Linear Cryptanalysis Applied to Logic Locking

Sayak Chakrabarti **Bholeshwar Khurana**
Mahajan Dipak Anil
Department of Computer Science and Engineering,
Indian Institute of Technology – Kanpur

Abstract

This paper proposes Linear Cryptanalysis attacks on state of the art Logic Locking algorithms. These attacks try to approximate the encrypted circuits to linear circuits and then exploit the linearity to get the keys. More ideas have been discussed in Future Work which can possibly give better results on logic locking circuits.

1 Introduction

Currently semiconductors are used in almost every electronic devices and their manufacturing relies on offshore foundries to be used in various fields. Validation of integrated circuits have led to IC counterfeiting, piracy and unauthorized production. Financial loses and security threats have necessitated the need for encryption of circuits. The Semiconductor Industry Association (SIA) estimates that 15% of all spare and replacement semiconductors purchased by the Pentagon are counterfeit [7].

2 Logic Locking

In order to overcome the threats discussed above, various research has been conducted to improve encryption protocols on ICs to prevent unauthorized user from using or duplicating it. Logic locking is such a design for security technique which modify the ICs by inserting gates with keys so that the circuit does not produce correct output corresponding to every input with wrong key values. A study of security on logic locking and attacks on them is necessary to evaluate their strength. In this paper we try different techniques and give intuition for future work regarding a new approach to attacking a logic locking circuit.

Potential solutions to the problem of locking circuits have been given in [1, 2], which include techniques to introduce additional gates and extra inputs, key bits, to the digital circuit to create an encrypted version. The circuit operates correctly if and only if all the key bits are correct. This key bits are set after manufacturing and prior to sale, assuming that untrusted foundry does not know the correct key inputs or the original circuit.

3 Attacks on Logic Locking

Initial locking protocols were quite vulnerable to attack [3, 5, 9, 10, 11, 12]. Rajendran et al. [10] used automatic test pattern generation (ATPG) algorithms to reveal the key bits. We studied about two of Subramanyan's attacks which are described in the following subsections:

3.1 The SAT Attack

Subramanyan et al. [3] developed the SAT attack which defeated all combinational logic encryption algorithms known at the time. This attack uses Boolean Satisfiability Solver (SAT Solver) to find

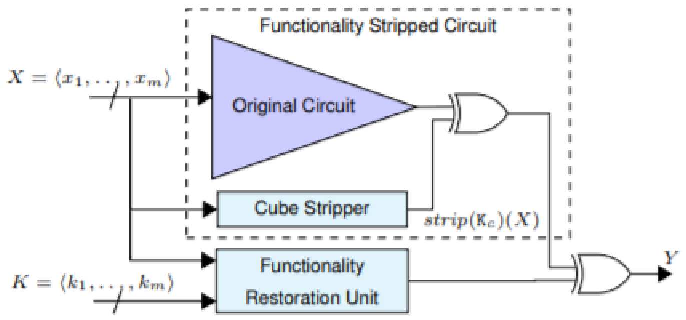
Functionality Stripped Circuit
$X = \langle x_1, \ldots, x_m \rangle$
Original Circuit
Cube Stripper
$strip(K_c)(X)$
Functionality Restoration Unit
$K = \langle k_1, \ldots, k_m \rangle$
Y

Figure 1: Overview of SAT attack resilient locking algorithms

the key bits fast enough, with several heuristics. For each input, we have an oracle (activated IC) to find the correct output, which is then fed into the SAT solver. The SAT solver computes the next distinguishing input, the input for which outputs are different. Efficiency of this attack depends on the number of equivalence classes of keys for the locking circuit.

Subsequent work has largely focused on developing SAT-attack resistant logic locking [13, 14, 15, 16, 17] to ensure that the number of equivalence classes of keys is exponential in the order of length of key bits. The proposals share mainly the structure shown in Figure 1.

A circuit is present which "flips" the output of the original circuit for a particular subset of the circuit, called cube. We refer to this component as cube stripping unit. This flipped output is then inverted by a key-dependent circuit which is referred to as programmable functionality restoration unit. This scheme ensures exponential number of equivalence classes successfully resisting SAT attack.

Algorithm:
The input vectors to the circuit are $\vec{X_1}, \vec{X_2} \ldots \vec{X_p}$ and the corresponding observed outputs are $Y_1, \vec{Y_2} \ldots Y_p$. The encrypted combinational circuit is represented as $C(\vec{X}, \vec{K}, \vec{Y}) \in \{0,1\}^{M+L+N}$ where $\vec{X} \in \{0,1\}^M$ is the input, $\vec{K} \in \{0,1\}^L$ is the entered key and $\vec{Y} \in \{0,1\}^N$ is the corresponding output of encrypted circuit. $eval$ is the activated IC. $eval(\vec{X}) = \vec{Y}$ means that the oracle (activated IC) returns $\vec{Y}$ when input $\vec{X}$ is passed. Correct key of the circuit is denoted as K_C.

Algorithm 1: Decryption Algorithm [3]

```
Result: Write here the result
Inputs: C and eval;
$F_1 \leftarrow C(\vec{X}, \vec{K_1}, \vec{Y_1}) \wedge C(\vec{X}, \vec{K_2}, \vec{Y_2})$;
while $sat[F_i \wedge \vec{Y_1} \neq \vec{Y_2}]$ do
    $\vec{X_i^d} \leftarrow sat\_assignment_{\vec{X}}[F_i \wedge \vec{Y_1} \neq \vec{Y_2}]$;
    $\vec{Y_i^d} \leftarrow eval(\vec{X_i^d})$;
    $F_{i+1} \leftarrow F_i \wedge C(\vec{X_i^d}, \vec{K_1}, \vec{Y_i^d}) \wedge C(\vec{X_i^d}, \vec{K_2}, \vec{Y_i^d})$;
    $i \leftarrow i + 1$;
end
```

3.2 FALL Attack

Sirone et al. [5] developed FALL attacks which defeat locking methods that use cube stripping and programmable functionality restoration. They identify sub-circuits corresponding to the cube stripping module and then extract the key using functional analysis of these nodes. The main ideas are based on the use of computationally cheap structural analyses that identify the cube stripping unit and use this in combination with specific Boolean functional properties of the cube stripping function for TTLock and SFLL. These structural and functional analyses defeat SFLL, which at the time, was the locking method resilient to all known attacks.

4 Contributions

In this paper we try to look at the new possibility of attacks: attacks derived from Linear Cryptanalysis. We try to approximate the encrypted circuits to linear circuits and then exploit the linearity to get the keys. We describe our approaches in the subsequent sections.
We also tried to break circuits locked with SFLL-fault plus random logic locking keys as part of CSAW-LLC 2019 [18] (CSAW - Logic Locking Competition).

5 Applications from Cryptography: Linear Cryptanalysis

Linear Cryptanalysis was an attack proposed by Matsui to break the DES Cryptosystem. It was an attack based on sending random inputs to the SASAS Cipher and checking the outputs to see if a linear combination exists. If we find a linear combination with the inputs we can iterate over all the subkeys. The linear combination consisting of inputs and the outputs can be found by giving a large number of random inputs, say N, to the cipher and checking the output. For each combination we create a counter and check the combination which given the highest deviation from mean ($N/2$). The linear combination with the highest deviation from mean behaves most linearly as it is the least random.

5.1 Linear Cryptanalysis to DES S-boxes

The S-boxes in DES are many-to-one functions mapping from $\{0,1\}^6$ to $\{0,1\}^4$. By linear cryptanalysis we aim to approximate these functions into linear forms as the encrypted inputs to the S-boxes are Xor-ed with the actual inputs. Linear cryptanalysis in general, and on logic locking circuits, have been explained in the following sub-section.

5.2 Linear Cryptanalysis on Logic Circuits

Let $a_1, a_2 \ldots a_n$ be the inputs to the logic locking system, and $x_1, x_2 \ldots x_m$ be its outputs. We find the linear combination by considering the equation:

$$C_1 a_1 \oplus C_2 a_2 \oplus \cdots \oplus C_n a_n \oplus C_{n+1} x_1 \oplus C_{n+2} x_2 \oplus \cdots \oplus C_{n+m} x_m = 0 \tag{1}$$

We are given with the original circuit where we pass N random inputs corresponding to $\{a_i \mid i \in [n]\}$ and store the outputs $\{x_i \mid i \in [m]\}$. Then we create a table with 2^n columns and 2^m rows, where the boolean value of the number of column (indexing from 0) corresponds to the coefficients $C_1 \ldots C_n$ being 0 or 1. Similarly the rows correspond to the value of coefficients $C_{n+1} \ldots C_{n+m}$. For example, if $n = 3$ and $m = 2$, the value of the 3^{rd} row (boolean value of 2) and the 6^{th} column (boolean value of 5) corresponds to the equation

$$a_1 \oplus a_3 \oplus x_1$$

Now we fill up this table with N random inputs and check the output of the original, unencrypted logic circuit. If we encounter 0 we increase the corresponding table entry by 1.
Now we subtract $N/2$ from each table entry. If the value is now negative, the corresponding equation must give a value 1 and if it is positive the corresponding equation must give 0. The table entry with the highest absolute value is the best linear approximation of our logic circuit. [1]
The chosen linear model performs best in one subkey of all the given inputs, and we chose this part of the key as the actual key. We first create an array with the different possibilities with the keys and send m random inputs to the cryptosystem with the same key. For each key we create a counter which gives correct output with the corresponding key. The key to which highest number of correct outputs are observed is chosen as the correct key. [2]

[1]We will try another approach to finding the coefficients in Section 5.2 and another way to find the linear approximation in Section 5.3

[2]Checking over all the key bits took a lot of time. To improve this we will try to modify the SAT Attack, discussed in Section 6

6 Applying Linear Cryptanalysis

6.1 Initial approach (Brute-force)

We used brute force initially to find the best set of coefficients which was giving us the best linear model. We calculated the probability of every combination possible and then found out the one which gave the maximum probability. Since this would take exponential amount of time $O(2^{i+k})$ where $i =$ number of input bits and $k =$ number of key bits, the method could work only for a maximum of 10 bits key.

We created some of our own examples with 10 bits keys and tested this brute force approach. It took quite much time to get the results but the results obtained were not that bad. The correct key was always present amongst the top 16 probabilities but it wasn't the one with the maximum probability.

We got an intuition that this method could work or at least we could deduce some important results from this method, so we tried to test this on larger circuits. We used various approaches to fasten the algorithm. One of which was using Genetic algorithm to obtain the best set of coefficients. Another one was MaxSAT. Both of them are described in the subsequent sections.

6.2 Genetic approach

As per Wikipedia, "In computer science and operations research, a genetic algorithm (GA) is a meta-heuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA)". Genetic algorithms are inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

For genetic algorithm to proceed we need a fitness function to decide whether the coefficients are better than the ones in the previous generation i.e. we need to decide whether the children of the next generation are fit. We used bias of the coefficients for deciding the fitness. Bias of a given set of coefficients is the absolute difference distance of the probability that the coefficients are approximate representation of the circuit from 1/2. Suppose we have an approximation of the circuit $A^i = \{a^i_j | j \in [m+n+k]\}$ which will be true for a given set of input,ouput and key values if -

$$\bigoplus_{i=0}^{m-1} a_i * x_i \oplus \bigoplus_{i=0}^{n-1} a_{i+m} * y_i \oplus \bigoplus_{i=0}^{k-1} a_{m+n+i} * k_i = 0$$

Now if the equation is satisfied for n_t such input-output-key pairs out of n_T total testings then bias is defined as $|(n_t/n_T) - 1/2|$.So we will generate new coefficients by making some changes to the set of coefficients in the current generation and look at the bias values of the coefficients generated. If the new generation coefficients are having more bias than some of the sets of coefficients in the current set then we remove the ones with lowest bias and add the new generation coefficients and thus the algorithm proceeds. There is no guarantee that the algorithm will end at the set of coefficients that is the best approximation for the circuit out of all the set of coefficients but as the problem of finding the best approximation by trying the circuit for all the possible approximations needs exponential time genetic algorithm provides us a heuristic to find out some set of coefficients that is going to be a decent approximation of the circuit. The algorithm terminates when the changes in the best bias of the set becomes negligible between two consecutive generations.

Algorithm 2: Genetic Selection of Coefficients

```
Result: Write here the result
Initialize randomly $k$ coefficients $C_1, C_2 \dots C_k$;
for $j$ in $1 : iters$ do
    Calculate bias on $C_i \; \forall i \in [k]$;
    Select worst performing coefficient $C_a$;
    Select second-worst performing coefficient $C_b$;
    for $bit$ in $1 : length$ do
        $C_a[bit] = \text{random}(C_a[bit], C_b[bit])$
    end
end
```

6.3 MaxSAT:

In order to find out the set of coefficients in the equation 5.3,
i.e $A = \{a_i \forall i \in [m+n+k], a_i \in \{0,1\}\}$

$$\bigoplus_{i=0}^{m-1} a_i * x_i \oplus \bigoplus_{i=0}^{n-1} a_{i+m} * y_i \oplus \bigoplus_{i=0}^{k-1} a_{m+n+i} * k_i = 0 \qquad \text{...(5.3)}$$

With

$$\sum_{i=0}^{m+n+k-1} a_i \neq 0$$

i.e. Not all of them are zero simultaneously.
We could consider the coefficients as different variables which can take two values i.e 0 or 1. Now we want the set A such that the equation 5.3 turns out to be true for most of the possible cases possible i.e for different assignments to the input bits,output bits and key bits. Note that we can not have all the coefficients zero simultaneously. Also note that we wanted an approximation for the given circuit with a good bias i.e. we wanted to find an assignment to the coefficients such that either the equation 5.3 turns out true with very high probability or with a very low probability as it works out in either case. So we want the equation 5.3 to be true for most of the cases or not true for most of the cases. In the cases when the equation 5.3 fails we want the RHS of the equation to be 1. Now if we had a linear circuit in which we had to find set A such that the equation 5.3 is true for all the cases (possible assignments for x,y and k) then we could have used SAT solver to solve the problem by giving all the possible equations as input.
We define condition C:$\{0,1\}^m\{0,1\}^n\{0,1\}^k \longmapsto \{0,1\}$; such that C(X,Y,K) = 1, if

$$\bigoplus_{i=0}^{m-1} a_i * x'_i \oplus \bigoplus_{i=0}^{n-1} a_{i+m} * y'_i \oplus \bigoplus_{i=0}^{k-1} a_{m+n+i} * k'_i = 0$$

Where $X = \{x'_i | i \in [m]\}$,$X = \{y'_i | i \in [n]\}$,$K = \{k'_i | i \in [k]\}$.

Now for a problem in which there exists a perfect assignment for A such that the equation 5.3 is true for all possible X,Y,K we can find the value of A by giving all the possible expressions for C(X,Y,K) as clauses. If we consider C(X,Y,K) = 1 as a clause for a given value of X,Y and K then it signifies that the equation is true for that particular assignment of the input, output and key bits. Now if there is value of A which makes all the clauses true then it means that for all the assignments the equation is true. Not that we have to give an extra clause to ensure that not all of them are zero simultaneously as that is one trivial solution for the equation which should not be considered.
Now coming back to the approximation problem, we have many clauses C for some assignments of input bits, output bits and key bits i.e. X,Y and K values. So our problem changes to maximizing the number of conditions satisfied out of all the conditions possible. Note that we are testing the circuit for specific number of times and observing the input in the applied key and input pairs. So we do not

have all the conditions possible but a finite number of conditions should give us the idea about the satisfiable assignment of the variables that satisfies most of the conditions. So our problem changes into finding out an assignment of variables for which maximum number of conditions are satisfied.

6.3.1 The Problem

We are going to use MaxSAT solver to find out the assignment of variables. The main problem that remains now is that MaxSAT takes in clauses as input and tries to maximise the number of clauses satisfied and finds a model for that particular optimal solution. But if we look at the conditions, the conditions consist of XORs of many literals. Now even if we consider having two literals in the given condition we can not give that particular condition as input to the MaxSAT as we need two clauses to specify that condition. Recall that $A \oplus B \models (A \wedge \neg B) \vee (\neg A \wedge B) \models \neg((A \vee \neg B) \wedge (\neg A \vee B))$
Therefore for $A \oplus B = 0 \implies \neg((A \vee \neg B) \wedge (\neg A \vee B)) = 0 \implies (A \vee \neg B) \wedge (\neg A \vee B) = 1$
So we need two clauses $(A \vee \neg B)$ and $(\neg A \vee B)$. For more than two literals in the condition we are going to require a lot of clauses to specify the CNF of the MaxSAT problem.

6.3.2 Tseytin Encoding

To solve the above problem we are going to use Tseytin Encoding for transforming the conditions into form so that we will be able to give it as an input to the MaxSAT solver. We are going to introduce new literals into the problem. Let the total number of conditions be c. Let X^i,Y^i and K^i be the input given, output observed and the keys used in i^{th} testing.
Let P^i = { p_1,p_2,p_3...p_d } the set of indices for which the input bits x_{p_j} = 1, $\forall j \in [d]$.
Let Q^i = { q_1,q_2,q_3...q_d } the set of indices for which the output bits y_{p_j} = 1, $\forall j \in [e]$.
Let R^i = { r_1,r_2,r_3...r_f } the set of indices for which the input bits k_{r_j} = 1, $\forall j \in [f]$. Note that we are talking about i^{th} testing as of for now. We define (d+e+f) new variables ξ^i_j $\forall j \in [d+e+f]$. We will represent the input,output and key bits into a single array T^i for i^{th} input
i.e. T^i = { x_{p_1},x_{p_2}...x_{p_d},y_{q_1},y_{q_2}...y_{q_e},k_{r_1},x_{r_2},...k_{r_f} } in that order where T^i_j represents j^{th} element in that array T^i.
Now we transform our condition by adding following conditions:

$$\xi^i_{j-1} \oplus T^i_{j+1} \leftrightarrow \xi^i_j \qquad\qquad \forall j \in [d+e+f-1]$$

Here $\xi^i_0 = T^i_0$ Now after adding those conditions to the list of all the conditions for all the inputs i.e $\forall i \in [c]$ we can consider all those to be hard clauses in the MaxSAT problem (section 6.3.3).
Now

$$\xi^i_{j-1} \oplus T^i_{j+1} \leftrightarrow \xi^i_j \models (\xi^i_j \rightarrow \xi^i_{j-1} \oplus T^i_{j+1}) \wedge (\xi^i_{j-1} \oplus T^i_{j+1} \rightarrow \xi^i_j) \tag{6.3.2.1}$$

$$\xi^i_{j-1} \oplus T^i_{j+1} \rightarrow \xi^i_j \models \neg(\xi^i_{j-1} \oplus T^i_{j+1}) \vee \xi^i_j$$

Theorem : $\neg(\xi^i_{j-1} \oplus T^i_{j+1}) \vee \xi^i_j \models (\neg\xi^i_{j-1} \vee T^i_{j+1} \vee \xi^i_j) \wedge (\xi^i_{j-1} \vee \neg T^i_{j+1} \vee \xi^i_j)$

Proof. We know that $\xi^i_{j-1} \oplus T^i_{j+1} \models (\neg\xi^i_{j-1} \wedge T^i_{j+1}) \vee \xi^i_{j-1} \wedge \neg T^i_{j+1})$
Therefore, $\neg(\xi^i_{j-1} \oplus T^i_{j+1}) \models \neg((\neg\xi^i_{j-1} \wedge T^i_{j+1}) \vee (\xi^i_{j-1} \wedge \neg T^i_{j+1})) \models \neg(\neg\xi^i_{j-1} \wedge T^i_{j+1}) \wedge \neg(\xi^i_{j-1} \wedge \neg T^i_{j+1})$ Let χ be the formula $\neg(\xi^i_{j-1} \oplus T^i_{j+1})$

$$\chi \models (\xi^i_{j-1} \vee \neg T^i_{j+1}) \wedge (\neg\xi^i_{j-1} \vee T^i_{j+1})$$

$$\chi \vee \xi^i_j \models \chi \vee (\xi^i_j \wedge (\xi^i_j \vee \top))$$

$$\chi \vee \xi^i_j \models \chi \vee (\xi^i_j \wedge (\xi^i_j \vee (\xi^i_{j-1} \vee \neg T^i_{j+1}) \vee (\neg\xi^i_{j-1} \vee T^i_{j+1})))$$

$$\chi \vee \xi^i_j \models ((\xi^i_{j-1} \vee \neg T^i_{j+1}) \wedge (\neg\xi^i_{j-1} \vee T^i_{j+1})) \vee (\xi^i_j \wedge (\xi^i_j \vee (\xi^i_{j-1} \vee \neg T^i_{j+1}) \vee (\neg\xi^i_{j-1} \vee T^i_{j+1})))$$

$$\chi \vee \xi^i_j \models (\neg\xi^i_{j-1} \vee T^i_{j+1} \vee \xi^i_j) \wedge (\xi^i_{j-1} \vee \neg T^i_{j+1} \vee \xi^i_j)$$

Hence , $\neg(\xi^i_{j-1} \oplus T^i_{j+1}) \vee \xi^i_j \models (\neg\xi^i_{j-1} \vee T^i_{j+1} \vee \xi^i_j) \wedge (\xi^i_{j-1} \vee \neg T^i_{j+1} \vee \xi^i_j)$
Therefore, $(\xi^i_{j-1} \oplus T^i_{j+1}) \rightarrow \xi^i_j \models (\neg\xi^i_{j-1} \vee T^i_{j+1} \vee \xi^i_j) \wedge (\xi^i_{j-1} \vee \neg T^i_{j+1} \vee \xi^i_j)$ (6.3.2.2)
□

This theorem 6.3.2.2 converts the formula into CNF form. Now to convert the whole condition into clauses we need to convert the remaining part into CNF form as well.

Theorem: $(\xi^i_j \rightarrow \xi^i_{j-1} \oplus T^i_{j+1}) \models (\xi^i_{j-1} \vee T^i_{j+1} \vee \neg\xi^i_j) \wedge (\neg\xi^i_{j-1} \vee \neg T^i_{j+1} \vee \neg\xi^i_j)$

Proof. Let Γ be the formula $\xi^i_{j-1} \oplus T^i_{j+1}$. Observe that Γ is just $\neg\chi$

$$\Gamma \models (\xi^i_{j-1} \wedge \neg T^i_{j+1}) \vee (\neg\xi^i_{j-1} \wedge T^i_{j+1})$$

$$\xi^i_j \rightarrow \Gamma \models \neg\xi^i_j \vee \Gamma$$

$$\xi^i_j \rightarrow \Gamma \models \neg\xi^i_j \vee (\xi^i_{j-1} \wedge \neg T^i_{j+1}) \vee (\neg\xi^i_{j-1} \wedge T^i_{j+1})$$

$$\xi^i_j \rightarrow \Gamma \models \neg\xi^i_j \vee (\xi^i_{j-1} \wedge \neg T^i_{j+1}) \vee (\neg\xi^i_{j-1} \wedge T^i_{j+1}) \vee \bot$$

$$\xi^i_j \rightarrow \Gamma \models \neg\xi^i_j \vee (\xi^i_{j-1} \wedge \neg T^i_{j+1}) \vee (\neg\xi^i_{j-1} \wedge T^i_{j+1}) \vee (\xi^i_{j-1} \wedge \neg\xi^i_{j-1}) \vee (T^i_{j+1} \wedge \neg T^i_{j+1})$$

$$\xi^i_j \rightarrow \Gamma \models (\neg\xi^i_j \wedge (\neg\xi^i_j \vee \xi^i_{j-1} \vee T^i_{j+1} \vee \neg\xi^i_{j-1} \vee \neg T^i_{j+1})) \vee (\xi^i_{j-1} \wedge \neg T^i_{j+1}) \vee (\neg\xi^i_{j-1} \wedge T^i_{j+1}) \vee (\xi^i_{j-1} \wedge \neg\xi^i_{j-1}) \vee (T^i_{j+1} \wedge \neg T^i_{j+1})$$

$$\xi^i_j \rightarrow \Gamma \models (\xi^i_{j-1} \vee T^i_{j+1} \vee \neg\xi^i_j) \wedge (\neg\xi^i_{j-1} \vee \neg T^i_{j+1} \vee \neg\xi^i_j)$$

Therefore,

$$(\xi^i_j \rightarrow \xi^i_{j-1} \oplus T^i_{j+1}) \models (\xi^i_{j-1} \vee T^i_{j+1} \vee \neg\xi^i_j) \wedge (\neg\xi^i_{j-1} \vee \neg T^i_{j+1} \vee \neg\xi^i_j) \qquad (6.3.2.3)$$

From claims 6.3.2.1, 6.3.2.2 and 6.3.2.3 we can convert the condition into CNF form as follow:

$$\xi^i_{j-1} \oplus T^i_{j+1} \leftrightarrow \xi^i_j \models ((\xi^i_{j-1} \vee T^i_{j+1} \vee \neg\xi^i_j) \wedge (\neg\xi^i_{j-1} \vee \neg T^i_{j+1} \vee \neg\xi^i_j)) \wedge ((\neg\xi^i_{j-1} \vee T^i_{j+1} \vee \xi^i_j) \wedge (\xi^i_{j-1} \vee \neg T^i_{j+1} \vee \xi^i_j))$$

□

6.3.3 Hard and Soft Clauses

We are going to use the variant of MaxSAT in which we have a set of hard clauses which is basically the set of clauses that must be satisfied among the set of all clauses. For the remaining clauses MaxSAT tries to maximise the number of clauses satisfied and returns the optimal assignment of the variables. In section 6.3.1 we said that there are conditions for each testing that where the number of satisfied conditions are to maximised. So in order to maximise the number of conditions we need to have a clause for each condition.
From section 6.3.2 we saw that the conditions can be converted into clauses using Tseytin encoding but there are lots of clauses for each condition. Now the idea for giving input to MaxSAT is that we will add the clauses generated in the section 6.3.2 as soft clauses. Now as soft clauses have to be satisfied we will get some relationship between the original variables in the circuit and the newly introduced variables. After that observe that $\xi^i_{|T^i|-1} = 1$ implies that i^{th} condition is satisfied. So there we have a clause for each of the testing and the clause contains only one literal aka $\xi^i_{|T^i|-1}$.
So this is how we formulate our problem as a MaxSAT problem. Note that regardless of the assignment of the newly introduced variables, the soft clauses are going to be satisfied for all the possible assignments of the testing variables. We can easily prove this as the encoded clauses are just representations of the conditions introduced earlier in section 6.3.2. Also each condition is equivalent to :

$$\bigoplus_{j=0}^{m-1} a_j * x^i_j \oplus \bigoplus_{j=0}^{n-1} a_{j+m} * y^i_j \oplus \bigoplus_{j=0}^{k-1} a_{m+n+j} * k^i_j \leftrightarrow \xi^i_{|T^i|-1}$$

7 Attacking circuits locked with SFLL-fault plus random keys

7.1 Stripped functionality Logic Locking (SFLL)

After SAT attack was developed, research started happening on advanced locking techniques that were resilient to SAT attacks. One of which was TTlock [17]. In TTlock, the original circuit is modified for exactly one input pattern and the output for this protected pattern is restored using a comparator block. Somehow if the attacker can remove the comparator block, he/she still is left with a different design from the original circuit.

Motivated by TTlock, Yasin et al. [16] proposed stripped functionality logic locking (SFLL). SFLL is resilient against most of the current attacks like SAT attacks and removal attacks [16]. It is based on the concept of "strip and restore", where some part of the original circuit is removed and the intended functionality is hidden. SFLL has three variants: SFLL-HD [16], SFLLflex [16] and SFLL-fault [19].

SFLL-HD (here HD stands for Hamming distance) allows the designer to protect a larger number of input patterns as compared to TTlock. More specifically, SFLL-HD^h protects $\binom{k}{h}$ input cubes which are Hamming distance (HD) h away from the k-bit secret key. As a result SFLL-HD has a restriction on the input cubes which can be protected.
In contrast, SFLL-flex$^{c \times k}$ allows to preotect any c selected input cubes, each with k specified bits. In SFLL-fault, fault injection-based heuristics are applied to protect multiple input patterns.

The problem

We had two sets of benchmark circuits (available at [2] locked with x bits of SFLL-fault key plus x bits of Random Logic Locking key where $x = \{40, 60, 80\}$. One of the set of benchmarks had their corresponding oracle while the other didn't have. Our objective was to break the circuits and extract their keys.

Our approach

There doesn't exist any well-known attack to break circuits locked with SFLL-fault logic locking currently. However, we used some brain storming to get the keys of randomly locked gates. As described above, SFLL circuits contain a cube stripper and functionality restoration unit other than the original circuit. They basically consist of comparators which are hardcoded in the benchmark circuits. We extracted the gates with these comparators and their outputs were grounded (set to 0). We then removed the corresponding keys which were responsible for SFLL-fault. We were now left only with circuits locked with random logic locking.

These circuits could be solved using Subramanyan's SAT attack. However SAT attack requires the benchmark of the original circuit but we only had their oracles. We modified the SAT attack to work with oracles rather than benchmark circuits of the original circuit and found the keys of the randomly locked circuits. The complete benchmarks, code and results are available at [20]

8 Future Work

We have tried the proposed Linear Cryptanalysis on small logic locking circuits. However we faced certain challenges and our propositions to overcome them are:

1. The main obstruction was with a high number of key bits. It is practically infeasible to run the algorithm checking each key bit for the random inputs. So we are trying to incorporate the linear model into the SAT attack. SAT attack finds a CNF form of the function C, as in the pseudocode. We are proposing to replace this function by the linear combination as found above. We are hoping this might improve the SAT attack by eliminating SARLock schemes as it will be reduced to a linear form.
2. The accuracy is not very high. In fact the key was estimated in a small fraction of the few highly probably keys. So instead of taking random inputs to find the key counter, we will aim to use the SAT solving technique to find the exact key value.
3. If a set of highly probable keys can be found, we can use a modified version of SAT attack to find the actual key

References

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

[2] R.S. Chakraborty and S. Bhunia. Hardware Protection and Authentication Through Netlist Level Obfuscation. In IEEE/ACM International Conference on ComputerAided Design, 2008.

[3] P. Subramanyan, S. Ray, and S. Malik. Evaluating the Security Logic Encryption Algorithms. In 2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), 2015.

[4] Matsui M. (1994) "Linear Cryptanalysis Method for DES Cipher", Helleseth T. (eds) Advances in Cryptology — EUROCRYPT '93. EUROCRYPT 1993. Lecture Notes in Computer Science, vol 765. Springer, Berlin, Heidelberg.

[5] D. Sirone, P. Subramanyan, "Functional Analysis on Logic Locking", Proceedings of Design Automation and Test in Europe. (2019). Florence, Italy. March 2019.

[6] H. M. Heys, "A tutorial on linear and differential cryptanalysis", Journal Cryptologia Vol. 26 Issue 3, July 2002 pp. 189-221.

[7] Semiconductor Industry Association: Anti-Counterfeiting Whitepaper One-Pager. http://www.semiconductors.org/clientuploads/directory/DocumentSIA/Anti%20Counterfeiting%20Task%20Force/ACTF%20Whitepaper%20Counterfeit%20One%20Pager%20Final.pdf, 2013.

[8] E. Biham, A. Shamir, "Differential Cryptanalysis on DES-like Cryptosystems", Journal of Cryptology, Vol. 4, pp. 3-72, (1991)

[9] S.M. Plaza and I.L. Markov. Solving the third-shift problem in ic piracy with test-aware logic locking. In IEEE Transactions on CAD of Integrated Circuits and Systems, 2015.

[10] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri. Security Analysis of Logic Obfuscation. In Proceedings of the Design Automation Conference, 2012.

[11] M. Yasin, B. Mazumdar, S.S. Ali, and Sinanoglu O. Security Analysis of Logic Encryption against the Most Effective Side-Channel Attack: DPA. In IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, 2015.

[12] M. Yasin, S.M. Saeed, J. Rajendran, and O. Sinanoglu. Activation of logic encrypted chips: Pre-test or post-test? In Design, Automation Test in Europe, 2016.

[13] Y. Xie and A. Srivastava. Mitigating SAT Attack on Logic Locking. In International Conference on Cryptographic Hardware and Embedded Systems, 2016.

[14] Y. Xie and A. Srivastava. Anti-sat: Mitigating sat attack on logic locking. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018.

[15] M. Yasin, B. Mazumdar, J. J. V. Rajendran, and O. Sinanoglu. SARLock: SAT attack resistant logic locking. In 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pages 236–241, 2016.

[16] Muhammad Yasin, Abhrajit Sengupta, Mohammed Thari Nabeel, Mohammed Ashraf, Jeyavijayan (JV) Rajendran, and Ozgur Sinanoglu. Provably-secure logic locking: From theory to practice. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17, 2017.

[17] Muhammad Yasin, Abhrajit Sengupta, Benjamin Carrion Schafer, Yiorgos Makris, Ozgur Sinanoglu, and Jeyavijayan (JV) Rajendran. What to lock?: Functional and parametric locking. In Proceedings of the on Great Lakes Symposium on VLSI 2017, 2017.

[18] csaw-llc 2019: https://sites.google.com/nyu.edu/logiclocking19.

[19] A. Sengupta, M. Nabeel, M. Yasin and O. Sinanoglu, "ATPG-based cost-effective, secure logic locking," 2018 IEEE 36th VLSI Test Symposium (VTS), San Francisco, CA, 2018, pp. 1-6.

[20] csaw-llc 2019 project: https://git.cse.iitk.ac.in:spramod/csaw-llc-2019.git