Modeling Techniques for Logic Locking

Joseph Sweeney
joesweeney@cmu.edu
Department of Electrical and Computer Engineering,
Carnegie Mellon University
Pittsburgh, PA

Marijn J. H. Heule
marijn@cmu.edu
Department of Computer Science,
Carnegie Mellon University
Pittsburgh, PA

Lawrence Pileggi
pileggi@cmu.edu
Department of Electrical and Computer Engineering,
Carnegie Mellon University
Pittsburgh, PA

ABSTRACT

Logic locking is a method to prevent intellectual property (IP) piracy. However, under a reasonable attack model, SAT-based methods have proven to be powerful in obtaining the secret key. In response, many locking techniques have been developed to specifically resist this form of attack. In this paper, we demonstrate two SAT modeling techniques that can provide many orders of magnitude speed up in discovering the correct key. Specifically, we consider relaxed encodings and symmetry breaking. To demonstrate their impact, we model and attack a state-of-the-art logic locking technique, Full-Lock. We show that circuits previously unbreakable within 15 days of run time can be solved in seconds. Consequently, in assessing the strength of any given locking, it is imperative that these modeling techniques be considered. To remedy this vulnerability in the considered locking technique, we demonstrate an extended version, logic-enhanced Banyan locking, that is resistant to our proposed modeling techniques.

CCS CONCEPTS

• **Security and privacy → Hardware reverse engineering**; *Hardware security implementation*; *Hardware attacks and countermeasures.*

KEYWORDS

logic locking, IP piracy, satisfiability, miter-based SAT attack

1 INTRODUCTION

Due to prohibitively high research and development costs, only a few foundries are manufacturing integrated circuits (ICs) in advanced technology nodes. Consequently, many IC companies tend to operate fabless, relying on untrusted foundries to manufacture their designs. Once a circuit is sent for fabrication, the foundry gains full visibility of the design in netlist form with minimal effort, allowing IP theft. This threat undermines the significant cost associated with developing digital circuits and is a growing concern in the IC industry [9].

To combat IP theft, a variety of logic locking techniques have been developed. These techniques add programmable elements to the logic of a digital IC. When programmed incorrectly, the elements disrupt the circuit, obfuscating the true functionality. The key, which correctly programs the elements, is stored in an on-chip, tamper-proof memory. This key is set post-manufacture, so the correct functionality is never revealed to the untrusted foundry.

Early examples of logic locking techniques inserted keyed exclusive-or (XOR) and multiplexer (MUX) gates to corrupt the next-state logic [12, 17]. Unfortunately, these methods have been largely broken using a variety of attacks, the most successful of which are miter-based SAT attacks [21]. Researchers have attempted to increase the difficulty of the miter-based attack by inserting resistant logic blocks into the locked circuit.

These resistant locking techniques generally fall into two categories based on how they resist the miter-based SAT attack. The first group [24, 25, 28, 29] focuses on reducing the number of keys ruled out per attack iteration, significantly increasing the expected *number of iterations*. In practice however, these techniques are susceptible to removal attacks since the circuitry is typically traceable through properties such as signal probability or Boolean sensitivity [22, 26].

The second group [13, 18] tries to extend the *time per iteration*. This is done by adding SAT-hard instances into the circuit. These instances typically have many interdependent keys; a prototypical example is the lookup table (LUT) combined with configurable routing. The resulting locks resemble field-programmable gate arrays (FPGAs) embedded into the circuit and have been shown to be highly resistant to the current miter-based SAT attacks.

In this paper, we explore the use of two modeling techniques targeting locked circuits from this second group. These techniques are shown to be powerful tools in revealing the key, dramatically reducing attack run time. Specifically, the contributions of this work are the following:

- Proposal of relaxed encoding and symmetry breaking as modeling techniques for attacking locked circuits
- Demonstration of impact of the modeling techniques in attacking a state-of-the-art scheme, Full-Lock
- Logic-enhanced Banyan locking, an improved version of Full-Lock, not susceptible to these new attack techniques

2 BACKGROUND

2.1 Attack Model

In the characterization of the security of a locking technique, an attack model is used to specify assumptions regarding the adversary's ability. In this paper, as in all the aforementioned locking techniques, it is assumed that the adversary has access to two artifacts: the locked circuit's netlist and an unlocked version of the

ICCAD '20, November 2–5, 2020, Virtual Event, USA
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8026-3/20/11.
https://doi.org/10.1145/3400302.3415668

circuit. The unlocked circuit has the correct key set in its tamper-proof memory, affording the attacker black-box access, commonly referred to as an oracle. These artifacts correspond to the access a foundry is likely to have when manufacturing a commercial design. The netlist can be easily reversed engineered from the design data and the unlocked circuit can be obtained on the open market. It is also assumed that the adversary has access to the unlocked design's scan chains. While additional side-channel techniques may augment an attacker, they are considered outside the scope of the paper.

In general, the problem being solved by the attacker is as follows. The attacker has access to an unlocked circuit containing a set of Boolean functions. Each function, $f : \{0,1\}^{n+k} \rightarrow \{0,1\}$, has n normal inputs and k unknown, fixed key inputs. The attacker also has knowledge of the structure of f. Using the unlocked circuit and knowledge of the structure, the goal of the attacker is to obtain a functionally equivalent version of the circuit.

2.2 Propositional Satisfiability

A common approach to deal with hard combinatorial problems, such as finding the key of locked circuits, is to encode them into propositional logic and to solve the resulting propositional formulas with a satisfiability (SAT) solver. The performance of SAT solvers improved significantly in the last two decades and they are used for many applications in hardware and software verification [4, 11]. In recent years, SAT solvers have also been successfully applied to various attacks, such as hash collisions [20] and mathematical challenges [14].

The typical encoding of the SAT problem is in the conjunctive normal form (CNF). This form consists of a set of clauses that must all be satisfied. Each clause is a disjunction of literals. A circuit can be encoded into propositional logic via the Tseitin transformation [23]. This transformation can take a circuit netlist and produce a set of clauses which, when collectively satisfied, will correspond to the original circuit's behavior.

The most successful class of SAT solvers are based on the conflict-driven clause learning (CDCL) algorithm [2, 16]. Briefly, CDCL solvers work by repeatedly selecting a variable through heuristics and assigning a value. Implications from the assignments are determined using a highly optimized process called unit propagation. If a conflict is found, a clause is added to the formula that rules out assignments causing the conflict. Then the solver non-chronologically backtracks based on the conflict and continues, repeating this process until a solution is found or the problem is found to be unsatisfiable.

2.3 Miter-Based SAT Attack

The above attack model enables the mounting of a more targeted, miter-based SAT attack. This attack uses the netlist and unlocked circuit to iteratively produce input-output (IO) relationships [21]. These relationships are used to rule out all keys that do not produce the same behavior, narrowing the space of possible circuit functionalities. The IO relationships are efficiently learned through a three-step procedure: **I.** First, a miter circuit [6] is used to determine an input that is guaranteed to rule out at least a single key. A miter circuit consists of two copies of the original circuit with the

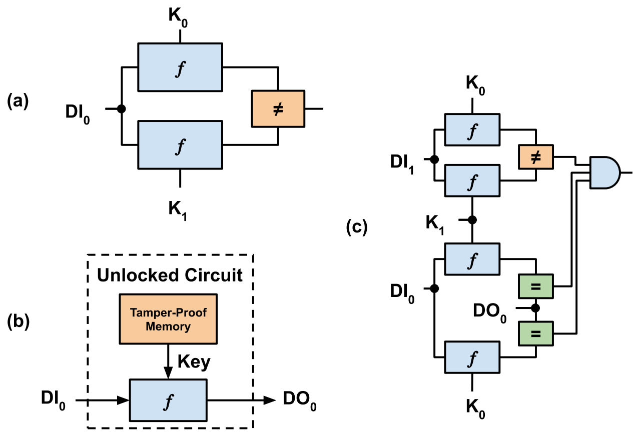

Figure 1: Miter-based SAT attack steps: (a) Miter circuit construction, (b) Unlocked (oracle) circuit produces correct IO functionality (c) Addition of learned IO constraint to miter circuit

inputs tied together, the key inputs kept separate, and the outputs connected to comparators. A diagram of the connections is shown in Fig. 1a. Additional key constraints, such as timing and loop breaking, can be conjuncted with the miter output. A SAT solver is used to find a setting of the shared input (I) and key inputs (K_0, K_1) such that the output of the miter circuit is logic 1. By construction, the solution to the SAT problem will have two different keys that, at that input value, disagree on the output value. The shared input value found by the solver is termed a differentiating input (DI). **II.** Next, as depicted in Fig. 1b, the learned DI is applied to the oracle circuit to determine the differentiating output (DO), forming an input-output (IO) pair which the correct key must respect; any key that does not conform to this IO pair is incorrect. **III.** Finally, as shown in Fig. 1c, the IO pair is added as a constraint to the miter circuit for the next iteration. Now, any keys that satisfy the miter circuit will also satisfy the learned IO relationship. While each relationship is guaranteed to rule out at least one key, in practice, a larger portion of the key space is ruled out due to overlapping key functionalities at a given input. These steps repeat, adding more constraints until the miter circuit is unsatisfiable. At this point, any key that respects all learned IO relationships will be a functionally correct key.

2.4 Full-Lock

Full-Lock is a logic locking technique specifically developed to be resistant to the miter-based SAT attack [13] via increasing the execution time of each iteration. This is done by integrating SAT-hard logic into the circuit using a combination of routing obfuscation and look-up tables (LUT). The added logic is highly symmetric with many keys mapping to the same functionality. Symmetry is known to be difficult for SAT solvers, trapping the algorithm by spending time exploring solutions that are isomorphic [7]. Furthermore, unit propagation of the circuit is hindered as each configuration depends on many keys: in order to determine the output of the Full-Lock circuitry, most keys must be assigned. Finally, the obfuscation is

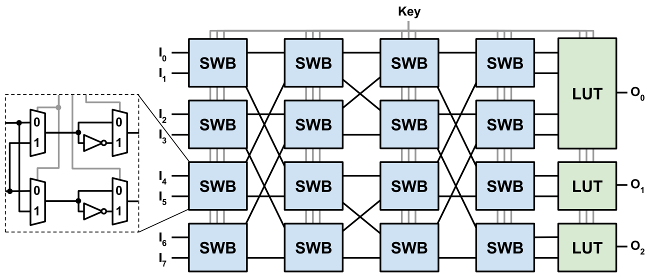

Figure 2: Full-Lock diagram. Each LUT replaces a gate from the original circuit; the switch boxes permute and invert their input signals.

parameterized such that locking scheme's clauses to variables ratio is close to 4.26, the phase-transition density for uniform random 3-SAT (SAT instances with exactly 3 variables per clause) [8]. Intuitively, instances with a higher ratio are over-constrained making contradictions easier to find and those with a lower ratio are under-constrained with potentially many satisfying solutions. While the instances produced by Full-Lock are not uniform random 3-SAT, and therefore likely have a different optimal ratio, the locking still produces hard SAT instances.

Full-Lock utilizes configurable routing and LUTs to obfuscate a set of gates and their corresponding input connections. The configurable routing is implemented with Banyan networks, a class of logarithmic networks, that permutes connections based on a key [13]. The network is made up of a series of 2-input switch boxes which connect the inputs to the outputs, either directly passing through or switched. Additionally, Full-Lock adds the ability to invert the polarity of the signals in each switch box. Diagrams of the switch boxes and overall network are shown in Fig. 2. The specific Banyan network configuration used has $2 * log_2(N) - 2$ stages where N is the network's input width (equal to the number of permuted lines). The Banyan network is almost non-blocking, meaning that almost all input to output connection permutations are possible.

The locking procedure is as follows. A set of gates with the desired number of total inputs is randomly selected from the circuit. A Banyan network is inserted into the circuit. The nets fanning into the selected gates are randomly inverted and connected to the network's inputs. The selected gates are replaced with LUTs of appropriate input size. The outputs of the network are connected the LUT inputs such that under the correct key, each LUT will receive the original inputs with proper polarity. The key to the circuit is thus the concatenation of the LUT and network configuration bits. The random selection of gates opens the possibility for combinational loops to be formed in the circuit. This has no impact on the circuit when the correct key is applied as all feedback paths will be broken. However, if not ruled out, these loops will corrupt the miter-based SAT attack. Several methods of building loop-breaking key constraints have been developed to re-enable the attack [19, 32].

As is, this locking method appears resistant to the miter-based SAT attack. The authors of the original work ran the attack for 15 days without termination on instances with 32 circuit lines permuted. Additionally, the authors considered a removal attack. The added circuitry is easily identifiable, even after synthesis due to the key lines and regular structure. Despite this, Full Lock is also resistant to a removal attack as the selected gates have been replaced with LUTs and the correct interconnections and polarities of their inputs are unknown.

3 MODELING TECHNIQUES

Critical to the performance of SAT solvers is the encoding of the problem. Many problems become hard for SAT due to a poor encoding. Often the best encoding is found after trying several different strategies [5]. Thus, when assessing the security of a locking technique, the encoding used can drastically influence the results. In this section, we describe two modeling techniques that are widely applicable to logic locking. We demonstrate the application of each technique to the example locking method, Full-Lock.

3.1 Relaxed Models

Each iteration of the miter-based SAT attack satisfies the miter circuit while respecting the system model. The system model captures the potential behaviors of the locked circuit under different keys and is encoded into propositional logic allowing the SAT solver to generate meaningful inputs. However, the exact system model can be difficult to specify or too complex for SAT solvers to efficiently handle. Often, a close analog to the original behavior can be captured with a much simpler encoding. Substituting the system model can allow significant decreases in attack time, sacrificing precision for reduced complexity.

Several factors must be considered when building a relaxed model for a locked circuit. *First*, the model's variables do not all need to directly map to system's logic. In fact, the only requirement on the variable mapping is that the inputs and outputs remain directly mapped between the encoding and original system model so that the produced DIs can be run on the oracle and the resulting DI-DO pair can be added to the miter. *Next*, the relaxed model must be able to produce a super-set of the input-output relationships under all key values. Perhaps counter intuitively, specifying a super-set of behaviors can be easier than the exact set. *Finally*, while the key variables do not need to be directly encoded, there must be a mapping from the relaxed model back to a valid key configuration of the original system.

An example of relaxed modeling is seen in TimingSAT [15], an attack methodology for TimingCamouflage [31]. TimingCamouflage substitutes flip-flops with combinational logic delays. This disrupts a naive attack strategy because a reverse engineered netlist will be missing flip-flops that correspond to the correct functionality. It is assumed that to obtain the system functionality, an attacker must meticulously time the circuit and check all possible paths for potential combination logic delays replacing a flip-flop. However, TimingSAT simply substitutes a relaxed model, overestimating the possible locations where a combinational delay may be used as a flip-flop. In each potential flip-flop location, a MUX is inserted selecting between a flip-flop or wire. The functionality is then determined using the standard miter-based SAT attack, solving for the proper MUX settings.

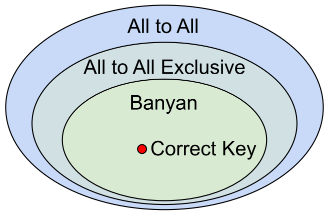

Figure 3: Relaxed models for Banyan network

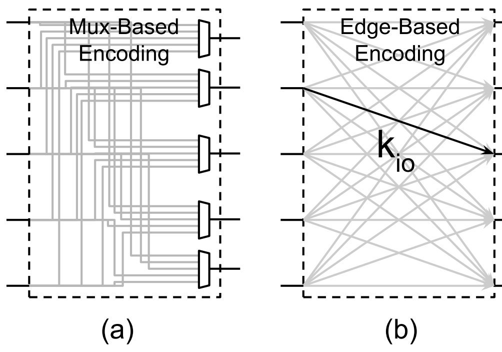

Figure 4: Edge-based and MUX-based encoding schemes for the all-to-all model

A relaxed encoding can also be used to remove key interdependence. Often the functionality of a locked circuit will depend on a large portion of the keys. To determine the output for a given input, the SAT solver must branch on many of the key variables. However, in some cases the functionality can be separated from the key variables. This allows the functionality to be selected without assigning all keys. An analogous example is encoding integers. The typical circuit for handling integers is representing them with binary numbers, however, to select an integer value all variables representing the binary number's bits must be assigned. For SAT solvers, an often more efficient strategy is one-hot encoding. Here, a value can be directly assigned by setting a single variable true (and unit propagating the others to false). In a similar sense a circuit functionality can be decoupled from the key bits, directly selecting the functionality rather than assigning all key bits.

Using this relaxed encoding strategy, we consider our example technique, Full-Lock. As previously established, the Banyan network is a SAT-hard circuit due to its large amounts of symmetry, key interdependence, and poor unit propagation behavior. Despite its complexity, the functionality is very simple: the outputs of the network are a permutation of the inputs. Due to the structure of the network, some permutations are prohibited, and others can be selected by multiple key settings. If we relax the encoding of the network, allowing the prohibited permutations in our model, we can significantly reduce the complexity.

We consider two relaxed models in place of the Banyan network: *all-to-all*, wherein every input can be routed to every output, and *all-to-all exclusive*, which additionally restricts an input to be routed to only a single output. A diagram of these functionalities is shown in Fig. 3. The correct key is in the set of functionalities that the Banyan network allows, which is a subset of the all-to-all exclusive model functionalities, and in turn, the all-to-all model functionalities.

From a circuit designer's perspective, the natural way to encode all-to-all functionality uses an N-to-1 MUX for each output, similar to the structure depicted in Fig. 4a. This can be easily specified in a high-level language such as verilog, then synthesized to a gate-level representation. The Banyan network in Full-Lock can then be substituted for these gates. Just as in the typical miter-based SAT attack, the circuit can then be encoded into SAT via the Tseitin transformation. The all-to-all exclusive encoding can be formed in the same fashion, adding circuitry to ensure that the select bits of each MUX are different.

We also consider an edge-based strategy in which a key variable, k_{io}, is created for each possible input to output connection. A diagram of this encoding is depicted in 4b. The CNF of the encoding is shown below where x_j is a variable representing a net j, I is the set of nets fanning into the Banyan network, and O is the set of nets in its fanout.

$$\bigwedge_{i \in I, o \in O} k_{io} \rightarrow (x_i \leftrightarrow x_o) \tag{1}$$

To ensure proper functional behavior we must also enforce that each network output is only connected to one input. This can be done using a cardinality encoding over the same variables as below:

$$\bigwedge_{o \in O} ExactlyOne(\{k_{io} : i \in I\}) \tag{2}$$

The edge-based all-to-all exclusive encoding is created with the additional clauses:

$$\bigwedge_{i \in I} ExactlyOne(\{k_{io} : o \in O\}) \tag{3}$$

Running the miter-based SAT attack on these encodings will produce the correct mapping from the network inputs to outputs. Obtaining the corresponding key for the original system model can be done by finding a key that propagates the same paths in the Banyan network. Our models allow a greater function space, but with an encoding much more amenable to SAT solvers as we will see in section 5.1.

3.2 Symmetry Breaking

Another modeling technique that is not entirely exclusive from relaxed encodings, but can be applied on its own, is symmetry breaking. In the context of SAT, a symmetry is defined as a permutation of variable assignments which maps one solution onto another [1]. In the miter-based SAT attack, symmetry results from classes of keys producing the same circuit functionality. All equivalent keys will be equisatisifiable with respect to the miter circuit inputs. If symmetry exists in the locked circuit, the attack will waste time exploring isomorphic parts of the search space.

Symmetry breaking in the miter-based SAT attack context entails ruling out all but one key from each equivalence class. Ideally, this is done with minimal additional clauses being added to the problem, otherwise the additional problem complexity may outweigh any benefit. While not specifically labeled as symmetry breaking, this strategy has been utilized in the key-sensitization attack on Strong Logic Locking [27], wherein back-to-back key XOR gates are converted into a single XOR.

Table 1: 2-Input LUT Symmetries Under Permuted Inputs

K_0 $(I_1, I_0 = 0, 0)$	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
K_1 $(I_1, I_0 = 0, 1)$	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
K_2 $(I_1, I_0 = 1, 0)$	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
K_3 $(I_1, I_0 = 1, 1)$	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Several examples of symmetry are seen in Full-Lock. In the Banyan network, multiple keys produce the same permutations of the inputs on the outputs. Additionally, the keys which optionally invert the switch box outputs are highly symmetric: all configurations of these keys can be reduced to a single bit for each output specifying whether it is inverted. Our relaxed models of the network already remove these two symmetries. However, there remains a significant amount of symmetry in Full-Lock's LUTs.

By themselves, LUTs have no symmetric assignments, but when coupled with external circuitry, they can become highly symmetric. This property is good for Field Programmable Gate Arrays (FPGAs) wherein flexible configurations can help meet timing, power, and area constraints, however, this flexibility hinders the miter-based SAT attack as the same logical functionality can be specified many ways.

Full-Lock allows the LUT-inputs to be permuted which creates *LUT configuration, input permutation* pairs that are symmetric. In Table 1, we show all the symmetric configurations of a 2-Input LUT with the inputs permuted. Each group with more than one equivalence is highlighted in a different shade of blue. Within the highlighted groups, permuting the inputs allows a single LUT configuration to function equivalently to the others. Thus, only one LUT configuration per group is needed. In the 2-input LUT case, only 4 out of 16 configurations are eliminated, however, as the input width increases the number of symmetric configurations grows significantly. For a 4-input LUT, there is over an order of magnitude reduction in the number of remaining configurations.

Full-Lock's input permutation symmetry can be broken by enforcing an ordering on the inputs connected to each LUT. This ensures that for every combination of inputs routed to a LUT, only a single permutation is allowed, ruling out all unnecessary configurations. To create this ordering, we add a unary mapping of the key variables of our edge-based encoding. For each LUT, where s_{io} is an auxiliary variable representing the unary mapping for a key k_{io} and O_L is the ordered set of network outputs that connect the LUT, we add the clauses in Eq. 4 to our solver.

$$\bigwedge_{i \in I, o \in O_L} (k_{io} \rightarrow s_{io}) \wedge (k_{io} \rightarrow \neg s_{io+1}) \wedge (k_{io} \rightarrow s_{i+1o}) \wedge (s_{io} \rightarrow \neg s_{i+1o}) \tag{4}$$

4 LOGIC-ENHANCED BANYAN LOCKING

4.1 Overview

Based on our attack data in Section 5.1 and the results from the original Full-Lock work, it is clear the Banyan network structure creates an instance that is difficult for the miter-based SAT attack. The strengths of the network are the large number of cycles it can potentially create, the interdependence of keys, and the lack of intermediate outputs. However, with the proposed modeling

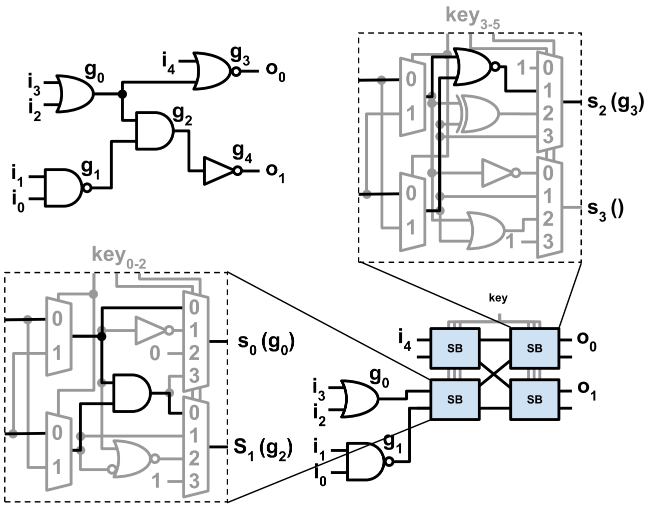

Figure 5: Diagram of circuit mapped to logic-enhanced Banyan network. The original circuit is shown top-left, the locked version bottom-right. The correct switch box function is highlighted in black, the decoy logic in gray.

techniques, we have exposed holes in the original formulation. Here, we describe a remedy based on breaking the assumptions of the modeling techniques through the addition of logic internal to the network.

Our locking technique, logic-enhanced Banyan locking, uses the same Banyan structure as Full-Lock, however, the functionality is extended beyond the simple invert and permute. This is achieved by moving logic from the locked circuit into the switch boxes of the Banyan network. In the original Full-Lock switch box, two key bits are used to optionally invert the lines passing through. Now, we use these two key bits to select one of four possible functions for each switch box output. One configuration produces the correct function, the others are randomly generated decoy functions of the switch box inputs.

A diagram of the new technique is depicted in Fig. 5. In this small example, a 4-input Banyan network is inserted. Using the switchbox outputs as reference points, gates from the original circuit are mapped to the Banyan network. Switch box outputs s_0, s_1, and s_2 respectively map to gates g_0, g_2, and g_3. We show the internal logic of two of the switch boxes; the logic corresponding to the original circuit highlighted in black whereas the decoy logic is in gray. Input, i_4 feeds through the top-left switch box and gate $g4$ is mapped to the upper output of the bottom-right switch box. The network's un-mapped inputs and outputs are connected to the surrounding circuitry

As the network size is increased, it incorporates a larger portion of the design. Since there is already a significant amount of reconfiguration, we forgo the use of LUTs. The intra-network logic prohibits the use of a simplified model for the network. The correct functionality is no longer just a permutation of the inputs to the network, but rather one of a very large space of functionalities dependent on nearly all the key bits. Additionally, the large amount of symmetry has been removed; while some corner case symmetry may remain, it will be highly complex to find and probably of little value to rule out.

4.2 Insertion Algorithm

While resistant to the modeling techniques, the insertion of the Banyan network is more complex than in Full-Lock. Now, gates from the original circuit must be mapped onto the structure of the Banyan network, instead of just being randomly selected. We automate this process to enable scalable exploration of the mapping solution space. To augment the ability to map onto the Banyan structure, we split all gates from the original circuit with three or more inputs into two input gates. We start with a Banyan network of the desired input width, W and encode the problem of finding a mapping as a SAT instance through constraints that we specify below.

The encoding uses a set of variables representing a mapping between an original gate g and a banyan switch box output s. The switch box outputs provide a reference point within the Banyan network that naturally correspond to gate outputs in the original circuit. For all pairs of gates in the original circuit and switch box outputs in the Banyan network, $(g, s) \in C \times B$, we create a mapping variable m_{gs}. The variable is true if gate g is mapped to switch box output s. Over these variables, we encode constraints that ensure the amount of mapping is sufficient. First we ensure at least W gates are mapped to the network.

$$AtLeastW(\{\bigvee_{s \in B} m_{gs} : g \in C\}) \tag{5}$$

Then we encode that at most one gate is mapped per switch box output.

$$\bigwedge_{s \in B} AtMostOne(\{m_{gs} : g \in C\}) \tag{6}$$

We allow multiple switch box outputs to map to the same gate enabling the mapping of gates with fanout. Finally, we prohibit any path directly feeding through from the network inputs to outputs, avoiding the simplest mappings. This is done by prohibiting a gate to be mapped to both the first and last layer of the Banyan network. We show the encoding below where B_i and B_o are respectively the sets of switch box outputs in the first and last layers of the network.

$$\bigwedge_{g \in C} \bigwedge_{s_i \in B_i} \bigwedge_{s_o \in B_o} AtMostOne(m_{gs_o}, m_{gs_i}) \tag{7}$$

To maintain the structure of the circuit, we add constraints that enforce a correspondence between the connectivity of the mapped gates and the switch box outputs. If a gate is mapped to a switch box output, the fanin of the gate in the original circuit must be mapped to the fanin of the switch box (i.e. the switch box outputs from the preceeding network layer). Similarly, we also ensure that at least one of the gate's fanout is mapped to the fanout of the switch box. We allow an exception to this rule if the gate is simply fed through the switch box, which adds flexibility to the circuit structures which can be mapped. Note that here we are allowing feed through for a switch box, but prohibit it through the entire network. More formally, m_{gs} implies that every fanin of g is mapped to the fanin of s or, in the case of a feedthrough, g itself is mapped to the fanin of s. This encoding is shown below.

$$\bigwedge_{s \in B} \bigwedge_{g \in C} \bigwedge_{g_f \in fanin(g)} m_{gs} \rightarrow \bigvee_{s_f \in fanin(s)} m_{g_f s_f} \vee m_{gs_f} \tag{8}$$

Additionally, m_{gs} implies that at least one fanout of g is mapped to the fanout of s or, in the case of a feedthrough, g is in the fanout of s. This encoding is shown below.

$$\bigwedge_{s \in B} \bigwedge_{g \in C} m_{gs} \rightarrow \bigvee_{s_f \in fanout(s)} \bigvee_{g_f \in fanout(g) \cup \{g\}} m_{g_f s_f} \tag{9}$$

This system of constraints is solved and the gates in the resulting mapping are inserted into their corresponding switch boxes and removed from the original circuit. The other MUX inputs are connected to randomly selected decoy functions of the switch box inputs. The network inputs and outputs are connected depending on which gates have been mapped to the first and last layer of switch boxes. It's important to emphasize that no intermediate connections are made to or from the network. Outputs with no mapping are randomly connected to remaining gates such that they have no impact on the logic of the system under the correct key.

5 ATTACK RESULTS

In this section, we provide experimental data to show the effect of using the proposed modeling techniques. We step through each part of our modeling process, showing the incremental results from each. We then demonstrate the resistance of our proposed technique to the miter-based SAT attack.

All attacks are run using a Python implementation of the miter-based SAT attack. The implementation uses PySAT's wrapper for the CDCL-based SAT solver CaDiCaL [3, 10]. Additionally, the attack implementation uses incremental addition of constraints as proposed in [30]. Although Python is not as efficient as c or other low-level languages, most attack time is spent inside the SAT solver and thus the difference is negligible. The logic-enhanced Banyan locking implementation can be found in our repository[1]

Each attack has a timeout 4 of hours, an iteration count of 10, and is executed on a machine with 756GB RAM and 16 2.1GHz cores. The attacks are conducted in parallel while ensuring minimal contention for resources by allotting memory greater than the maximum usage of the largest instances to each run.

5.1 Relaxed Model Comparison

We assess the impact of the model and encoding on attack run time for the standalone Banyan networks. We compare five model, encoding schemes as described in Section 3.1, namely the original Banyan network model and encoding, and all combinations of MUX-based and edge-based encodings with the all-to-all and all-to-all exclusive models. We also compare our logic-enhanced Banyan locking. For each iteration we randomly select a new key. The data is shown in Fig. 6. We report several dimensions: overall attack time, number of attack iterations, number of key variables, number of total variables, and number of clauses.

Immediately obvious is the grouping of attack times. The original and logic-enhanced Banyan networks respectively timeout at input

[1]https://github.com/jpsety/logic_enhanced_banyan_locking

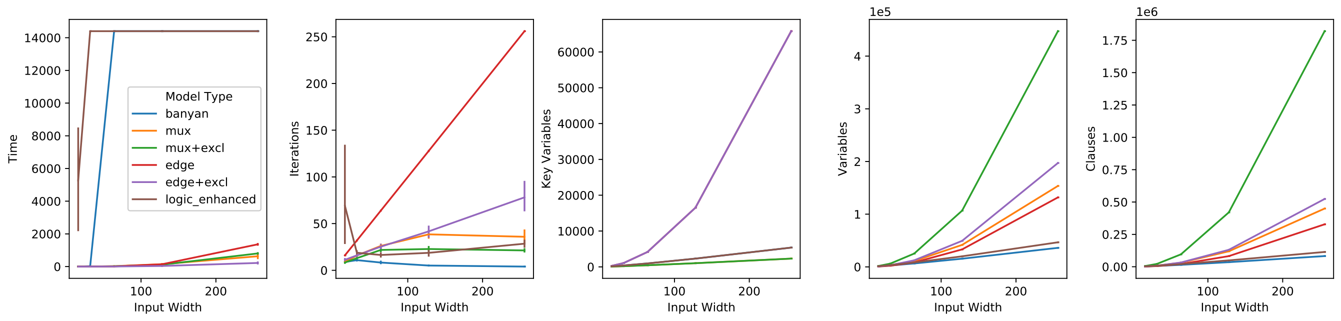

Figure 6: Comparison of encoding schemes for standalone Banyan network, n=10

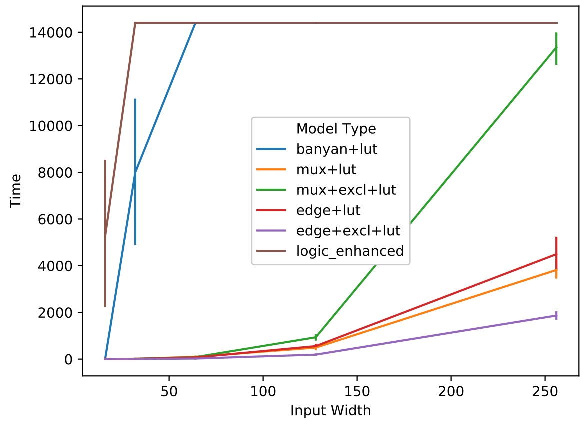

Figure 7: Attack time comparison of encoding schemes for Banyan network with 2-input LUTs connected to the outputs, n=10

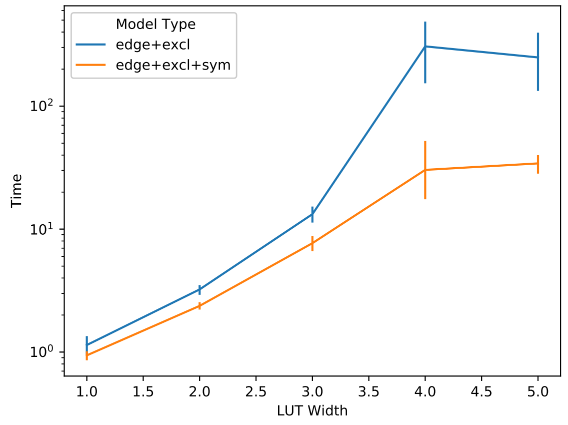

Figure 8: Comparison of attack time at network input width of 32 between encodings with and without LUT symmetry breaking, n=10

widths of 64 and 16. Whereas the attack time of all proposed model-encoding pairs is significantly less, highlighting the impact of the improved models. The number of iterations completed for the original and logic-enhanced Banyan models remain low, a testament to the hardness of the problems. While the edge-based encoding has significantly more key variables, the overall variable and clause counts remain close to the MUX-based encoding's values. Both encoding schemes result in larger formulations than the original Banyan network, however are significantly easier to solve. The MUX-based all-to-all exclusive encoding stands out as by far the largest relaxed encoding.

We then add the output LUTs to the Full-Lock network models and repeat the attacks, reporting just the execution time in Fig. 7. Again, for comparison we show our proposed logic-enhanced Banyan locking scheme. Here we can clearly discern that the edge-based all-to-all encoding performs the best, quickly terminating even with an input width of 256. For reference, this scenario would require the original Banyan network 5,889 keys to implement (two thirds of which are dedicated to inversions as in Fig. 2).

5.2 Effect of Symmetry Breaking

Next, we demonstrate the impact of the LUT symmetry breaking on attack time. Since the amount of symmetry scales with LUT input size, we sweep this parameter and hold the network width fixed at 32. The resulting attack times with and without symmetry are shown in Fig. 8. As the LUT width increases, the advantage of symmetry breaking grows exponentially. At a LUT input width of 5, the difference in attack time about an order of magnitude. This result is consistent with the difference in potential solutions reported in section 3.2.

5.3 Full-Lock and Logic-Enhanced Banyan Attack Results

Finally, the proposed techniques are together applied to Full-Lock. We use the best encoding from the previous results: an edge-based, all-to-all exclusive with symmetry breaking along with the CycSAT acyclic constraints [32]. This is compared to the attack model outlined by the Full-Lock authors: the full Banyan network model alongside the CycSAT acyclic constraints. We also demonstrate our

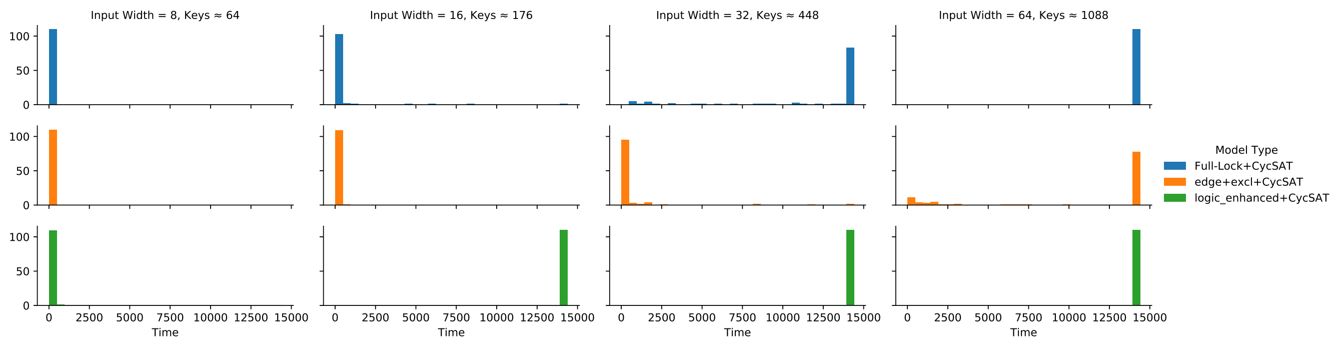

Figure 9: SAT-based attack time for ISCAS85 circuits locked with Full-Lock and logic-enhanced Banyan locking schemes

proposed solution, logic-enhanced Banyan locking. All techniques are run on the ISCAS85 benchmark circuits, sweeping the network input width from 8 to 64. The corresponding key widths range is around 64 to 1088. The results are shown in Fig. 9.

The Full-Lock run times show a trend that mostly agrees with the original results with the exception that some circuits at 32-input width are deobfuscated. These improved results are likely due to the use of a different SAT solver. Our relaxed model shows run times several orders of lower than the original paper. Most circuits at 32-input width (around 448 key bits) are deobfuscated in seconds, with some outliers taking minutes, clearly demonstrating the effectiveness of these techniques. As the input width scales to 64, many circuits are still deobfuscated, but the majority take longer than our 4-hour timeout.

The logic-enhanced Banyan scheme provides a significantly better ratio of key bits to attack time than the Full-lock predecessor. At an input width of 16, the attack times out for all circuits. While this is good, we do not suggest that such small input widths are viable locking techniques as simple enumeration attack schemes may easily deobfuscate them.

6 DISCUSSION

Our experiments show that these modeling techniques are highly effective. The actual attack times for the original Full-Lock implementation are unknown, we just have a lower bound. With this, we can claim that the techniques have decreased attack times by *at least* several orders of magnitude. Important to the success of our strategy was trying different models of the system.

Several concerns remain unexplored. First, the amount of corruption produced by the remaining keys after the miter-based SAT attack has been run for some time. It is common to only report attack run times, however, the keys that can be obtained at timeout may be very close to a correct solution. Understanding the trend in remaining corruption is critical information for a given locking scheme. Also, an attacker can often specify constraints on the keys that the correct solution must respect. One applicable example is timing constraints, wherein every valid key must produce a circuit with a critical path less than the period. Just like the acyclic constraints that are necessary for the attack to complete, timing constraints could rule out significant portions of the key space. Importantly, like our modeling techniques, encoding has a large impact on effectiveness of a constraint.

More specific to our proposed technique, an overhead analysis has not been conducted. It is likely that more must be done to make this a viable solution for high speed designs. Towards that end, a parameter exploration of similar structures may produce increased attack resistance. Our initial implementation maintained the amount of keys from Full-Lock. Varying the amount of keys, connectivity of the network, or decoy logic selection may produce significantly overhead and attack resistance results.

7 CONCLUSION

We have proposed two widely-applicable modeling techniques that can substantially decrease the attack time for logic locking schemes. Any locking scheme that has human-comprehensible regularities is potentially vulnerable to these techniques. We have demonstrated the application of these techniques on a state-of-the-art locking scheme, Full-Lock. The experiments show many orders of magnitude decrease in attack time compared to previously reported results. In general, these modeling techniques are essential considerations in any logic locking technique.

Additionally, we have described logic-enhanced Banyan locking, an extension to the Full-Lock method, that appears to be resistant to these modeling techniques. We demonstrated promising initial attack results, showing that structure of the Banyan network combined with randomly selected decoy logic is not only a mechanism of resisting these techniques, but also harder for the CDCL-based SAT solver used. Of course, this resistance may change with some additional insight or varied attack strategies.

ACKNOWLEDGMENTS

This work was supported in part by the Defense Advanced Research Projects Agency under contract FA8750-17-1-0059 “Obfuscated Manufacturing for GPS (OMG)”, Honeywell Federal Manufacturing & Technologies, LLC under contract A023646, and NSF grant CCF-1618574.

REFERENCES

[1] Fadi A. Aloul, Igor L. Markov, and Karem A. Sakallah. 2003. Shatter: Efficient Symmetry-Breaking for Boolean Satisfiability. In *Proceedings of the 40th Annual Design Automation Conference* (Anaheim, CA, USA) *(DAC '03)*. Association for

Computing Machinery, New York, NY, USA, 836–839. https://doi.org/10.1145/775832.776042

[2] Roberto J Bayardo Jr and Robert Schrag. 1997. Using CSP look-back techniques to solve real-world SAT instances. In *Aaai/iaai*. Providence, RI, 203–208.

[3] Armin Biere. 2018. Cadical, Lingeling, Plingeling, Treengeling and YalSAT entering the sat competition 2018. In *Proceedings of SAT Competition 2018*. 13–14.

[4] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. 1999. Symbolic Model Checking without BDDs. In *TACAS '99: Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*. Springer-Verlag, London, UK, 193–207.

[5] Armin Biere, Marijn Heule, and Hans van Maaren. 2009. *Handbook of satisfiability*. Vol. 185. IOS press.

[6] D. Brand. 1993. Verification of large synthesized designs. In *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*. 534–537.

[7] James Crawford, Matthew Ginsberg, Eugene Luks, and Amitabha Roy. 1996. Symmetry-breaking predicates for search problems. *KR* 96 (1996), 148–159.

[8] Jian Ding, Allan Sly, and Nike Sun. 2015. Proof of the satisfiability conjecture for large k. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. 59–68.

[9] Ujjwal Guin, Ziqi Zhou, and Adit Singh. 2017. A novel design-for-security (DFS) architecture to prevent unauthorized IC overproduction. In *Proceedings of the IEEE VLSI Test Symposium*. https://doi.org/10.1109/VTS.2017.7928946

[10] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. 2018. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In *SAT*. 428–437. https://doi.org/10.1007/978-3-319-94144-8_26

[11] Franjo Ivančić, Zijiang Yang, Malay K. Ganai, Aarti Gupta, and Pranav Ashar. 2008. Efficient SAT-based bounded model checking for software verification. *Theoretical Computer Science* 404, 3 (2008), 256 – 274. International Symposium on Leveraging Applications of Formal Methods (ISoLA 2004).

[12] F. Koushanfar J. A. Roy and I. L. Markov. 2008. EPIC: Ending Piracy of Integrated Circuits. *2008 Design, Automation and Test in Europe* (2008). https://doi.org/10.1109

[13] Hadi Mardani Kamali, Kimia Zamiri Azar, Houman Homayoun, and Avesta Sasan. 2019. Full-Lock. Association for Computing Machinery (ACM), 1–6. https://doi.org/10.1145/3316781.3317831

[14] Boris Konev and Alexei Lisitsa. 2014. A SAT Attack on the Erdős Discrepancy Conjecture. In *Theory and Applications of Satisfiability Testing – SAT 2014*, Carsten Sinz and Uwe Egly (Eds.). Springer International Publishing, Cham, 219–226.

[15] Meng Li, Kaveh Shamsi, Yier Jin, and David Z. Pan. 2019. TimingSAT: Decamouflaging Timing-based Logic Obfuscation. In *Proceedings - International Test Conference*, Vol. 2018-October. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/TEST.2018.8624671

[16] João P Marques-Silva and Karem A Sakallah. 1999. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Comput.* 48, 5 (1999), 506–521.

[17] Jeyavijayan Rajendran, Huan Zhang, Chi Zhang, Garrett S. Rose, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. 2015. Fault Analysis-Based Logic Encryption. *IEEE Trans. Comput.* 64, 2 (2015), 410–424. https://doi.org/10.1109/TC.2013.193

[18] Kaveh Shamsi, Meng Li, David Z Pan, and Yier Jin. 2018. Cross-Lock: Dense Layout-Level Interconnect Locking using Cross-bar Architectures. (2018). https://doi.org/10.1145/3194554

[19] Kaveh Shamsi, David Z Pan, and Yier Jin. 2019. IcySAT: Improved SAT-based Attacks on Cyclic Locked Circuits. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–7.

[20] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. 2017. The First Collision for Full SHA-1. In *Advances in Cryptology – CRYPTO 2017*, Jonathan Katz and Hovav Shacham (Eds.). Springer International Publishing, Cham, 570–596.

[21] Pramod Subramanyan, Sayak Ray, and Sharad Malik. 2015. Evaluating the security of logic encryption algorithms. *Proceedings of the 2015 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2015* (2015), 137–143. https://doi.org/10.1109/HST.2015.7140252

[22] Joseph Sweeney, Marijn J. H. Heule, and Lawrence Pileggi. 2020. Sensitivity Analysis of Locked Circuits. In *LPAR23. LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning (EPiC Series in Computing, Vol. 73)*, Elvira Albert and Laura Kovacs (Eds.). EasyChair, 483–497. https://doi.org/10.29007/7tpd

[23] Grigori S Tseitin. 1983. On the complexity of derivation in propositional calculus. In *Automation of reasoning*. Springer, 466–483.

[24] Yang Xie and Ankur Srivastava. 2019. Anti-SAT: Mitigating SAT Attack on Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 2 (2 2019), 199–207. https://doi.org/10.1109/TCAD.2018.2801220

[25] Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan J V Rajendran, and Ozgur Sinanoglu. 2016. SARLock: SAT attack resistant logic locking. *Proceedings of the 2016 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2016* (2016), 236–241. https://doi.org/10.1109/HST.2016.7495588

[26] M Yasin, B Mazumdar, and O Sinanoglu. 2017. Security analysis of anti-sat. *(Asp-Dac), 2017 ...* (2017), 342–347. http://ieeexplore.ieee.org/abstract/document/7858346/

[27] M. Yasin, J. J. Rajendran, O. Sinanoglu, and R. Karri. 2016. On Improving the Security of Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 9 (2016), 1411–1424.

[28] Muhammad Yasin, Abhrajit Sengupta, Benjamin Carrion Schafer, Yiorgos Makris, Ozgur Sinanoglu, and Jeyavijayan Rajendran. [n.d.]. What to Lock? Functional and Parametric Locking. ([n. d.]). https://doi.org/10.1145/3060403.3060492

[29] Muhammad Yasin, Abhrajit Sengupta, Mohammed dari Nabeel, Mohammed Ashraf, Jeyavijayan Rajendran, and Ozgur Sinanoglu. [n.d.]. *Provably-Secure Logic Locking: From Theory To Practice.* Technical Report.

[30] Cunxi Yu, Xiangyu Zhang, Duo Liu, Maciej Ciesielski, and Daniel Holcomb. 2017. Incremental SAT-Based Reverse Engineering of Camouflaged Logic Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36, 10 (2017), 1647–1659.

[31] G. L. Zhang, B. Li, B. Yu, D. Z. Pan, and U. Schlichtmann. 2018. TimingCamouflage: Improving circuit security against counterfeiting by unconventional timing. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 91–96.

[32] Hai Zhou, Ruifeng Jiang, and Shuyu Kong. 2017. CycSAT: SAT-based attack on cyclic logic encryptions. In *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*. https://doi.org/10.1109/ICCAD.2017.8203759