Evaluating the Security of Logic Encryption Algorithms

Pramod Subramanyan, Sayak Ray and Sharad Malik
Department of Electrical Engineering, Princeton University.

***Abstract*—Contemporary integrated circuits are designed and manufactured in a globalized environment leading to concerns of piracy, overproduction and counterfeiting. One class of techniques to combat these threats is logic encryption. Logic encryption modifies an IC design such that it operates correctly only when a set of newly introduced inputs, called *key inputs*, are set to the correct values.**

In this paper, we use algorithms based on satisfiability checking (SAT) to investigate the security of logic encryption. We present a SAT-based algorithm which allows an attacker to "decrypt" an encrypted netlist using a small number of carefully-selected input patterns and their corresponding output observations. We also present a "partial-break" algorithm that can reveal some of the key inputs even when the attack is not fully successful. We conduct a thorough evaluation of our attack by examining six proposals for logic encryption from the literature. We find that all of these are vulnerable to our attack. Among the 441 encrypted circuits we examined, we were able to decrypt 418 (95%). We discuss the strengths and limitations of our attack and suggest directions that may lead to improved logic encryption algorithms.

I. Introduction

Considerations of capital costs and economies of scale dictate that semiconductor manufacturing is now reliant on offshore foundries that are organizationally separate and geographically distant from the design houses that design and validate integrated circuits (ICs). This has led to increased concerns of IC counterfeiting, piracy and unauthorized overproduction by the contract foundry [7, 16, 23]. The financial risk due to counterfeit and unauthorized ICs was estimated to be over $169 billion a year by IHS Technology [11]. Besides financial losses, this issue potentially has national security implications. The Semiconductor Industry Association (SIA) estimates that 15% of all spare and replacement semiconductors purchased by the Pentagon are counterfeit [20].

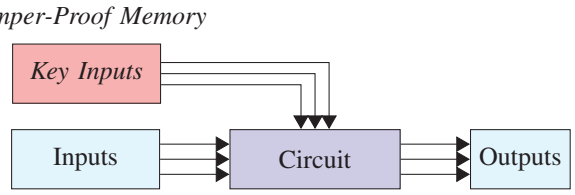


Fig. 1: Overview of logic encryption.

One set of techniques to prevent counterfeiting and overproduction by an untrusted foundry is called *logic encryption* [1, 4, 8, 14, 17–19].[1] The insight underlying these proposals is shown in Figure 1. Additional logic is introduced to the IC and is connected to a set of newly introduced inputs, called *key inputs*, as well as some parts of the original IC. The key inputs are connected to a tamper-proof memory and the modified IC produces the correct output only if the key inputs are set correctly. This correct value is not revealed to the foundry. The foundry manufactures the IC and returns the fabricated units to the design house. The design house loads the tamper-proof memory with the correct key value, "activating" the IC and the activated IC is marketed to end-users.

A. Attacks on Logic Encryption

Logic encryption rests upon the assumption that *the foundry does not know and cannot compute* the correct values of the key inputs. Otherwise, the foundry could just program these values and overproduction could not be prevented.

1) Attack Model: Considering a malicious foundry, we assume the attacker has access to layout and mask information. The gate-level netlist can be reverse-engineered from this [22]. We also assume that the attacker has access to an activated IC on which to apply input patterns and observe outputs. This could be obtained by purchasing an activated IC from the open market. *The components of our attack model are therefore: (i) a gate-level netlist of the encrypted IC and (ii) a means for applying arbitrary input patterns and observing the resultant outputs on an activated IC.*[2]

2) Potential Attacks: Given the above attack model, an attack is possible when an attacker can determine the correct values of the key inputs. Let us consider potential attacks. The naïve idea of brute-force search does not work. If the circuit has M inputs and L key inputs, this requires 2^M observations from an activated IC and $O(2^{M+L})$ computations on the encrypted design. Clearly, this is not practical.

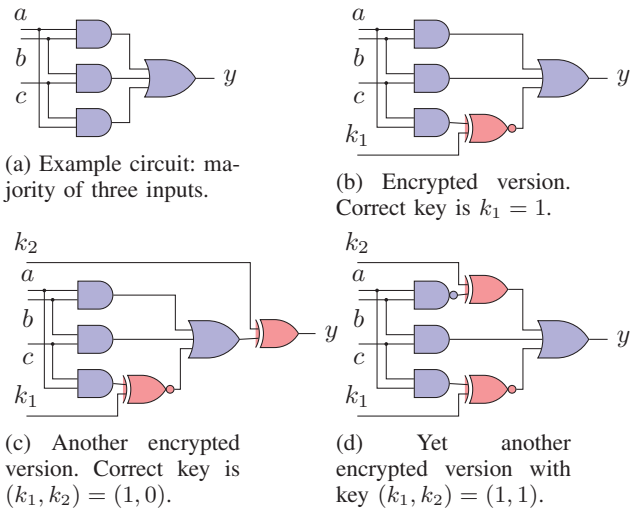
Rajendran et al. [17] propose using automatic test pattern generation (ATPG) tools [3] to generate input patterns that expose the value of a key input. In Figure 2(b), when the input is $a = b = c = 0$, the output is $y = \neg k_1$. Therefore, an ATPG tool can find such patterns to reveal keys. But in Figure 2(c), no input pattern can expose k_1 if k_2 is also unknown. The solution is to first attack k_2 with the pattern $a = b = c = 1$. From this we deduce $k_2 = 0$. Now attacking k_1 is possible. But even this strategy does not guarantee a successful attack. For the circuit in Figure 2(d), there is no single input pattern that can reveal the value of either k_1 or k_2. *Therefore, if logic encryption is done carefully, the fault-analysis attack is ineffective.*

This work was supported in part by the Center for Future Architectures Research (C-FAR), one of the six SRC STARnet Centers, sponsored by MARCO and DARPA.

[1]The terms *active hardware metering* [13], and *logic obfuscation* [17] have also been used for this technique. This paper only uses *logic encryption*.

[2]We note that other attacks are possible. For example, an attacker may evade/subvert the tamper-resistant packaging and probe the signals corresponding to the key inputs.

978-1-4673-7421-7/15/$31.00 ©2015 IEEE



(a) Example circuit: majority of three inputs.

(b) Encrypted version. Correct key is $k_1 = 1$.

(c) Another encrypted version. Correct key is $(k_1, k_2) = (1, 0)$.

(d) Yet another encrypted version with key $(k_1, k_2) = (1, 1)$.

Fig. 2: Simple logic encryption example.

Formal analyses can potentially be used to attack logic encryption. Roy et al. [19] suggest formulating the attack as a solution to a quantified boolean formula (QBF). Suppose the original circuit is represented by the Boolean function $C_O(\vec{X})$, and its encrypted version is represented by $C(\vec{X}, \vec{K})$. Here $\vec{X}$ and $\vec{K}$ are the circuit's primary inputs and key inputs respectively. We can retrieve the key by solving the QBF: $\exists \vec{K} \ \forall \vec{X} \ C(\vec{X}, \vec{K}) = C_O(\vec{X})$. In plain English, the QBF attempts finds an assignment to the key inputs $\vec{K}$ such that for all values of the primary inputs $\vec{X}$, the functions C and C_O are equal. *While we do have relatively efficient algorithms to solve this particular type of QBF [10], the formulation itself is moot. The attacker does not have access to the unencrypted IC's gate-level netlist and cannot construct the formula* $C_O(\vec{X})$.

The attacker can only observe the outputs for a small set of input patterns on the activated IC and must somehow determine the key values from these observations. Given a set of input/output observations, a SAT solver can determine a key value that is consistent with *these* observations. But this key value may not be correct. For example, suppose we make the observation $y = 1$ when $a = b = c = 1$ in Figure 2(b). Given this information, a SAT solver may assign $k_1 = 0$ since this value is consistent with the above observation. Even if we provide additional observations to the solver, *e.g.*, $(a, b, c, y) = (1, 1, 0, 1)$ and $(a, b, c, y) = (0, 1, 1, 1)$, $k_1 = 0$ is still a valid solution. *While we are guaranteed that the solver returns a key value that yields the correct output for the input patterns observed thus far, there may be other input patterns for which this key value produces incorrect output.* Furthermore, even if the SAT solver returns the correct key, verifying its correctness seemingly requires evaluation of outputs for all possible input patterns (2^M in a circuit with M inputs). Clearly, it is impractical to apply these many input patterns and so a simple SAT formulation is not enough.

B. Contributions of This Paper

The first contribution of this paper is a SAT-based algorithm that determines key values. The algorithm iteratively finds special input patterns, called *distinguishing inputs*, which rule out equivalence classes of keys. *The algorithm can determine when it has found a correct key value and can prove that this key is guaranteed to be consistent with all possible input/output observations.* It is agnostic to the algorithm used for logic encryption and requires a relatively small number of input/output observations from an activated IC. To the best of our knowledge, this is the first attack that in principle guarantees the decryption of a circuit encrypted with any combinational logic encryption scheme.[3]

A second contribution of this paper is a comprehensive evaluation of our attack algorithm on six proposed schemes for logic encryption [1, 8, 17–19]. To the best of our knowledge, ours is the first direct head-to-head comparison of the strength of these different logic encryption proposals. We find that all of these are vulnerable to our attack algorithm. We examine 21 benchmark circuits, encrypted with 21 different combinations of logic encryption algorithms and parameter values. Of these 441 encrypted circuits, our algorithm determines the correct key for 418 (95%) within 10 hours of compute-time. Among these, 397 encrypted circuits (90%) required 250 or fewer input/output observations for successful decryption.

A third contribution is a "partial-break" algorithm which can be used when the attack is not outright successful. This algorithm determines the correct value of *some* of the key inputs, thus reducing the effective security of logic encryption.

A final contribution is that we are making the attack tool and encrypted benchmarks available to the community [6]. We believe release of the tool will help the community develop even more secure logic encryption algorithms.

II. Description of Attack Algorithm

Before we describe the attack algorithm, let us formalize the notation and circuit model used in the rest of this paper.

A. Notation and Formal Model

Assume the encrypted combinational logic circuit[4] is represented as the relation $C(\vec{X}, \vec{K}, \vec{Y}) \subseteq \mathbb{B}^{M+L+N}$.[5] $\mathbb{B} = \{0, 1\}$ is the Boolean domain. The vector $\vec{X} \in \mathbb{B}^M$ represents the M primary inputs of the circuit. The vector $\vec{K} \in \mathbb{B}^L$ represents the L key inputs, while $\vec{Y} \in \mathbb{B}^N$ represents the N primary outputs. $C(\vec{X}, \vec{K}, \vec{Y})$ is equivalent to the input/output relation

[3] In practice, an implementation may run into scalability limitations if the encrypted circuit requires a very large number of iterations for decryption.

[4] This paper considers only combinational circuits. Sequential circuits can be treated as combinational circuits where the flip-flop inputs and outputs are treated as primary outputs and inputs respectively. We assume all flip-flops in a sequential design can be accessed through the scan chain, which is likely to be true for prevalent design-for-testability (DFT) [3] solutions. The attacker can use the scan-chain to read/write the values of all flip-flops in the design.

[5] Note C is actually the characteristic function of the relation. In this paper we use the relation and its characteristic function interchangeably. We choose to use Boolean relations rather than Boolean functions to represent the input/output behavior of the circuit because relations map directly to the CNF encoding used in modern SAT solvers.

of the unencrypted circuit when $\vec{K}$ is set to a correct key value, which may not be unique. $C_O(\vec{X}, \vec{Y}) \subseteq \mathbb{B}^{M+N}$ is the input/output relation for the unencrypted circuit. The attacker cannot construct the relation C_O but *can* apply input patterns and observe the outputs on an activated IC. We model this through the black-box function $eval : \mathbb{B}^M \mapsto \mathbb{B}^N$. For an input vector $\vec{\mathtt{X}}_1$, $eval(\vec{\mathtt{X}}_1) = \vec{\mathtt{Y}}_1$ if and only if $\vec{\mathtt{Y}}_1$ would be the output obtained when the input pattern $\vec{\mathtt{X}}_1$ is applied on an activated IC: $eval(\vec{\mathtt{X}}_1) = \vec{\mathtt{Y}}_1 \iff C_O(\vec{\mathtt{X}}_1, \vec{\mathtt{Y}}_1)$.

B. Problem Statement

The attacker's goal is to find $\vec{K} = \vec{\mathtt{K}_C}$ such that for all inputs $\vec{X}$: $C(\vec{X}, \vec{\mathtt{K}_C}, \vec{Y}) \iff eval(\vec{X}) = \vec{Y}$. This is equivalent to solving the QBF: $\exists \vec{K}\ \forall \vec{X}\ C(\vec{X}, \vec{K}, \vec{Y}) \wedge C_O(\vec{X}, \vec{Y})$, but of course, the attacker cannot construct the formula for C_O. For a practical attack, $eval$ can be evaluated only on a small number of input patterns, *i.e.*, only small number of input/output observations can be made on the activated IC.

C. Algorithm Overview

Given a set of input vectors $\vec{\mathtt{X}}_1, \vec{\mathtt{X}}_2, \ldots \vec{\mathtt{X}}_p$ and the corresponding output observations $\vec{\mathtt{Y}}_1, \vec{\mathtt{Y}}_2, \ldots, \vec{\mathtt{Y}}_p$, determining a key value that is consistent with these p observations is straightforward. One just needs to use a SAT solver to find a satisfying assignment to $\vec{K}$ in the formula $\wedge_{j=1}^{p} C(\vec{\mathtt{X}}_j, \vec{K}, \vec{\mathtt{Y}}_j)$. However, suppose we now make a new observation $eval(\vec{\mathtt{X}}_{p+1}) = \vec{\mathtt{Y}}_{p+1}$. We have no guarantee that a satisfying assignment to K for the formula $\wedge_{j=1}^{p} C(\vec{\mathtt{X}}_j, \vec{K}, \vec{\mathtt{Y}}_j)$ will also be a satisfying assignment to K for the formula $\wedge_{j=1}^{p+1} C(\vec{\mathtt{X}}_j, \vec{K}, \vec{\mathtt{Y}}_j)$. This is just a formal restatement of the observation we made in §I-A2. Although the SAT solver can find a key value that is consistent with the observations seen thus far, this value may not be correct. Even if the solver finds the correct key, verifying this apparently requires checking all 2^M input/output patterns.

Our algorithm is able to resolve these problems through two insights. First, instead of considering key values individually, let us consider *equivalence classes of keys*. Let us define two keys $\vec{K}_1$ and $\vec{K}_2$ to be equivalent, denoted as $\vec{K}_1 \equiv \vec{K}_2$, if and only if for each input value $\vec{X}_i$, the encrypted circuit produces the same output value $\vec{Y}_i$ for both keys $\vec{K}_1$ and $\vec{K}_2$. Precisely stated: $\vec{K}_1 \equiv \vec{K}_2$ iff $\forall \vec{X}_i : C(\vec{X}_i, \vec{K}_1, \vec{Y}_i) \wedge C(\vec{X}_i, \vec{K}_2, \vec{Y}_i)$.

The intuition is that instead of finding *the* correct key, we are looking for *a member of the equivalence class of keys* which produces the correct output for all input patterns. To "zero-in" on the correct equivalence class, we will iteratively rule out equivalence classes which produce the wrong output value for at least one input pattern. Given two key values $\vec{\mathtt{K}}_1$ and $\vec{\mathtt{K}}_2$, define the input pattern $\vec{\mathtt{X}}^d$ as a *distinguishing input pattern* if the encrypted circuit outputs different values $\vec{\mathtt{Y}}_1^d$ and $\vec{\mathtt{Y}}_2^d$ when the key inputs are set to $\vec{\mathtt{K}}_1$ and $\vec{\mathtt{K}}_2$ respectively. More precisely, $\vec{\mathtt{X}}^d$ is a distinguishing input pattern for $\vec{\mathtt{K}}_1$ and $\vec{\mathtt{K}}_2$ iff $C(\vec{\mathtt{X}}^d, \vec{\mathtt{K}}_1, \vec{\mathtt{Y}}_1^d) \wedge C(\vec{\mathtt{X}}^d, \vec{\mathtt{K}}_2, \vec{\mathtt{Y}}_2^d) \wedge (\vec{\mathtt{Y}}_1^d \neq \vec{\mathtt{Y}}_2^d)$.

The second insight is that if a distinguishing input pattern $\vec{\mathtt{X}}^d$ is found, then we can examine the output of the activated IC for input $\vec{\mathtt{X}}^d$ and use this to rule out one (or both) of $\vec{\mathtt{K}}_1$ and $\vec{\mathtt{K}}_2$ as not being in the equivalence class of correct keys.

Algorithm 1 Logic Decryption Algorithm

Function: $decrypt$.
Inputs: C and $eval$.
Output: $\vec{\mathtt{K}_C}$.

1: $i := 1$
2: $F_1 = C(\vec{X}, \vec{K}_1, \vec{Y}_1) \wedge C(\vec{X}, \vec{K}_2, \vec{Y}_2)$
3: **while** $sat[F_i \wedge (\vec{Y}_1 \neq \vec{Y}_2)]$ **do**
4: $\quad \vec{\mathtt{X}}_i^d := sat\_assignment_{\vec{X}}[F_i \wedge (\vec{Y}_1 \neq \vec{Y}_2)]$
5: $\quad \vec{\mathtt{Y}}_i^d := eval(\vec{\mathtt{X}}_i^d)$
6: $\quad F_{i+1} := F_i \wedge C(\vec{\mathtt{X}}_i^d, \vec{K}_1, \vec{\mathtt{Y}}_i^d) \wedge C(\vec{\mathtt{X}}_i^d, \vec{K}_2, \vec{\mathtt{Y}}_i^d)$
7: $\quad i := i + 1$
8: **end while**
9: $\vec{\mathtt{K}_C} := sat\_assignment_{\vec{K}_1}(F_i)$

This suggests Algorithm 1 which repeatedly finds distinguishing inputs (*line 4*) for some two keys $\vec{K}_1$ and $\vec{K}_2$, while asserting that the encrypted circuit must have outputs consistent with the input/output patterns observed thus far on the activated IC (*line 6*). The loop ends when no distinguishing inputs can be found. The correct key value is any assignment to $\vec{K}_1$ or $\vec{K}_2$ that satisfies the formula F_i (*line 9*).

Theorem: *Algorithm 1 terminates and upon termination* $\vec{\mathtt{K}_C}$ *is set to a value in the equivalence class of correct keys.*

Proof Sketch: Each iteration of the while loop rules out at least one incorrect equivalence class. This is because $(\vec{Y}_1 \neq \vec{Y}_2)$ and so at least one of the assignments to $\vec{K}_1$ or $\vec{K}_2$ must be incorrect. When the algorithm terminates, we know that no distinguishing inputs can be found, *i.e.*, this equivalence class cannot be refined further. Since circuit outputs are consistent with all the input/output patterns observed so far, we must be in the equivalence class of correct keys.

Consider the operation of Algorithm 1 on Figure 2(d). In the first iteration, the SAT-solver may generate $\vec{\mathtt{X}}_1^d$ as $(a, b, c) = (1, 0, 1)$. Evaluating the output yields $y = 1$. When F_2 is computed in line 6 of the algorithm, the solver deduces that $(k_1, k_2) = (0, 1)$ is not a valid key. The next iteration may have $\vec{\mathtt{X}}_2^d$ as $(a, b, c) = (0, 0, 0)$. Now we evaluate the output and get $y = 0$. The algorithm now terminates because the only key consistent with these observations is $(k_1, k_2) = (1, 1)$. In contrast to Algorithm 1, the fault-analysis attack is restricted to propagating key values to outputs. *It may seem that when no key input can be propagated to the output, an attacker needs to brute-force search the space of all possible keys. But as seen here, this is not true.* The algorithm is able to carefully select input patterns and infer key values even for encrypted circuits that are "immune" to the fault-analysis attack.

III. Partial-Break Algorithm

While Algorithm 1 is effective on most benchmark circuits we examine, it occasionally runs into scalability problems. This is because each iteration of the algorithm increases the size of the formula to be examined (*line 6*). Experimental results show this is not an issue in practice for most circuits because the algorithm converges very quickly. However, for

the few circuits that do not converge quickly, algorithms that can extract partial information about key values by analyzing subsets of the circuit (called "slices") are desirable.

A. Finding Suitable Circuit Slices

The first task here is finding suitable "slices." We define a slice of a circuit as a subset of the outputs of the circuit along with all the nodes in their transitive fanin cones. Slices required for the partial-break algorithm must satisfy the following properties.

1) For each output that is included in the slice, all the gates in its transitive fanin cone are also included in the slice.
2) The slice contains at least one key input such that all the outputs that are in the transitive fanout cone of the key are also included.
3) No more than the fraction P of the gates in the circuit are included in the slice. ($P = 0.3$ in our experiments.)

The first requirement defines a well-formed slice. The second condition ensures that at least one key input's "effect" on the circuit is fully-defined. The third requirement results in small slices. We find such slices by formulating the above requirements as constraints in an Integer Linear Program (ILP). The objective of the ILP is to maximize the number of key inputs included in the slice.

B. Extracting Key Values From Circuit Slices

At first glance, it may seem that we can run Algorithm 1 on a slice and then directly use the resultant key values in the larger circuit. But this is not possible because the key value returned by Algorithm 1 is a member of an equivalence class that is defined in terms of the subset of circuit outputs in the slice. But this equivalence class may be further refined by the outputs not part of the slice. And so this key value may be incorrect in the context of the full circuit. Instead, the insight behind the partial-break algorithm is to examine the input/output observations generated while running Algorithm 1 and determine whether these necessarily imply something about the values of the key variables in the full circuit.

Let us define C_{slice} as the input/output relation of the slice under consideration. Suppose we run Algorithm 1 on the slice and it converges after λ iterations. We now have λ input/output observations: $\left\{(\vec{\mathtt{X}}_1^d, \vec{\mathtt{Y}}_1^d), \ldots, (\vec{\mathtt{X}}_\lambda^d, \vec{\mathtt{Y}}_\lambda^d)\right\}$. Now consider the formula: $G = \wedge_{i=1}^{\lambda} C_{slice}(\vec{\mathtt{X}}_i^d, \vec{K}, \vec{\mathtt{Y}}_i^d)$. G encodes exactly the constraints on the key input variables imposed by the λ input/output observations. Now suppose $k_j \in \vec{K}$ is a key input, and that $G \implies k_j$. This means that k_j is a *backbone* of G, *i.e.*, all satisfying assignments to G must have $k_j = 1$, or equivalently for the input/output observations to hold, we must have $k_j = 1$. Therefore, k_j can be *generalized* to the full circuit. A symmetric argument applies when $G \implies \neg k_j$.

This leads to the complete procedure shown in Algorithm 2. Line 2 invokes Algorithm 1. However we do not use the return value $\vec{\mathtt{K}}_{\mathtt{C_{slice}}}$ but instead examine the input/output observations $\left\{(\vec{\mathtt{X}}_1^d, \vec{\mathtt{Y}}_1^d), \ldots, (\vec{\mathtt{X}}_\lambda^d, \vec{\mathtt{Y}}_\lambda^d)\right\}$. We use the probing algorithm from [24] to find backbones.

Algorithm 2 Slice Analysis Algorithm

Function: $partialBreak$.
Inputs: C and $eval$.
Output: $\vec{\mathtt{K}}_{\mathtt{C}}$.

1: $C_{slice} := findSliceUsingILP(C)$
2: $\left\{(\vec{\mathtt{X}}_1^d, \vec{\mathtt{Y}}_1^d), \ldots, (\vec{\mathtt{X}}_\lambda^d, \vec{\mathtt{Y}}_\lambda^d)\right\} := decrypt(C_{slice}, eval)$
3: $G := \wedge_{i=1}^{\lambda} C(\vec{\mathtt{X}}_i^d, \vec{K}, \vec{\mathtt{Y}}_i^d)$
4: $\vec{\mathtt{K}}_C := findBackbones(G)$

IV. Evaluation

Our attack algorithms apply to all combinational logic encryption schemes. Therefore, we evaluated the attack by implementing six state-of-the-art combinational logic encryption algorithms in our framework. EPIC [19] is the orginal proposal for logic encryption and it inserts XOR and XNOR gates at randomly chosen locations in the IC. We refer to this as "RND" in the results. Baumgarten et al. [1] ("DTC'10/LUT") insert a set of lookup tables (LUTs) to the IC such that every path from an input to an output goes through a lookup table. The values stored in the lookup tables are effectively the key inputs. Rajendran et al. [17] ("DAC'12") propose an algorithm to insert XOR/XNOR gates at carefully chosen locations in the circuit so that the encrypted circuit is not vulnerable to the fault-analysis attack. Rajendran et al. [18] also propose two algorithms that insert XOR ("ToC'13/xor") and MUX ("ToC'13/mux") gates at locations which maximize the Hamming distance between correct and incorrect outputs. Dupuis et al. [8] ("IOLTS'14") propose an encryption scheme which attempts to minimize low-controllability locations in the circuit by inserting AND and OR gates. When performing the encryption for ToC'13/mux and IOLTS'14, we estimated signal probabilities using the algorithm from [15].

A. Methodology

For the algorithms IOLTS'14, ToC'13/mux, ToC'13/xor, RND, and DAC'12, we generated four sets of encrypted circuits corresponding to an area overhead of 5%, 10%, 25% and 50% for logic encryption for each benchmark circuit. We believe an area overhead of 5% is a realistic budget for logic encryption, 10% may be acceptable for sensitive designs, and the values of 25% and 50% overhead are unrealistic but help us understand the limits of our attack. DTC'10/LUT does not describe a method to "tune" the overhead due to logic encryption. Hence we generated only one version of the encrypted circuits for this algorithm. In total, we generated 21 encrypted circuits from each benchmark circuit. When generating the benchmarks, we only considered the area overhead and did not consider the impact on timing, which may rule out certain encryptions. As a result, our evaluation is a pessimistic estimate of the strength of the attack algorithm and provides an upper bound on the security of the logic encryption algorithms.

Our benchmark circuits are listed in Table I. They consist of the ISCAS'85 benchmarks and the combinational circuits from the Microelectronics Center of North Carolina (MCNC)

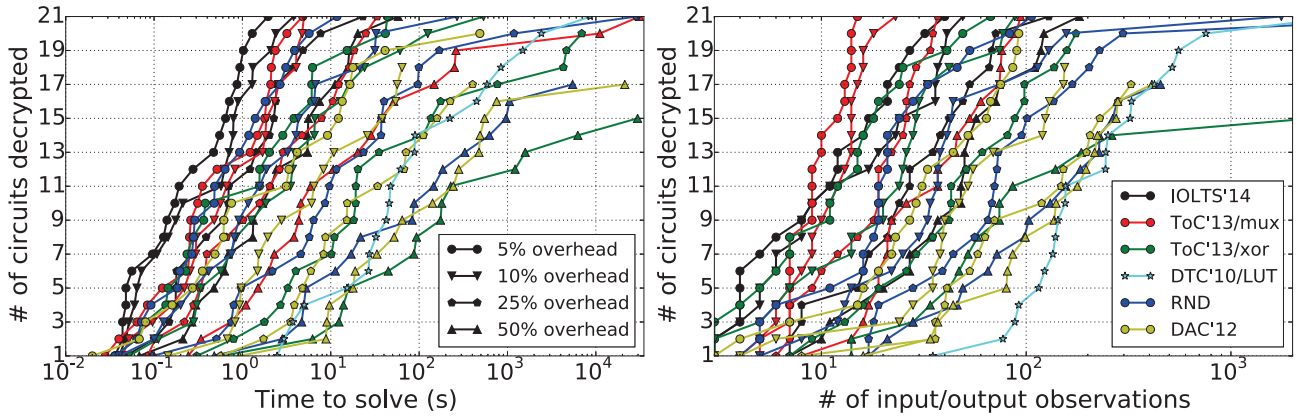


Fig. 3: Distributions of execution time (left) and number of input/output observations (right). Legend for both graphs is the same, each color represents a different encryption algorithm while the marker symbols show different area overheads.

(a) ISCAS'85 circuits

Circuit	#in	#out	#gates
c1355	41	32	546
c17	5	2	6
c1908	33	25	880
c2670	233	140	1193
c3540	50	22	1669
c432	36	7	160
c499	41	32	202
c5315	178	123	2307
c6288	32	32	2416
c7552	207	108	3512
c880	60	26	383

(b) MCNC circuits

Circuit	#in	#out	#gates
apex2	39	3	610
apex4	10	19	5360
dalu	75	16	2298
des	256	245	6473
ex1010	10	10	5066
ex5	8	63	1055
i4	192	6	338
i7	199	67	1315
i8	133	81	2464
i9	88	63	1035
k2	46	45	1815
seq	41	35	3519

TABLE I: Benchmark circuits used in the evaluation.

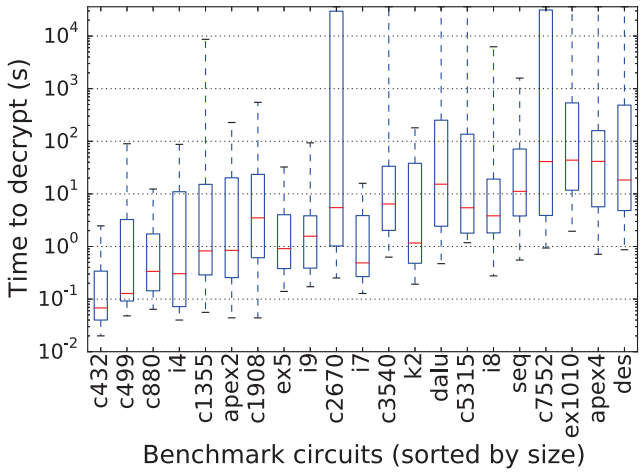


Fig. 4: Box plot of the runtime of Algorithm 1.

benchmark suite. Recall that our methodology applies to combinational circuits. These benchmarks continue to be reflective of the size of individual combinational logic circuits.

We exclude c17 and c6288 from the evaluation, leaving us with 21 benchmark circuits. c17 is trivial to decrypt and uninteresting. c6288 is a multiplier, and multipliers are challenging for SAT solvers in all contexts, not just logic encryption [5]. Since 21 encrypted circuits are generated from each benchmark circuit, our evaluation examined a total of $21 \times 21 = 441$ encrypted circuits.

We implemented Algorithms 1 and 2 in a C++ tool. We used the Lingeling [2] SAT solver and version 12.5 of the CPLEX ILP solver. Source code, binaries and benchmark circuits are available at [6]. Experiments were run on an Intel Xeon E31320 CPU with 32 GB of RAM. The execution time-limit was 10 hours (3.6×10^4 seconds).

B. Results

A summary of the results obtained using Algorithm 1 is shown as a "cactus plot" in Figure 3. For the plot on the left, the x-axis is in log-scale and shows the execution time of the algorithm in seconds. The y-axis is the number of benchmark circuits decrypted within these many seconds. The plot on the right shows number of the input/output observations on the x-axis (also in log-scale) and the y-axis is the number of circuits decrypted with these many or fewer observations. The lines show combinations of different logic encryption algorithms and area overheads for encryption. Each color represents a specific encryption algorithm while the different marker symbols show different area overheads for encryption.

Consider an area overhead of 5%. We assert this is a realistic budget for logic encryption. In this scenario, all 21 circuits are decrypted for IOLTS'14, ToC'13/mux, ToC'13/xor and RND. For DAC'12, the only circuit we are unable to decrypt is c2670. In all successful attacks, the execution time was less than 8 minutes and fewer than 104 input/output observations were required. All circuits for DTC'10/LUT were also decrypted; the most challenging circuit needed a runtime of about 2.5 hours and 3594 input/output observations.

Even when allowing for a budget of 50% for the area overhead, we are able to decrypt all 21 circuits encrypted with IOLTS'14 and ToC'13/mux. We are able to decrypt 17

circuits encrypted with DAC'12 and RND and 15 circuits encrypted with ToC'13/xor. The maximum execution time for a successful attack was about 8 hours and the maximum number of observations needed was 15981. In total, 418 (95%) of the 441 encrypted circuits were decrypted. 397 of these (90%) needed fewer than 250 input/output observations. These numbers suggest that the attack is feasible and easy to perform on all encryption schemes and almost all encrypted circuits.

Insertion of AND, OR and MUX gates as in IOLTS'14 and ToC'13/mux is less secure than the insertion of XOR/XNOR gates. XOR/XNOR gates result in clauses that are harder to satisfy with the DPLL algorithm used in modern SAT solvers [21]. The targeted placement of XOR/XNOR gates done by DAC'12 and ToC'13/xor is more secure than RND.

Figure 4 shows a "box plot" of the distribution of the time taken to decrypt the different encrypted versions of each benchmark circuit using Algorithm 1. The benchmark circuits are shown on the x-axis and are sorted by size. The leftmost circuit is the smallest and the rightmost is the largest. For each circuit, we consider the vector consisting of the time-to-decrypt values for each of its 21 different encrypted versions. If a particular circuit could not be decrypted, its time-to-decrypt value is set to the time limit of 10 hours (3.6×10^4 seconds). The box shows the 25%-75% percentile distribution of this vector, the red horizontal line shows the median value of this vector, and the two black lines show the minimum and maximum values. We expect that larger circuits will take more time to decrypt, and we can see from the plot that this is largely true. But we also observe that the strength of the encryption has a significant impact on the runtime of Algorithm 1. For weak encryption schemes, or when the number of key inputs is small, Algorithm 1 completes execution in just a few seconds for even the largest circuits in our benchmark suite. And conversely, when the number key inputs is very large and if these key inputs are placed strategically, the decryption of small circuits can also be challenging.

C. Partial-Break Algorithm Results

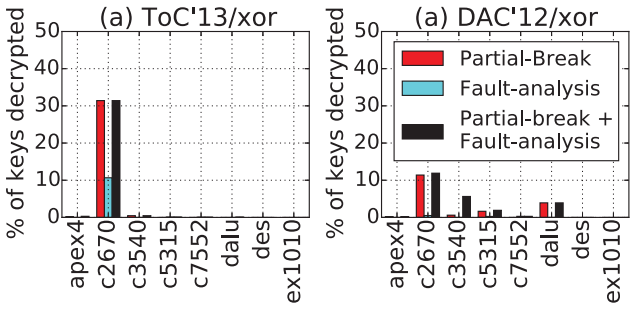


Fig. 5: Percentage of keys decrypted with the partial-break algorithm and the fault-analysis attack. All circuits are encrypted with 50% area overhead for encryption.

Figure 5 shows the results of the partial-break algorithm from Section III. We consider the two most challenging encryption schemes: ToC'13/mux and DAC'12, with 50% area overhead. Eight circuits are shown; these are the circuits which could not be decrypted by Algorithm 1 for one or both of these encryption algorithms. Three sets of results are shown, the partial-break algorithm, the fault-analysis algorithm from [17] and a hybrid algorithm which runs the fault-analysis attack and partial-break algorithm iteratively. The y-axis in the figure shows the percentage of keys that were decrypted. We see that all algorithms find these circuits very challenging. In particular the fault-analysis algorithm [17] is able to decrypt very few keys. The partial-break algorithm is based on backbones, so it will succeed only if some of the key input values are unique. But when so many XOR/XNOR gates are inserted, the keys are unlikely to be unique. So it is unsuccessful in many instances. Despite this, it is able to decrypt a significant percentage of keys in some benchmarks, including over 30% in c2670.

V. Discussion

A. Challenging Circuits for Logic Decryption

Surprisingly, the relatively small benchmark circuit c2670 turns out to be challenging for logic decryption algorithms. We investigated the reason for this and found that the benchmark contains an "and-tree," which is a tree of AND gates that computes the function $y = \wedge_{i=1}^{N} x_i$. Now suppose the encrypted circuit is $y^e = \wedge_{i=1}^{N} (x_i \oplus k_i)$. For this circuit, it is easy to see that each assignment to $k_1 \ldots k_N$ is a singleton equivalence class. Therefore the and-tree with N inputs encrypted as above has 2^N equivalence classes of keys. Moreover, any single input/output observation in which $y^e = 0$ rules out only one equivalence class. So if the SAT solver in Algorithm 1 chooses its distinguishing inputs from a uniform distribution, the expected number of observations required to decrypt the and-tree is $\approx 2^{N-1}$. The and-tree also has a "graceful degradation of encryption" property: if $y^e = \left(\wedge_{i=1}^{k} (x_i \oplus k_i) \right) \wedge \left(\wedge_{i=k+1}^{N} x_i \right)$, $k < N$, then we still have 2^k equivalence classes.

All of these observations explain why c2670 is hard to decrypt. They also suggest interesting directions for the design of provably-secure combinational logic encryption algorithms.

B. Related Work

1) Logic Encryption: This paper considered only combinational logic encryption schemes. Sequential logic encryption schemes have also been proposed [4, 14] but analyzing them is outside the scope of this work. Our attack can be extended to some sequential logic encryption schemes too. E.g., in ObfusFlow [4], the circuit is modified by inserting a finite state machine (FSM) whose outputs are XOR'd with some signals in circuit. Algorithms 1 and 2 can be used to infer these FSM output values. A model checker can then find a trace that takes the FSM to a state which outputs these values.

2) Partial Synthesis: The logic decryption algorithm presented in Section II is similar to the oracle guided program synthesis algorithm from Jha et al. [12], and the partial synthesis algorithm from Fujita et al. [9]. Jha et al. [12] present a program synthesis algorithm that can synthesize

small loop-free programs using a simulation oracle. Fujita et al. [9] propose an algorithm for partial synthesis in the scenario where parts of a circuit have been replaced by lookup tables. This is useful in implementing engineering change orders (ECOs), debugging logical bugs etc. Both algorithms are based on generating candidate solutions and then counterexamples to these candidates. Algorithm 1 directly generates the equivalent of counterexamples. Fujita et al. [9] use a slightly more complex formulation to generate the counterexamples, they encode the equality of outputs across two candidate solutions. We avoid this by directly asserting the output values as concrete Boolean values (see *line 6*).

VI. Conclusion

In this paper, we used SAT-based techniques to investigate the security of logic encryption. We presented a SAT-based attack on logic encryption that allows an attacker to infer the correct values of the key inputs using only a small number of input/output observations taken from an activated IC. We also presented a "partial-break" algorithm that allows an attacker to infer the values of some of the key inputs even when the full attack is not successful. We performed a thorough evaluation of these attacks by investigating the security of six different proposals for logic encryption. We found all these proposals are vulnerable to our attacks. Out of 441 encrypted circuits we examined, we could decrypt 418 (95%). Among these 397 circuits (90%) required only 250 or fewer input/output observations. We discussed the strengths and weaknesses of our attacks and suggested some directions towards provably-secure logic encryption algorithms.

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] A. Biere. Lingeling, Plingeling and Treengeling. In A. Balint, A. Belov, M. Heule, and M. Järvisalo, editors, *Proceedings of the SAT Competition*, 2013.

[3] M. Bushnell and V. Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Springer Publishing Company, 2013.

[4] R.S. Chakraborty and S. Bhunia. Hardware Protection and Authentication Through Netlist Level Obfuscation. In *IEEE/ACM International Conference on Computer-Aided Design*, 2008.

[5] S. A. Cook and D. G. Mitchell. Finding Hard Instances of the Satisfiability Problem: A Survey. In *Proceedings of the DIMACS Workshop on Satisfiability Problems*, 1997.

[6] Decryption tool binaries and benchmark circuits. https://bitbucket.org/spramod/host15-logic-encryption, 2015.

[7] Defense Science Board Task Force on High Performance Microchip Supply. http://www.acq.osd.mil/dsb/reports/ADA435563.pdf, 2005.

[8] S. Dupuis, P.-S. Ba, G. Di 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*, 2014.

[9] M. Fujita, S. Jo, S. Ono, and T. Matsumoto. Partial synthesis through sampling with and without specification. In *IEEE/ACM International Conference on Computer-Aided Design*, 2013.

[10] A. Gascon, P. Subramanyan, B. Dutertre, A. Tiwari, D. Jovanovic, and S. Malik. Template-based circuit understanding. In *Proceedings of Formal Methods in Computer-Aided Design*, 2014.

[11] IHS Technology Press Release: Top 5 most counterfeited parts represent a $169 billion potential challenge for global semiconductor industry. https://technology.ihs.com/405654/top-5-most-counterfeited-parts-represent-a-169-billion-potential-challenge-for-global-semiconductor-market, 2012.

[12] S. Jha, S. Gulwani, S. A. Seshia, and A. Tiwari. Oracle-guided component-based program synthesis. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, 2010.

[13] F. Koushanfar. Hardware metering: A survey. In Mohammad Tehranipoor and Cliff Wang, editors, *Introduction to Hardware Security and Trust*. Springer New York, 2012.

[14] F. Koushanfar. Provably Secure Active IC Metering Techniques for Piracy Avoidance and Digital Rights Management. *IEEE Transactions on Information Forensics and Security*, 7(1), Feb 2012.

[15] B. Krishnamurthy and I.G. Tollis. Improved Techniques for Estimating Signal Probabilities. *IEEE Transactions on Computers*, 38(7), Jul 1989.

[16] M. Pecht and S. Tiku. Bogus! Electronic manufacturing and consumers confront a rising tide of counterfeit electronics. *IEEE Spectrum*, May 2006.

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

[18] 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), Feb 2015.

[19] J. A. Roy, F. Koushanfar, and I. L. Markov. EPIC: Ending Piracy of Integrated Circuits. In *Proceedings of Design, Automation and Test in Europe*, 2008.

[20] 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.

[21] M. Soos, K. Nohl, and C. Castelluccia. Extending SAT Solvers to Cryptographic Problems. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, 2009.

[22] R. Torrance and D. James. The State-of-the-Art in IC Reverse Engineering. In *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems*, 2009.

[23] J. Villasenor and M. Tehranipoor. The Hidden Dangers of Chop-Shop Electronics. *IEEE Spectrum*, Sep 2013.

[24] C. S. Zhu, G. Weissenbacher, D. Sethi, and S. Malik. SAT-based Techniques for Determining Backbones for Post-Silicon Fault Localisation. In *Proceedings of the IEEE High Level Design Validation and Test Workshop*, 2011.