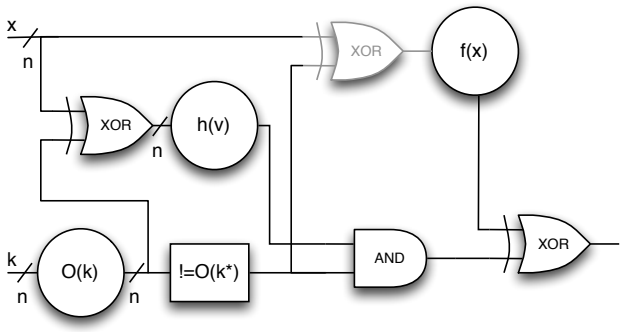

Figure 7: Suggested obfuscation for general logic encryption before further resynthesis; O(k) is one-way hash function, h(v) has an on-set of size M, and gray XOR gate only applies to a small subset of x bits.

6 SCIENTIFIC BENCHMARKS FOR APPROXIMATE ATTACKS

One big challenge for studying approximate attack methods is the lack of scientific measure of their performance. Different from the exact attacks, where the correctness can be easily measured by comparing the keys or by comparing the circuits if there are more than one correct keys, the performances of approximate attacks cannot be easily measured by the generated keys or even the generated circuits. For two approximate attacks, one is better than the other if the circuit generated by one has less error rate than that generated by the other. Exact measure of error rate needs to do circuit

simulation for all possible inputs or to do SAT to find the all errors one by one. None of them is cheap.

Not any better is the current practice of using the combination of a traditional encryption and a specific encryption against the SAT-based attack such as Anti-SAT [24]. The easy thing to report is the number of benchmarks where the key to the traditional encryption is correct. However, if an approximate attack could not get the key to the traditional encryption correct, which is very common for large or complex benchmarks, we get lost again. Measuring how many bits are correct in the key to the traditional encryption is of no use, since a mistake on one bit may have more errors than a mistake on many other bits. Using random sampling for error rate measurement is relatively cheap but its accuracy is highly in doubt.

For these reasons, it is desperate for the study of approximate attacks to develop a set of scientific benchmarks to measure their performances. The desired properties for the scientific benchmarks include:

(1) Different keys should have different error rates;
(2) The error rate is known for each key;
(3) The error rates should be adjustable;
(4) The benchmarks should be difficult for the SAT-based attack.

Based on the theory developed in Section 3, we are able to design such a suite of scientific benchmarks. The general design is given in Figure 8.

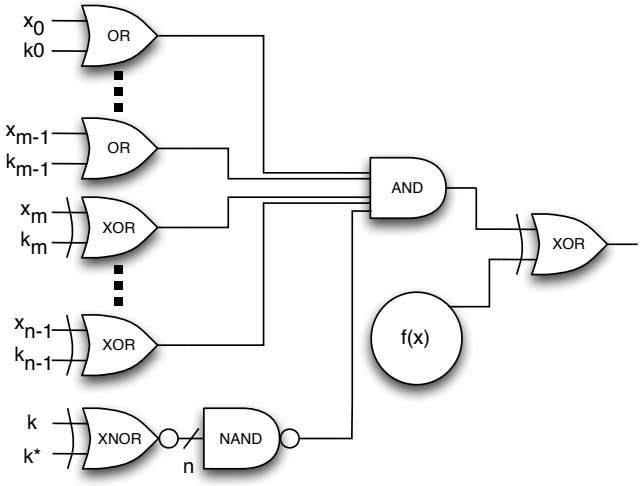

Figure 8: The general design scheme for our scientific benchmarks

Here we assume that the original circuit $f(x)$ has n input bits and the logic encryption has n key bits. We are going to select a correct key k^*, and a number $m < n$ as a design parameter. As shown in the figure, we are going to have an

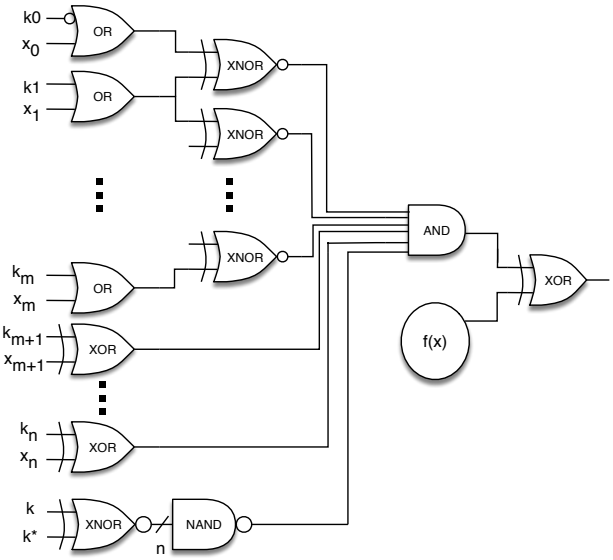

Figure 9: Scientific benchmarks with extra XNOR gates

OR of x_i and k_i for all $i \in 0..m-1$ and to have a XOR of x_i and k_i for all $i \in m..n-1$. Then we feed all these n output to an AND gate. The equivalence of k and k^* will generate a zero also to the same AND gate. Then the output of the AND gate is used to flip the output of $f(x)$ by XOR. Here, we do not worry about circuit analysis attacks to these benchmarks, since we can do resynthesis to hide the circuit structure.

We can also introduce randomness to the benchmarks by randomly selecting k^*, randomly selecting the m indices for the OR gates (the remaining will be XOR gates), and randomly inserting an inverter after each key bit in front of the circuit. A similar but more complex benchmark is given in Figure 9. Now we can prove the following theorem for our scientific benchmarks.

THEOREM 6.1. *The scientific benchmarks designed as shown in Figure 8 will have different error rates ranging from 2^{-n} to 2^{m-n}. The error rate is known for each key, and the minimal number of iterations for the SAT-based attack is 2^{n-m}.*

The upper bound of error rate happens when k_i for every $i \in 0..m-1$ is set to one, so the value of flip signal depends on the result of XOR gate. For a random assignment of k_i for all $i \in m..n-1$, the flip signal is 1 only if x_i is the opposite of k_i for all $i \in m..n-1$. So the error rate is $2^m/2^n = 2^{m-n}$.

The lower bound of error rate happens when k_i for every $i \in 0..m-1$ is set to zero, so the flip signal is one only if $x_i = 1$ for $i \in 0..m-1$ and $x_i = \bar{k_i}$ for $i \in m..n-1$. In that case, the error rate is 2^{-n}.

Any other key values will have error rate ranging from 2^{-n} to 2^{m-n}. To solve the correct key, the SAT-based attack

should prune out all wrong keys, so the number of iterations for the SAT-based attack is at least 2^{n-m}.

7 EXPERIMENTAL RESULTS

Based on the theory developed in the paper, we can design the logic encryption for any circuit to fully determine its attack complexity (in terms of the number of queries to the original circuit) and the error rate for wrong keys. In this section, we conduct experiments on the SAT-based attack [21] to verify our theory, to check the effectiveness of Goldreich's one-way function [11], and to measure the actual attack time by the SAT-based attack.

It should be noted that in our encryption, the attack complexity and error rate is independent of the original circuit $f(x)$. We have verified this by checking the attack time by SAT-based attack on the same encryption on a set of different original circuits. In fact, even using a constant function $f(x) = 0$ gives us the same result. Therefore, in our later reports, we will not going to specify what is the original circuit.

We use the simple logic encryption design shown in Figure 4. Please recall that in this encryption, the $h(v)$ function has an on-set of $2^{n/2}$. Thus, it has $2^{n/2}$ as both attack complexity and error number. We have created a sequence of encryptions with the input lengths ranging from 12 to 26. Then we run the SAT-based attack on them and collect the runtime. The result is plotted in Figure 10, where the x-axis shows the input lengths and y-axis gives the attack time in log scale.

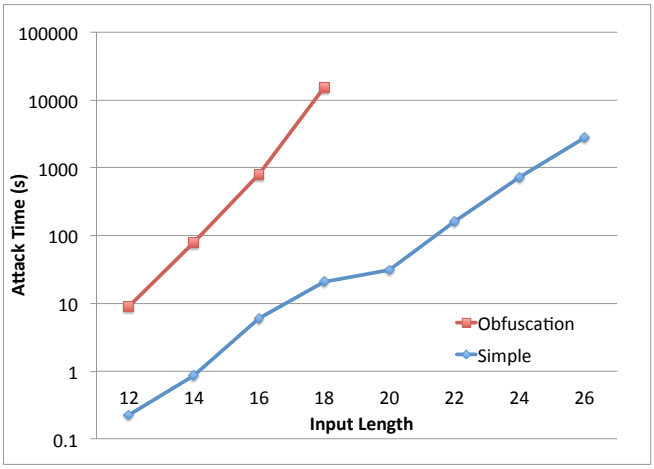

Figure 10: Runtime by SAT-based attack on simple encryptions in Figure 4 and obfuscation in Figure 7.

We also want to check the effectiveness of using Goldreich's one-way function in logic encryption. We implement the first step of obfuscation shown in Figure 7 on top of the simple logic encryption in the first set of experiments. The $O(k)$ function in the circuit is implemented by Goldreich's one-way function with an input length of 80 and an output length of 40. We use the simple $P = x_0 \oplus x_1 \oplus x_2 \oplus (x_3 \wedge x_4)$ discussed in Section 4. To minimize the disturbance on the attack complexity, the gray XOR function is not implemented. The SAT-based attack is run on these encryptions with different input bit-length and the results are shown in Figure 10.

As can be verified in Figure 10, both the simple logic encryption and its obfuscation with Goldreich's one-way function have exponential growths of attack time in terms of the input lengths. With Goldreich's one-way function, the growth of the attack time becomes bigger and steeper. They confirms our theory for logic encryption in the paper.

8 CONCLUSION

We have developed a theory for logic encryption in the paper. Our development started with a seperation of the two entangled goals in logic encryption, that is, a logic requirement of locking and a structural requirement of hiding. We consider only the logic locking in the first part of the paper, and developed a theory that gives a complete view of the logic encryption design space, and its relations with attack complexity and error rate. In the theory, we also proved a contention between attack complexity and error rate. A general logic encryption circuit of linear size is derived from the theory.

Circuit obfuscation is applied on top of our logic encryptions to address the structural requirement of hiding. We discussed the current development in cryptographic obfuscation, and showed that resynthesis operations are complete for best-possible obfuscation. We also discussed approaches to use one-way functions to protect the sensitive key checking process and to burden the SAT engine in attack.

We tested our logic encryption designs and the obfuscation with one-way function by the SAT-based attack. The experimental results have confirmed our theory and the robustness of our approach.

REFERENCES

[1] D. Apon, Y. Huang, J. Katz, and A. J. Malozemoff. 2014. Implementing Cryptographic Program Obfuscation. In *International Cryptology Conference*.

[2] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. 2001. On the (Im)possibility of Obfuscating Programs. In *International Cryptology Conference*. 1–18.

[3] A. Baumgarten, A. Tyagi, and J. Zambreno. 2010. Preventing IC Piracy Using Reconfigurable Logic Barriers. *IEEE Design and Test* 27, 1 (2010).

[4] R. Canetti. 1997. Towards realizing random oracles: Hash functions that hide all partial information. In *International Cryptology Conference (LNCS 1294)*, B. S. Kaliski Jr. (Ed.). 455–469.

[5] R. Canetti and R. R. Dakdouk. 2008. Obfuscating point functions with multibit output. In *IACR Conference on the Theory and Applications of Cryptographic Techniques (LNCS 4965)*, N. P. Smart (Ed.). 489–508.

[6] R. Canetti, Y. T. Kalai, M. Varia, and D. Wichs. 2010. On symmetric encryption and point obfuscation. In *IACR Theory of Cryptography Conference (LNCS 5978)*, D. Micciancio (Ed.). 52–71.

[7] J. Cook, O. Etesami, R. Miller, and L. Trevisan. 2009. Goldreich's One-Way Function Candidate and Myopic Backtracking Algorithms. In *IACR Theory of Cryptography Conference (LNCS 5444)*, O. Reingold (Ed.). 521–538.

[8] O. Coudert. 1996. On Solving Covering Problems. In *Proc. of the Design Automation Conf.*

[9] S. Dupuis, P.-S. Ba, G. Di Natale, M.-L. Flottes, and B. Rouzeyre. 2014. A Novel Hardware Logic Encryption Technique for Thwarting Illegal Overproduction and Hardware Trojans. In *IEEE International On-Line Testing Symposium*.

[10] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. 2013. Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits. In *Proc. IEEE Symposium on the Foundations of Computer Science*. 40–49.

[11] Oded Goldreich. 2000. Candidate One-Way Functions Based on Expander Graphs. *Electronic Colloquium on Computational Complexity* 7, 90 (2000).

[12] Shafi Goldwasser and Guy N. Rothblum. 2007. On best-possible obfuscation. In *Proceedings of the 4th conference on Theory of cryptography*. 194–213.

[13] M. Li, K. Shamsi, T. Meade, Z. Zhao, B. Yu, Y. Jin, and D. Pan. 2016. Provably Secure Camouflaging Strategy for IC Protection. In *Proc. Intl. Conf. on Computer-Aided Design*. Austin, TX.

[14] B. Lynn, M. Prabhakaran, and A. Sahai. 2004. Positive Results and Techniques for Obfuscation. In *IACR Conference on the Theory and Applications of Cryptographic Techniques*.

[15] Jeyavijayan Rajendran, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. 2012. Logic encryption: A fault analysis perspective. In *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 953–958.

[16] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri. 2013. Security analysis of integrated circuit camouflaging. In *CCS*.

[17] J. Rajendran, H. Zhang, C. Zhang, G. S. Rose, Y. Pino, O. Sinanoglu, and R. Karri. 2015. Fault Analysis-Based Logic Encryption. *IEEE Trans. Comput.* 64, 2 (Feb. 2015).

[18] Jarrod A. Roy, Farinaz Koushanfar, and Igor L. Markov. 2008. EPIC: ending piracy of integrated circuits. In *Proceedings of the conference on Design, automation and test in Europe*. 1069–1074.

[19] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Pan, and Y. Jin. 2017. AppSAT: Approximately Deobfuscating Integrated Circuits. In *Proc. IEEE International Symposium on Hardware Oriented Security and Trust*.

[20] Yuanqi Shen and Hai Zhou. 2017. Double DIP: Re-Evaluating Security of Logic Encryption Algorithms. In *Proc. ACM Great Lakes Symposium on VLSI*. Banff, AB, Canada.

[21] P. Subramanyan, S. Ray, and S. Malik. 2015. Evaluating the Security of Logic Encryption Algorithms. In *Proc. IEEE International Symposium on Hardware Oriented Security and Trust*.

[22] L. G. Valiant. 1984. A Theory of The Learnable. *Commun. ACM* 27, 11 (1984), 1134–1142.

[23] H. Wee. 2005. On obfuscating point functions. In *Proc. ACM Symposium on the Theory of Computing*, H. N. Gabow and R. Fagin (Eds.). 523âĂŞ532.

[24] Y. Xie and A. Srivastava. 2016. Mitigating SAT Attack on Logic Locking. In *Conference on Cryptographic Hardware and Embedded Systems (CHES)*.

[25] M. Yasin, B. Mazumdar, J. J. V. Rajendra, and O. Sinanoglu. 2016. SARLock: SAT attack resistant logic locking. In *Proc. IEEE International Symposium on Hardware Oriented Security and Trust*.

[26] M. Yasin, J. Rajendran, O. Sinanoglu, and R. Karri. 2016. On improving the security of logic locking. *IEEE Transactions on Computer Aided Design* 35, 9 (Sept. 2016).

[27] Hai Zhou. 2017. Structural Transformation-Based Obfuscation. In *Hardware Protection through Obfuscation*. Springer International Publishing, 221–239.