A TECHNICAL REPORT ON LOGIC OBFUSCATION USING RECONFIGURABLE LOGIC AND ROUTING BLOCKS

Hadi Mardani Kamali

Department of Electrical and Computer Engineering (ECE)
George Mason University, Fairfax, VA

Fall 2019

Table of Contents

List of Tables

List of Figures

Abstract

The increasing cost of building, operating, managing, and maintaining state-of-the-art silicon manufacturing facilities has pushed several stages of the semiconductor device's manufacturing supply chain offshore. However, many of these offshore facilities are identified as untrusted entities. Processing and fabrication of ICs in an untrusted supply chain poses a number of challenging security threats such as IC overproduction, Trojan insertion, Reverse Engineering, Intellectual Property (IP) theft, and counterfeiting.

To counter these threats, various hardware design-for-trust techniques have been proposed. Logic locking, as a proactive technique among these techniques, has been introduced as a technique that obfuscates and conceals the functionality of IC/IP using additional key inputs that are driven by an on-chip tamper-proof memory.

Shortly after introducing the primitive logic locking solutions, a very strong Boolean attack, the Satisfiability (SAT) attack. It was shown that the SAT attack could break all previously proposed primitive locking mechanisms in almost polynomial time. To thwart the strength of SAT attack, researchers have investigated many directions, such as formulating locking solutions that significantly increase the number of required SAT iterations, or formulating the locking solutions such that it is not translatable to a SAT problem.

However, further investigations demonstrated that some of these locking techniques are vulnerable to other types of attacks such as Signal Probability Skew (SPS) attack, removal attack, approximate-based attack(s), bypass attack, and Satisfiability Module Theories (SMT) attack. In addition, these techniques suffer from very low output corruption. Hence, an unactivated IC behaves almost identical to an unlocked IC with exception of one or few inputs.

Recent obfuscation schemes have leveraged reconfigurable logics to alleviate various hardware security threats. In this report, first we will introduce LUT-Lock, which demonstrate that how using reconfigurable logics, e.g. Look-Up-Tables (LUTs) can provide resilience against such powerful attacks, i.e. SAT attack and its derivations. LUT-Lock obfuscates a netlist while embedding several key features that make the obfuscation a hard problem for state of the art attacks with particular attention to Satisfiability (SAT) Attacks. To develop this defense mechanism, we have identified several key features that increase the difficulty of obfuscation for SAT attacks. We illustrate how by utilizing each feature during the obfuscation, the SAT problem becomes harder. We propose LUT-Lock algorithm which combines all features, providing the best defense against SAT attacks. Then we demonstrate that how routing and logic blocks can be used for building SAT-hard solutions, which is thoroughly discussed in this report, is to significantly increase the run-time of each iteration of the SAT solver. We explore the characteristics and principles of designing this SAT-hard obfuscation solutions, using reconfigurable logic and routing blocks, where the goal is to exponentially increase the time required for each iteration of the SAT attack. As a strong representative member of this class of obfuscation techniques, we introduce Full-Lock. The Full-Lock is constructed using a set of cascaded fully programmable logic and routing blocks (PLR) networks that replace parts of the logic and routing in the desired netlist. The PLRs are SAT-hard instances designed to construct a desired ratio between the number of clauses and the number of variables with PLRs are translated to their Conjunctive Normal Form (CNF). The cascaded and non-blocking design of PLR pushes the SAT solver's algorithm to build a very deep decision tree and to spend significant time in hopeless regions of the decision tree, causing a significant increase in each iteration of SAT attack.

Chapter 1: Introduction

1.1 Why Hardware Obfuscation

The cost of building a new semiconductor fab was estimated to be $5.0 billion in 2015, with large recurring maintenance costs [3], and sharply increases as technology migrates to smaller nodes. To reduce the fabrication cost, most of the manufacturing and fabrication is pushed offshore [3]. However, many of the offshore fabrication facilities are considered to be untrusted. Manufacturing in untrusted foundries has raised concern over potential attacks in the manufacturing supply chain, with an intimate knowledge of the fabrication process, the ability to modify and expand the design prior to production, and an unavoidable access to the fabricated chips during testing. Accordingly, fabrication in untrusted foundries has introduced multiple forms of security threats from supply chain including that of overproduction, Trojan insertion, Reverse Engineering (RE), Intellectual Property (IP) theft, and counterfeiting [4].

To counter these threats, various hardware *design-for-trust* techniques have been proposed, including *watermarking*, *IC metering*, *split manufacturing*, *IC camouflaging*, and *logic locking* [4]. The watermarking and IC metering techniques are passive protection models that could be used to detect overproduction or illegal copies, however, they cannot prevent IP theft or overproduction. The *Camouflaging* techniques use logic gates (or other physical structures such as dummy vias) with high structural similarity, that are indistinguishable from one another to protect against reverse engineering. However, camouflaging is only effective against post-manufacturing attempt(s) of reverse engineering, while it provides no limitations against a foundry's attempt at reverse engineering, as a foundry has access to all masking layers and is not trapped by structural ambiguity for being able to logically extract a netlist. The obfuscation (*logic locking*) [5] on the other hand, introduce

limited programmability by inserting key programmable gates to hide or lock the functionality. By using obfuscation, the target chip produces the correct output only when the key inputs are correct, and the unlock keys are almost stored in a tamper-proof non-volatile memory. However, as a more resilient solution can be managed remotely to avoid storage of key inside the chip [6]. The purpose of obfuscation is to protect against RE at an untrusted foundry. By using obfuscation, even by having all mask information, the attacker cannot generate the correct functionality of a circuit without the correct key values, and such key values are not shared with the manufacturer.

1.2 Challenges in Hardware Obfuscation

Shortly after the introduction of first published obfuscation schemes, a new and powerful attack based on Boolean Satisfiability (SAT) was formulated and revealed [7]. In this attack model, the attacker has access to a reverse engineered but obfuscated netlist, and a functional and unlocked chip. Using this attack model, the formulated Boolean Satisfiability Attack (SAT Attack) can effectively break all previously proposed logic encryption techniques, including random insertion (RLL), fault-analysis (FLL), interference-based logic locking (SLL), and logic barriers [5,8–10]. The SAT solver iteratively eliminates sets of incorrect keys and finds the correct key within a small time, and unlike Brute force attack that requires attack time exponential with respect to the number of inputs, its execution time grows almost polynomially. Existing SAT attack, which can be modeled with query-by-disagreement (QBD) or uncertainty-sampling, minimizes the number of queries (inputs) required to learn (deobfuscate) the target function (obfuscated logic). Also, SAT attack terminates when no more disagreeing inputs can be found, at which time the attack guarantees to find the correct key. However, to defend against powerful SAT attacks, different obfuscation schemes have been proposed, such as SARLock and Anti-SAT [11,12]. However, further research illustrated that some of these locking schemes are vulnerable to other types of attacks such as Signal Probability Skew (SPS) and removal attacks [13].

1.3 Hardware Obfuscation using Reconfigurable Logic

A plethora of work have focused on utilizing reconfigurable logic such as LUT-based obfuscation as a defense against reverse engineering (RE) attacks [8, 14–16]. In LUT-based obfuscation, the design is partially mapped to LUTs, for example, if a 2-input AND gate has to be obfuscated using LUT of size 2, one can set the configuration bits of the LUT to '0001' as per the truth table of the AND gate. The partial mapping of the circuit results in a design implementation that is a hybrid of custom (ASIC) and programmable (FPGA) styles. Besides, reconfigurable bits can be stored in non-volatile memory (NVM) such as magnetic tunnel junction (MTJ). These, stored bits are highly susceptible to be lost during RE de-layering process. With the incorrectly configured LUT block, the design will remain unintelligible, which will refrain the attacker from understanding the functionality of the design.

LUT-based obfuscation has been previously visited by few researchers. The work in [8] suggest using LUTs for obfuscation and provides several replacement strategies to secure a netlist. However, the proposed mapping algorithms are not resilient against SAT attacks, and are only evaluated in terms of power, performance and area (PPA) overhead, while the claim on the security of these schemes is made solely base on inability to readout the content of LUTs after reverse engineering. The work in [14] proposed a STT-LUT-based obfuscation with three different LUT placement algorithms. This work further focuses on PPA impact of their solution and illustrates that utilizing STT-based LUTs could reduce the PPA impact. However, the proposed solution does not consider its resiliency against SAT attack. As a part of this report, we introduce LUT-Lock, which obfuscates a netlist while embedding several key features that make the obfuscation a hard problem for state of the art attacks with particular attention to Satisfiability (SAT) Attacks. To develop this defense mechanism, we have identified several key features that increase the difficulty of obfuscation for SAT attacks. We illustrate how by utilizing each feature during the obfuscation, the SAT problem becomes harder. We propose LUT-Lock algorithm which

combines all features, providing the best defense against SAT attacks.

1.4 SAT-Hard Instances via Configurable Logic and Routing

Although engaging LUTs in LUT-Lock makes the obfuscation solution resilient to SAT attack, our study on the existing reconfigurable logic obfuscation schemes shows that most of the LUT-based techniques impose almost prohibited PPA overhead that is not acceptable. In fact, among the previously proposed reconfigurable logic obfuscation schemes, we investigate LUT-based obfuscation and observe that in the LUT-based obfuscation, utilizing large-size LUTs guarantees the security against state-of-the-art SAT attacks, but at the cost of significant PPA overheads. Hence, only using reconfigurable logic cannot be considered as a promising solution for hardware obfuscation.

In general, to thwart the strength of SAT attack, researchers have investigated two main directions

1. formulating locking solutions that significantly increase the number of required SAT iterations (inputs to be tested)

2. formulating the locking solutions such that it is not translatable to a SAT problem.

The first approach in which formulating obfuscation and locking solutions significantly increase the number of SAT iterations was assumed to be a perfect anti-SAT solution, such as SARLock, Anti-SAT, SFLL, and LUT-Lock [11, 12, 15, 18]. In extreme case, using these techniques, each tested input (each iteration) invalidates a single key combination. Hence, by using these techniques, a SAT attack, similar to a brute force attack, faces an exponential runtime. However, further investigations demonstrated that some of these locking techniques are vulnerable to other types of attacks [19] such as Signal Probability Skew (SPS) attack [13], removal attack [20], approximate-based attack(s) [21], bypass attack [22], and Satisfiability Module Theories (SMT) attack [23]. In addition, these techniques suffer from very low output corruption. Hence, an un-activated IC behaves almost identical to an

unlocked IC with exception of one or few inputs.The second approach investigated by researchers was formulating obfuscation and locking mechanisms that were not translatable to SAT problems. Example of such techniques includes the use of cyclic Boolean logic for locking [24] or behavioral locking of the logic [25]. The cyclic obfuscation creates combinational cycles in the design. This invalidates the Directed Acyclic Graph (DAG) nature of logic and forces a SAT attack to either be trapped in an infinite loop or to generate an incorrect key upon termination [24]. Alternatively, in [25], a behavioral (non-Boolean) locking scheme was introduced where the locking mechanism targeted the setup and hold properties (timing properties) of the circuit. However, shortly after the introduction of these obfuscation techniques, researchers revealed stronger and more advanced modeling and attack solutions such as cycSAT [26], and Satisfiability Module Theories (SMT) attack [23] that were able to model the cyclic or behavioral locking into a SAT or SAT+theory solvable logic problems. A new (and third) direction for building SAT-hard solutions, which is thoroughly discussed in this paper, is to significantly increase the runtime of each iteration of the SAT solver. The only existing solution that somewhat fits this category is the Cross-lock [27], in which a one-time programmable interconnect mesh is used to obfuscate the routing of a netlist, and the resulting obfuscated netlist substantially increase the runtime of each iteration of the SAT attack. However, we will illustrate that obfuscation solution in [27], although a step in the right direction, is not a strong solution in this space, and by following the principles and design guidelines discussed in this paper, it is possible to construct obfuscated circuits that translate into far harder SAT circuits than Cross-lock.

As the second part of this report, in Section 3 we explore the characteristics and principles of designing this new category of SAT-hard obfuscation solutions, where the goal is to exponentially increase the time required for each iteration of the SAT attack. As a strong representative member of this class of obfuscation techniques, we introduce Full-Lock [28]. The Full-Lock is constructed using a set of cascaded fully programmable logic and routing blocks (PLR) networks that replace parts of the logic and routing in the desired netlist. The PLRs are SAT-hard instances designed to construct a desired ratio between the number of

clauses and the number of variables with PLRs are translated to their Conjunctive Normal Form (CNF). The cascaded and non-blocking design of PLR pushes the SAT solver's algorithm to build a very deep decision tree and to spend significant time in hopeless regions of the decision tree,causing a significant increase in each iteration of SAT attack.

Chapter 2: LUT-Based Obfuscation for for ASIC and FPGA

2.1 LUT-based obfuscation in FPGA

In an FPGA solutions the hardware resources are are fixed and is designed independent of a given netlist. Hence by nature, state of the art FPGAs provide a large pool of resources to be applicable to a wide range of applications, resulting in a large number of non-utilized LUTs after mapping a netlist to the FPGA. For instance, the study in [29] depicts the utilization of Altera Cyclone V after mapping a diverse set of benchmarks of various scale and complexity to this FPGA, and reported that FPGA utilization is typically low. This phenomenon was coined as FPGA-Dark-Silicon [29]. These unmapped and unutilized LUTs are freely available and could be used for obfuscating a to-be-mapped netlist. Hence, LUT-based obfuscation in FPGAs could be considered as utilizing unused LUTs, or using larger than needed LUTs, where the connectivity and impact of additional logic is controlled using keys. The process of using LUTs in FPGA for the purpose of logic obfuscation is illustrated in in Fig. 2.1(b), where some of 2-input (or 3-input) logic gates could be mapped to a LUT of larger size (e.g. size 4 or 5). Then, the additional inputs can be taken from the output of an internally implemented Non-Linear Feedback Shift Register (NLFSR) or a Physical Unclonable Function (PUF) [30]. In addition, by changing the ordering of inputs based on the key inputs (generated by PUF), the obfuscated circuit possibilities increases. Lets assume a PUF is used. In this case, each FPGA has a unique PUF response. By knowing the PUF response ahead of time, the bitstream will load the LUTs with proper values and will transmit the directives for connecting the known PUF outputs to the proper LUT inputs and switch box select lines. However, the PUF values will not be transmitted in the bitstream. This missing key values serve as the obfuscation key in LUT based obfuscation. Also note that the bitstream in this case is unique for each FPGA, as each FPGA has a unique PUF

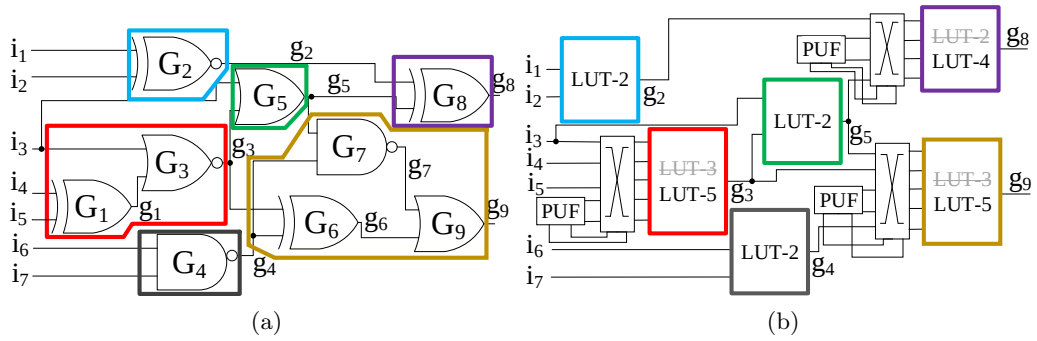


Figure 2.1: (a) Original circuit (b) Modified using configurable switches + PUFs (NLFSRs).

response. In this case, even if the bitstream is leaked, the PUF response remains unknown, making the problem similar to ASIC flow, where after reverse engineering the obfuscated netlist is available, but the keys are unknown.

2.2 LUT-based obfuscation in ASIC

In ASICs, utilizing LUTs for obfuscation can lead to the considerable area and delay overhead. In the CMOS implementation of LUTs, the area overhead of the memory elements in a LUT exponentially increases as a function of its input size. Hence, the imposed area overhead limits the number of LUTs that could replace regular gates in a netlist. In addition, the performance/delay requirements constrain the placement of LUTs in timing critical and near timing critical paths. However, with the introduction of STT and MTJ based LUTs [14, 31] and the promise of integration of STT and MTJ/pMTJ-based LUTs into the same CMOS process, the area overhead of LUTs is expected to sharply reduce. Integration of CMOS and MTJ/STT devices makes it possible for a larger number of LUTs to be implemented given a fixed area overhead. Using LUTs for obfuscation in ASICs is straightforward: selected cells are removed and replaced by LUTs. The functionality of cell remains hidden to the manufacturer. LUTs are then programmed after fabrication in a trusted testing facility.

2.3 Proposed LUT-Lock Obfuscation Algorithm

Our proposed LUT-Lock algorithm combines several key features, each enhancing its ability to resist against SAT attacks. In this section, we first explain each key feature, and then propose the LUT-Lock algorithm that combines all features into a comprehensive solution. In the result section of this paper, we illustrate how by adding each key feature, the resiliency of obfuscated netlist against SAT attack increases, proving that the resiliency gained from adding this features are orthogonal to one another.

2.3.1 FIC: Focusing on the Fan-In Cone of minimum number of primary output

The first criteria for selection of candidate gates is derived from the observation that higher output corruption reduces resiliency of obfuscation solution against SAT attacks [11, 12]. Hence, by mapping the LUTs such that it affects the minimum number of primary outputs (POs), the degree of output corruption reduces, increasing the strength of obfuscation against SAT attacks. To achieve this, we limit the LUT insertion to the fan-in cone of smallest possible set of primary outputs (best case being single output), and we refer this algorithm as FIC. Note that FIC LUT-replacement still corrupts other outputs, as the intersection of fan-in cones of different outputs is not empty. In addition, the number of gates in the intersection of fan-in cones increases as we move from outputs toward inputs. Hence the obfuscation should be designed to replace the closest cells to the selected output first. This could be achieved by means of a Breadth First Search (BFS). In order to avoid timing violation due to replacing a gate with LUT, we estimates the delay of all timing paths through a gate selected for replacement. If the estimated delay is more than predefined threshold (e.g. 10% delay overhead), the allowance of replacement for this candidate will be revoked, and next candidate will be checked for replacement. After replacing all gates in the current Fan-In Cone, a new primary output will be selected.

In FIC algorithm, the output pin(s) selected for obfuscation should meet two conditions:

(1) Total Positive Slack (TPS) of all timing paths leading to that primary output(s) should be large. This is because replacing a gate with LUT incurs additional delay in every timing path that passes through that gate. Hence, we need available timing slack for replacement of faster logic gates with slower LUTs. (2) it must have a large fan-in cone size, giving us more candidate gates for replacement. Fig. 2.2(a) illustrates the FIC replacement strategy. Between the two outputs, i.e. g_8 and g_9, g_9 is not selected, as it contains the largest number of timing critical paths. When using BFS for gate selection, FIC selects gates {G_8 and G_5} or {G_8, G_5, G_2, and G_4} when its asked to replace 2 or 4 gates respectively. For large circuits, we define two coefficients (α and β) for prioritizing these two conditions to generate a cumulative weight which helps selecting the best candidate output. For this purpose, we normalize the TPS (into TPS*) and FIC (into FIC*) with respect to their maximum possible values in the given circuit. Then using $\alpha.TPS^* + \beta.FIC^*$, we obtain the cumulative weight for the FIC selection process.

2.3.2 HSC: Focusing on Higher Skew Gates in FIC

Our investigation on the hardness of many tested LUT placement strategies revealed that the cells with higher Signal Probability Skew (SPS) at their output are better candidates for obfuscation. The SPS at the output of a gate is defined as $|P_r(0) - P_r(1)|$, with $P_r(1)$ and $P_r(0)$ being the probability of having a 1 or 0 at the output of the gate respectively. The SPS of a gate is a measure of its controllability using primary inputs. The higher the SPS, the lower the controllability of the respective gate. Hence, selecting a high SPS output gate lowers the chances of SAT solver selecting an input that tests the output of that gate.

With this observation, the second step of our proposed algorithm is to modify the FIC to perform the gate selection based on its measure of gate's output (higher) skew probability. In this modified FIC algorithm, which is now referred to as HSC, the gate selection strategy is modified as follows: within the Fan-In cone of selected output(s) based on FIC, the replacement priority is given to gates with higher SPS; In HSC, when a gate is selected for obfuscation, its fan-in gates will be added to the list of gates that could be visited in the

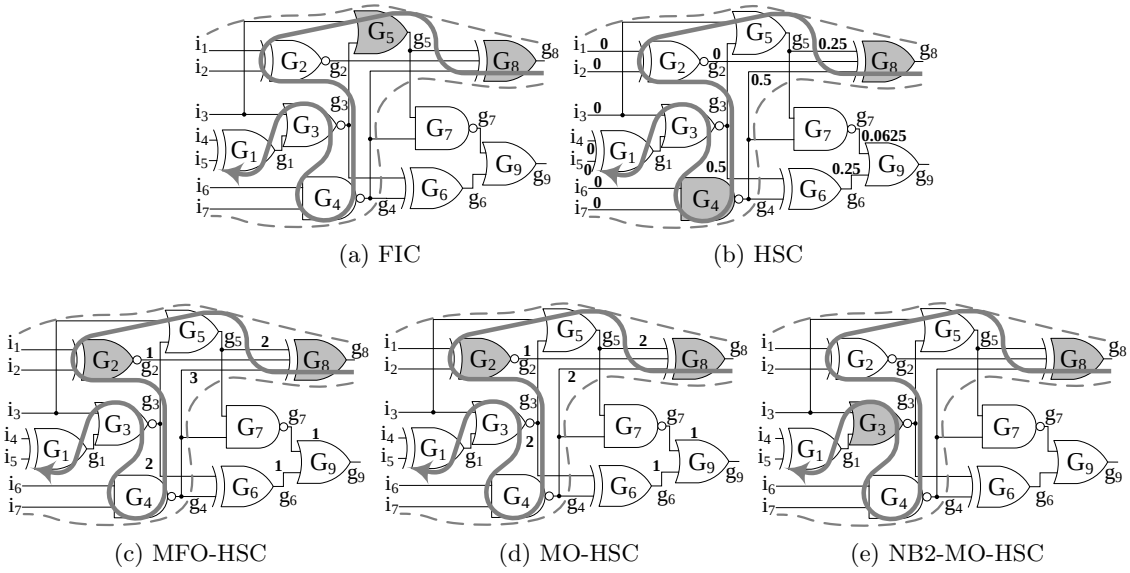


Figure 2.2: Gate selection process based on various sub-algorithms.

next search for gate replacement, and the gates with the highest SPS will be selected among all gates in the list. Similar to FIC algorithm, each gate replacement candidate should pass the timing check, otherwise ignored. HSC replacement flow is illustrated in Fig. 2.2(b). In the first invocation of HSC, fan-in cone of gate G_8, for satisfying the FIC requirements, is selected and is obfuscated. For the 2^{nd} gate selection, HSC has three candidates G_2, G_5, and G_4. Based on the skew probability of wires, as illustrated in Fig. 2.2(b), G_4 with SPS of 0.5 is selected over G_5 and G_2 with SPS of 0.5 and *zero* respectively. For the 3rd gate selection, HSC appends the fan-in gates of G_4 (Here is primary inputs and will be ignored!) as candidate gates for replacement along with G_2 and G_5. Hence, among these 2 gates, G_5 is selected for having the higher SPS.

2.3.3 MFO-HSC: Focusing on gates with Minimum Fan-Out

As mentioned previously, lowering the output corruption increases the difficulty of the SAT attack [11,12]. Although we develop FIC in the first step, the probability of having a fan-in cone with no common gate with other fan-in cones is close to *zero*. Separating the fan-in cones of different outputs could be achieved by replicating common gate, however this will

result in a large area overhead. In order to limit the primary output corruption without exploding the area, we introduce another sub-algorithm in which we give obfuscation priority to candidate gate with lowest fan-out. We refer to this gate selection strategy as MFO-HSC.

In MFO-HSC algorithm, a BFS search is first deployed (FIC), visiting all candidate gates at the current breadth, and gate(s) with a minimum number of fan-outs will be selected. Whenever a tie between two or more gates is observed, the gate with the highest SPS is selected. When a gate is obfuscated, its fan-in gates are added to the list of candidate gates that will be visited in the next gate selection. Similar to FIC, each gate replacement candidate should pass the timing check, otherwise ignored. Fig. 2.2(c) depicts how the MFO-HSC works; Similar to FIC, the fan-in cone of g_8 is selected for obfuscation and G_8 is obfuscated. Based on BFS, the next candidates are G_5, G_2, and G_4. The gate G_2 is selected over G_5 and G_4 for having fan-out of 1. The fan-in of G_2 is then added to the candidate gates for the next visit. In this figure, the fan-in of G_2 are primary inputs, and they are ignored, and the the next candidate gate is only G_5.

2.3.4 MO-HSC: Focusing on Gates with least impact on POs

Based on our observation in MFO-HSC, there are some gates that have more than one fan-out, but they only affect one output. For instance, as it can be seen Fig. 2.2(c), the fan-out of g_4 is 2. However, it affects only g_9. This observation led us to introduce a similar but more efficient sub-algorithm, which is called MO-HSC. In this sub-algorithm rather than looking at the fan-out of the candidate gates, we count the number of outputs that are connected to each candidate gate. MO-HSC requires additional parsing and processing, however it further reduces the output corruption as a result of obfuscation. Similar to MFO-HSC, the tie between two candidate gates (for affecting an equal number of outputs) is broken using SPS of respective gates. Each time a gate is selected for its obfuscation, the fan-in of the gate is added to the list of candidate gates to be considered for the next gate selection. Similar to FIC algorithm, each gate replacement candidate should pass the timing check, otherwise ignored. MO-HSC is illustrated in Fig. 2.2(d), where after selecting

the G_8 based on FIC selection policy, the gate G_2 is selected over G_5 and G_4 for impacting smaller number of outputs.

2.3.5 NB2-MO-HSC: Avoiding Back-to-Back insertion of LUTs

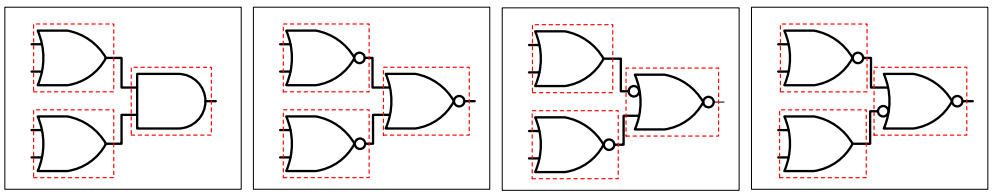
The back-to-back obfuscation of gates with LUTs suffers from the increased number of key-possibilities as a result of the provided freedom in exploiting gate conversion based on De Morgans's Laws. For instance, as it can be seen in Fig. 2.3, the back-to-back obfuscation of the function $(A \vee B) \wedge (C \vee D)$, using 2-input LUTs, could have 4 different combinations of programmable logic based on De Morgans's Laws. So, the number of correct keys from the intended 1 increases to 4. Each additional gate obfuscated in the fan-in of this logic cone, creates another set of possibilities after application of De Morgans's law, leading to exponential increase in the number of valid keys, a phenomenon that we refer to as *correct key explosion*. Depending on the growth rate of the set of valid-keys and the number of keys, obfuscating more gate may even reduce the obfuscation strength. This is illustrated in Fig. 2.4 where execution time of a SAT solver, and a number of generated keys per each inserted LUT for the benchmark C5315 of ISCAS-85 is plotted. The LUTs are placed back-to-back, hence, insertion of each LUT increases the number of keys. The plot focuses on the insertion of 38th to the 45th LUT. The insertion of 41st and 42nd LUT, produces a large number of new keys (around 10^4) based on De Morgan gate conversion possibilities. Hence, the SAT solver execution time doesn't increase. On the other hand, replacement of gate 40 produces far less number of new keys (in range of 10s). Hence the growth of the set of candidate/possible keys exceeds the growth rate of correct keys, significantly increasing the run-time of SAT solver. From this key observation, we need to suppress the growth-rate of correct keys from exploitation of De Morgan's gate conversion laws. So, we introduce another algorithm, NB2-MO-HSC, which implements this restriction by avoiding back-to-back obfuscated, keeping the set of correct keys at a minimum. In this gate replacement strategy, we first select the candidates in FIC using no back-to-back constraint. Then, the selection among the candidates is made based on candidate gate's connectivity to the

minimum number of outputs. If there is a tie among candidates, the SPS of candidate gates determines the selection. As soon as a gate is selected, the NB2-MO-HSC searches the fan-in of the selected gate, skips one logic level (no back to back), and adds the fan-in of all skipped gates to the set of candidate gates for the next gate selection. Similar to FIC, each gate replacement candidate should pass the timing check, otherwise ignored. As illustrated in Fig. 2.2(e), the application of NB2-MO-HSC results in the selection of G_8 and G_3 as first two gates to be obfuscated.

The Algorithm 1 captures the detail implementation of the proposed Lut-Lock obfuscation flow implementing the NB2-MO-HSC policy. As mentioned previously, the overall structures of MFO-HSC and MO-HSC are the same, and since MO-HSC provides slightly more resilient behaviour and also more possible candidates during each iteration, we embed MO-HSC in the proposed LUT-Lock algorithm.

2.4 Experimental Setup

For benchmarking the proposed LUT-Lock algorithm, we used a farm of desktops equipped with Intel Core-i5 processor and 8GB of RAM. For a fair comparison, and to reduce the impact of the operating system background processes, we dedicated one machine to each SAT solver at a time, and installed Ubuntu Server 16.04.3 LTS operating system in shell mode. We used the largest ISCAS-85 benchmarks (C2670, C3540, C5315, C6288, and C7552) to show the effectiveness of the proposed algorithm. We employed the Lingling-based SAT attack described and developed by [7]. We measured the SAT solver execution



(a) OR-AND (b) NOR-NOR (c) Custom1 (d) Custom2

Figure 2.3: Gate conversion using De-Morgan's law.

Algorithm 1 LUT-Lock: Implementing NB2-MO-HSC for LUT-based netlist obfuscation

```
1: α = β = 0.5;                                          ▷ α: TPS coeff, β: FIC size coeff;
2: γ = 0.1                                               ▷ γ: feasible delay overhead
3: max_delay_thr = γ×CriticalPath;
4: MaxSize_FIC = Max_TPS = 0;                            ▷ Total Positive Slack (TPS);
5: Forbidden_output_list = []
6: outputs_list = find_outputs(Circuit C);
7: for each (output in outputs_list) do
8:     if (output not in Forbidden_output_list) then
9:         current_FIC = BFS(output);
10:        for all (paths in current_FIC) do
11:            Current_TPS = TPS_Calc(current_FIC, paths);
12:            Current_Weight = α×Current_TPS + β×sizeof(current_FIC)
13:            Max_Weight = α×Max_TPS + β×MaxSize_FIC
14:            if (Current_Weight > Max_Weight) then
15:                candidate_output = output;
16:                MaxSize_FIC = sizeof(BFS(candidate_output));
17:                Max_TPS = Current_TPS;
18: candidate_list = Forbidden_list = [];
19: candidate_list.append(candidate_output);
20: while (num_of_obfuscated < target_no) do
21:     if (candidate_list == φ) then
22:         Forbidden_output_list.append(candidate_output)
23:         go to line 5
24:     else
25:         current_candidate = candidate_list[0];
26:         if (delay_estimate(current_candidate) < max_delay_thr) then
27:             replace_LUT(current_candidate);
28:             current_candidate_childlist = current_candidate.child;
29:             Forbidden_list.append(current_candidate_childlist);
30:             for each (current_child in current_candidate_childlist) do
31:                 if (current_child.child not in Forbidden_list) then
32:                     candidate_list.append(current_child.child)
33:             sort_list(candidate_list, min_out_impact);
34:             for all (candidate_list_members with equal min_out_impact) do
35:                 sort_list(candidate_list, skew_probability);
36:         else
37:             remove current_candidate;
```

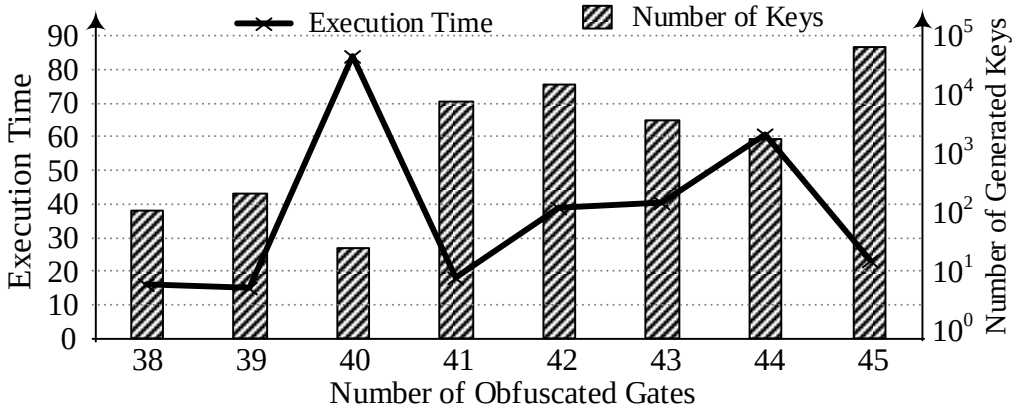


Figure 2.4: Impact of B2B Insertion on Correct Key Pool Size and SAT Runtime (C5315).

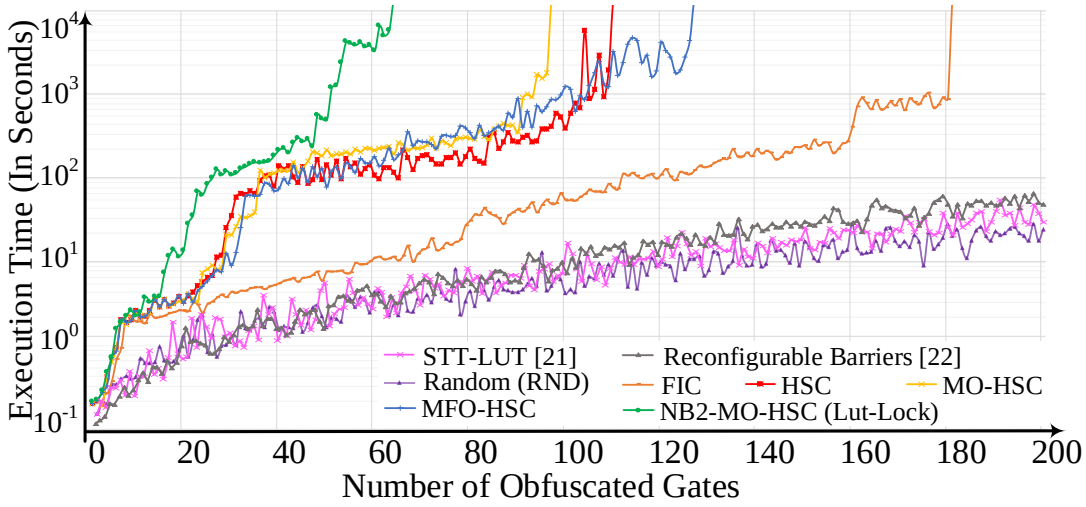
time by increasing the number of obfuscated gates from 1 to 200. A run-time limit of 1.1×10^4 seconds was set for the SAT solver.

2.5 Results and Discussion

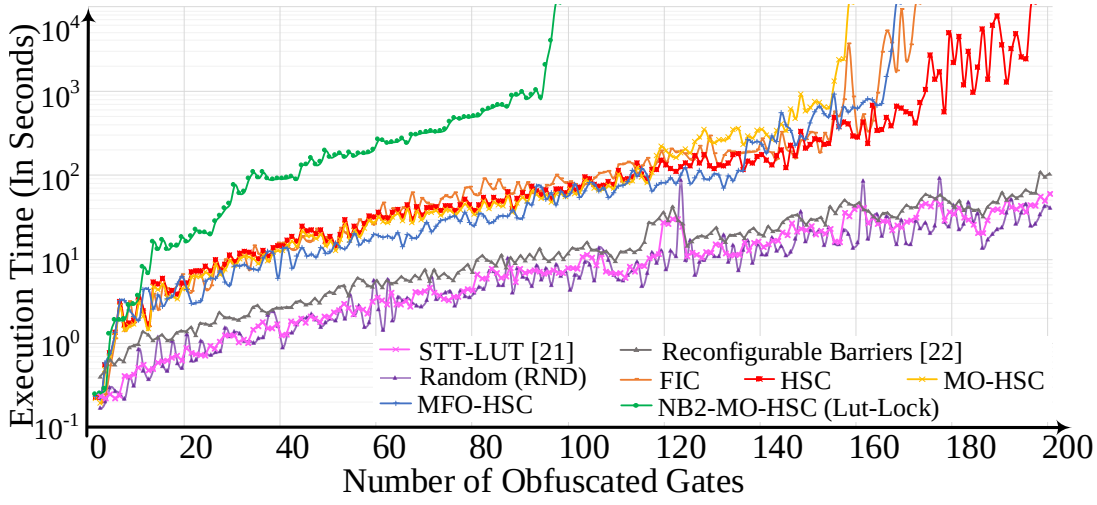
In order to show the effectiveness of each key feature of the proposed algorithm, we compared the execution time of SAT solver on circuits which are obfuscated based on these sub-algorithms. We also compare the effectiveness of the proposed LUT-Lock with that of previous work in STT-LUT [14] and Reconfigurable barriers [8].

As illustrated in Fig. 2.5 the SAT solver's execution time increases as the replacement algorithm evolves from Random replacement to FIC to HSC to MFO-HSC to MO-HSC to MB2-MO-HSC, illustrating the orthogonal improvement of added features in providing resiliency against SAT attacks. The LUT-Lock algorithm, implementing the NB2-MO-HSC replacement policy, combines all key features and provides a close to exponential increase in the execution time of SAT attack with respect to the number of obfuscated gates.

As illustrated in Fig. 2.5, the execution time of the SAT solver, although increases steadily, faces small variation. The variation in the execution time is the result of (1) random nature of SAT solver in selecting DIPs from run-to-run, and (2) rate of growth in the size of valid keys (as a result of gate conversion using the application of De Morgan's laws, as explained in section 2.3.5), compared to the rate of growth in the number of possible



(a) ISCAS-85 c5315



(b) ISCAS-85 c7552

Figure 2.5: SAT Runtime in Presence of LUT-Lock (NB2-MO-HSC).

keys. A poor selection of candidates for obfuscation results in a faster growth in the number of valid keys, reducing the overall effectiveness of obfuscated netlist against the SAT attack. As illustrated, the LUT-Lock has the least variation, as it eliminates the explosion of the set of valid keys by preventing back-to-back gate obfuscation.

Table 2.1 captures the fitted function of execution time for different sub-algorithms and LUT-Lock, where x denote the number of obfuscated gates. As illustrated in this Table, the LUT-Lock (NB2-MO-HSC) poses an exceptionally more challenging SAT problem compare to other obfuscation scheme. Table 2.2 compare the execution time of SAT attack, across

selected number of ISCAS 85 benchmarks, once obfuscated by random LUT insertion and once using LUT-Lock. As illustrated, despite random policy, the SAT execution time grows exponentially when LUT-Lock policy is adopted.

Table 2.1: Exponential Regression on SAT Runtime.

	RND	FIC	HSC	MFO-HSC	MO-HSC	NB2-MO-HSC (LUT-Lock)
Exponential Regression (Ae^{Bx})	A = 0.2065 B = 0.8875	A = 38.769 B = 0.9961	A = 15.238 B = 1.217	A = 41.252 B = 1.316	A = 38.644 B = 1.339	**A = 0.352** **B = 3.518**

Table 2.2: SAT Runtime on LUT-Lock with Different Percentages.

	1%		2%		3%		5%		10%	
Circuits	RND	Lut-Lock	RND	Lut-Lock	RND	Lut-Lock	RND	Lut-Lock	RND	Lut-Lock
c2670	0.18	0.876	0.5	1.388	0.93	1.924	2.41	24.64	3.48	time-out
c3540	0.6	1.244	1.07	6.12	2.25	988.2	2.66	time-out	5.29	time-out
c5315	0.5	9.052	1.21	115.012	1.66	941.02	3.93	time-out	12.04	time-out
c6288	0.57	23.508	2.14	1299.04	6.12	time-out	15.7	time-out	251.6	time-out
c7552	0.79	28.432	2.61	182.9	3.71	492.04	11.1	time-out	264.9	time-out

Figure 2.5 visualizes the growth in the execution time of SAT attack, for two of ISCAS-85 benchmarks obfuscated using various LUT replacement policies. Other benchmarks have similar behaviour and are omitted for lack of space. In addition to replacement policies discussed in this paper, the SAT resiliency of replacement policies in prior work, namely STT-LUT [14] and Reconfigurable barriers [8] are captured in this figure. From this figure, the SAT resiliency of prior work is close to that of random replacement, showing slow growth in SAT attack execution time with respect to the number of inserted gates, where the Lot-Lock replacement policy clearly shows a much faster exponential increase in difficulty. As illustrated, in both benchmarks, with only 20 replaced LUTs, the LUT-Lock obfuscated netlist is as resilient as the netlist produced by [14] and [8] replacement policy when using

$10X$ (200 gates) the number of gates. And by increasing the number of gates, the SAT resiliency of LUT-Lock insertion policy still grows exponentially.

Chapter 3: SAT-hard Instances via Configurable Logic and Routing

3.1 A New Perspective of SAT Hardness

A SAT attack, in each of its iterations, finds a *Discriminating Input Patterns* and rules out one or more incorrect key value(s). Hence, many SAT-resilient locking schemes tried to weaken the pruning power of one DIPs, making sure each DIP can only rule out one incorrect key. This forces the number of needed iterations to exponentially increase with respect to the number of keys as a mean of exponentially increasing the required execution time of the SAT attack, although, the execution time of each iteration of SAT solver could be quite short.

The strength of SAT solvers come from their Conflict-Driven Clause Learning (CDCL) ability. In each iteration of the SAT attack, a new SAT problem is defined. The goal of the SAT solver is to finds a satisfying value for all its literals. The literal values are either assigned or derived. Each assignment of value to a literal pushes the solver down into one of the branches of its decision tree implemented using a recursive call. During this recursive procedure, if the solver reaches a state where the derived value of a literal is different from its previously derived or assigned value, a *conflict* is detected. This is when the solver investigates how the conflict was driven, identifies a set of literal assignments that cause the conflict, and generates a clause that prevents the identified literal assignment. The newly learned conflict-clause is then added to the original problem set, aiding the solver to prune its decision tree and to avoid reaching the same conflict in the future. Then, the decision tree is backtracked a safe point prior to the conflict.

Davis-Putnam-Logemann-Loveland (*DPLL*) algorithm (or one of its derivatives), which is used to perform CDCL, is illustrated in Algorithm 2. Each SAT iteration invokes the

DPLL function. In addition, DPLL may also call itself. As it can be seen in *line* 12 and 16, new recursive call adds a new variable, l or $\bar{l}$, to Φ. Hence, an increase in the number of recursive calls (*line* 12 and 16) increases the complexity of the next DPLL call. So, the number and complexity of recursive DPLL calls could be a dominant factor for each invocation of SAT solver (a SAT Attack iteration).

Algorithm 2 DPLL Algorithm Pseudo-code

```
1:  function DPLL(Φ)
2:      if Φ has an empty clause then
3:          return "UNSAT";
4:      if Φ is [] then                                   ▷ Φ is empty
5:          SAT_assign ← Current Assignment;
6:          return "SAT";
7:      if Φ contains a unit clause l then                ▷ Unit Propagation
8:          Φ ← Φ - all clauses with l;
9:          Φ ← Φ with eliminating all l̄;
10:         return DPLL(Φ);
11:     if Φ contains a pure literal l then               ▷ Purification
12:         return DPLL(Φ ∪ l);
13:     if DPLL(Φ ∪ l) is SAT then                        ▷ Branching
14:         return "SAT";
15:     else
16:         return DPLL(Φ ∪ l̄);                           ▷ (One more level in Tree)
```

The runtime of a SAT attack could be obtain from:

$$T_{Attack} = \sum_{i=1}^{N} T(i) = \sum_{i=1}^{N} (t + T_{DPLL}(\Phi)) \tag{3.1}$$

A difficult problem requires a very large runtime. The first solution is weakening the DIP and increasing the number of iteration (N) to a very large number [11, 12, 15, 18]. In spite of very shallow DPLL recursive tree, and for having a very large N these solution exhibit resistance against SAT attack. However, this type of defense, as suggested previously is prone to SPS [12], Approximate-based [21], bypass [22], and possibly removal attack [20].

Based on the discussion on DPLL, an alternative solution is smaller N but larger recursive trees. Hence, as illustrated in equation 3.2, the attack time could also increase beyond

acceptable if the number of recursive calls (M) grows to a very large number.

$$T_{Attack} = \sum_{i=1}^{N}(t + T_{DPLL}(\Phi)) \simeq \sum_{i=1}^{N}\sum_{j=1}^{M}(T_{DPLL}^{Avg}) \simeq MN \times T_{DPLL}^{Avg} \tag{3.2}$$

The very strong aspect of this form of building SAT-hard solutions is that (1) the problems posed at each iteration of SAT attack is a SAT-hard problem, (2) the output corruption of this methods is significantly higher than obfuscating solution relying on increasing the N, (3) it is not susceptible to SPS, removal or approximate attack.

Motivated from this discussion, in this paper we present the Full-Lock. Full-Lock is able to considerably and exponentially increase the number (M) and computational complexity (T_{DPLL}^{Avg}) of recursive calls in DPLL function via replacing some of the logic and routing in the circuit by one or more SAT-hard obfuscation instance(s) in the circuit.

3.2 Full-Lock

Many SAT-hard problems (instances) are introduced annually in SAT competition. These problems aim to trap *Davis-Putnam-Logemann-Loveland* (*DPLL*) or generate extremely complex and time-consuming computational models for this algorithm. Although, none of them is directly convertible to a logic circuit, feature and tricks used in these SAT-hard problems could be used in designing SAT-hard circuit (SATC) obfuscation problems.

In [1], the SAT hardness of formulas produced using fixed-length clause generator was investigated. This work concluded that "*For formulas that are either relatively short, in which the number of clauses per variable is less than 3, or relatively long, in which the number of clauses per variable is larger than 6, DPLL finishes quickly, but the formulas of medium length, between 3 to 6, take significantly longer*". This is because formulas that have few clauses are *under-constrained*, and have several satisfying assignments. Providing under constrained clauses to the algorithm 3.1 increases the chances of one satisfying assignment

to be found early in the search using *unit propagation* or *purification*. Note that these two steps of DPLL algorithm are used to simplify the size of formula before *branching*, while *branching* assigns a value to an unassigned variable, making the DPLL tree one level deeper. Formulas that have many clauses on the other hand are *over-constrained*. In over-constrained clauses, the contradictions are found easier and the search is quickly concluded.

SAT hardness of medium length formulas is higher than under or over-constrained formulas. This is because they only have relatively few (if any) satisfying assignments. Hence, throughout the search and after assigning values to many variables, many empty clauses will be generated. This results in a deep DPLL recursive tree for testing each assumption [32]. Fig. 3.1 demonstrates the number of recursive calls made by DPLL for solving the formula for fixed-length 3-SAT formulas, where the ratio of clauses to variables is varied from 2 to 8. As illustrated, the ratio from 3 to 6 provides much higher DPLL calls, and 4.3 clauses per variable is the best ratio, generating the most computational challenging SAT instances with the highest number of DPLL calls. For example, a 100-variable 300-clause instance (clause/variable = 3 "under-constrained"), or a 100-variable 5000-clause instance (clause/variable = 50 "over-constrained") is easily solvable within few seconds. However, the SAT solver takes a very long time solving a 3-SAT instance which is constructed with 100 variables and 450 clauses. From this discussion, an obfuscated circuit is SAT-hard when its *Conjunctive Normal Form (CNF)* has medium-length clauses with a ratio of clauses to variables between 3 to 6 (best if close to 4)

3.2.1 Logarithmic Networks for SAT-Hardness

Table 3.1 lists the Tseytin transformation [34] of various logic gates into their respective CNF expression. From this table, only XOR/XNOR and MUX have 4 clauses per gate. This is when the clauses to variables ratio is 1 and 4/3 in MUX and XOR/XNOR respectively. In spite of the observation that for a single gate the XOR/XNOR has a larger clause to variables ratio, MUXes provides a better building block for constructing SAT-hard circuits. This is because: (1) with no unit propagation and purification, for having four variables,

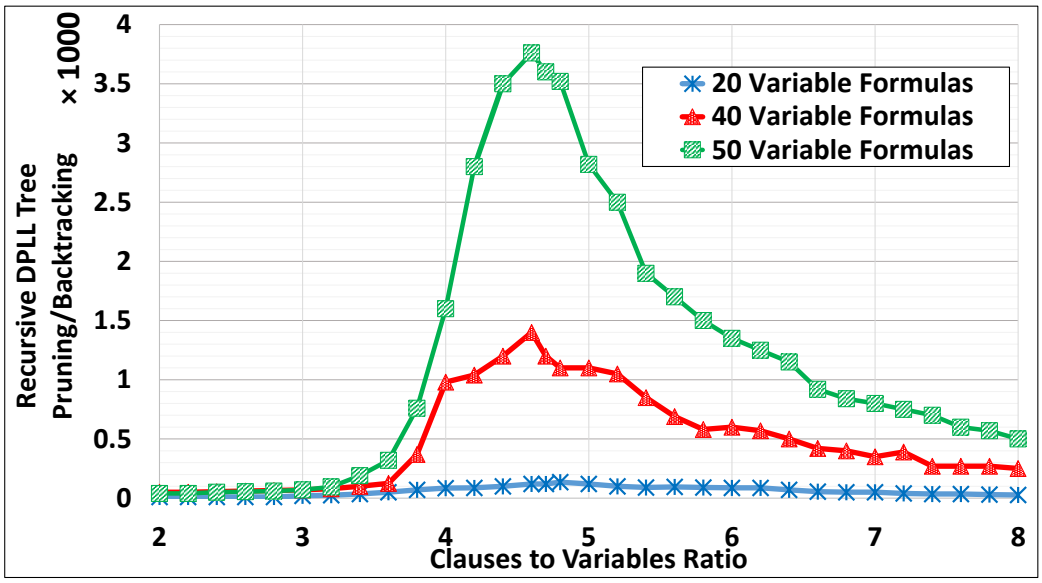


Figure 3.1: Median Number of Recursive DPLL Tree Pruning/Backtracking for Random 3-SAT Formulas [1].

Table 3.1: Tseytin Transformation of Basic Logic Gates.

Gate	Operation	CNF (sub-expression)
$C = \text{AND}(A,B)$	$C = A.B$	$(\overline{A} \vee \overline{B} \vee C) \wedge (A \vee \overline{C}) \wedge (B \vee \overline{C})$
$C = \text{NAND}(A,B)$	$C = \overline{A.B}$	$(\overline{A} \vee \overline{B} \vee C) \wedge (A \vee \overline{C}) \wedge (B \vee \overline{C})$
$C = \text{OR}(A,B)$	$C = A + B$	$(A \vee B \vee \overline{C}) \wedge (\overline{A} \vee C) \wedge (\overline{B} \vee C)$
$C = \text{NOR}(A,B)$	$C = \overline{A + B}$	$(A \vee B \vee C) \wedge (\overline{A} \vee \overline{C}) \wedge (\overline{B} \vee \overline{C})$
$C = \text{BUFF}(A,B)$	$C = A$	$(A \vee \overline{C}) \wedge (\overline{A} \vee C)$
$C = \text{NOT}(A,B)$	$C = \overline{A}$	$(\overline{A} \vee \overline{C}) \wedge (A \vee C)$
$C = \text{XOR}(A,B)$	$C = A \oplus B$	$(\overline{A} \vee \overline{B} \vee \overline{C}) \wedge (A \vee B \vee \overline{C}) \wedge (A \vee \overline{B} \vee C) \wedge (\overline{A} \vee B \vee C)$
$C = \text{XNOR}(A,B)$	$C = \overline{A \oplus B}$	$(\overline{A} \vee \overline{B} \vee C) \wedge (A \vee B \vee C) \wedge (A \vee \overline{B} \vee \overline{C}) \wedge (\overline{A} \vee B \vee \overline{C})$
$C = \text{MUX}(S,A,B)$	$C = A.\overline{S} + B.S$	$(S \vee \overline{A} \vee C) \wedge (S \vee A \vee \overline{C}) \wedge (\overline{S} \vee \overline{B} \vee C) \wedge (\overline{S} \vee B \vee \overline{C})$

a MUX can make the recursive DPLL tree one level deeper, (2) unit propagation and purification steps in DPLL algorithm provide more simplified and smaller formula using enhanced Gaussian elimination while the contribution of XOR/XNOR gates are much higher [35]. Hence, MUXes needs more DPLL recursive tree prunings/backtrackings compared to XORs/XNORs. Moreover, since unit propagation and purification satisfy less formula, the clause to variable ratio will increase while MUXes have more contribution.

The next step for building a SAT hard problem, and to push the clause to variable ratio to the desired range of 3 to 6 (4.3 as the best), is preventing the propagation and

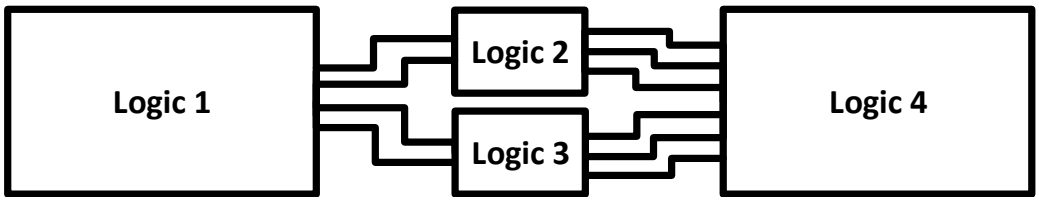


Figure 3.2: N-by-M switch-boxes for Building Hard Satisfiable Instances [2].

purification from simplifying the circuit before branching into recursive DPLL tree. This could be achieved by building a switching network using MUXes, where none of the variable related to a given MUX in a switching network could be resolved, unless their cascaded variables (related to cascaded MUXes in the original circuit) are resolved, a requirement that is recursively continued. This would prevent purification and simplification prior to reaching the leaves of the decision tree, as each variable in an intermediate layer of switching network is cascaded, while pushing up the clause to variable ratio to the desired range. This is consistent with the finding in the [37], in which investigating Boolean formulations of global detailed interconnect constraints, authors concluded that the CNF of symmetric switching networks is a hard problem for SAT solvers. In addition, using N-by-M switch-boxes, with back-to-back interconnection, illustrated in Fig. 3.2 creates hard satisfiable instances that trap even the best solvers in hopeless regions of their solution space for a long time before a satisfying solution can be found [2].

In Full-Lock we achieve this by constructing a key-configurable logarithmic-based network (CLN) for obfuscation of routes. For this purpose, we create small and lightweight switch-boxes (SwB) that are implemented easily using only MUXes. These small and lightweight SwBs allow us to create large logarithmic switching ($log_2 N$) network to (1) increase the clauses to variables ratio using MUXes that are independently interconnected back-to-back (cascaded) to each other, and (2) benefit from the hardness of switch-boxes while the power, performance, and area overhead remains reasonable.

Across all switching networks, a set of self-routing logarithmic networks, $log_2 N$ networks, provides configurable interconnection with less overhead compared to conventional networks such as mesh or crossbar. There are numerous self-routing networks in this category, such as

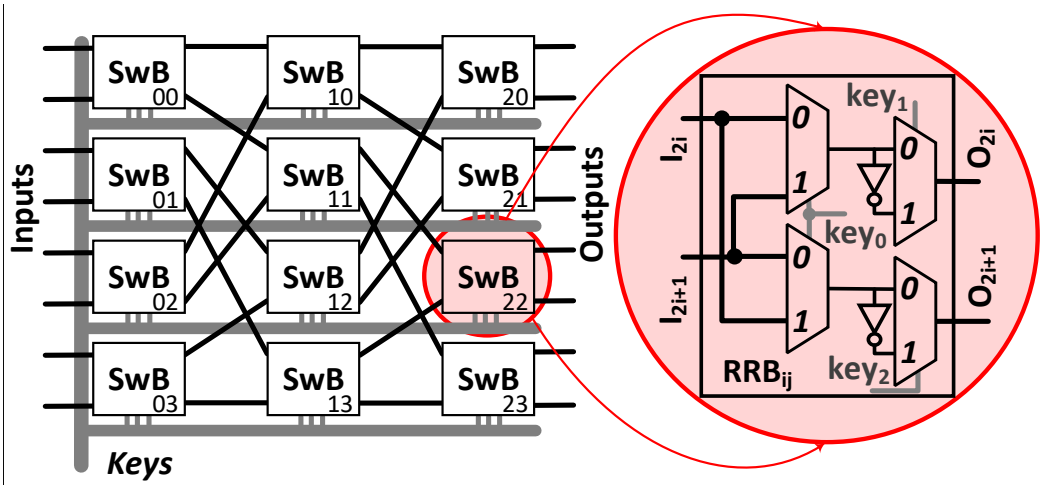


Figure 3.3: Shuffle-based Blocking *CLN* with $N = 8$.

banyan, *baseline*, *shuffle*, etc. Fig. 3.3 demonstrates a simple implementation of a 8×8 CLN using the blocking *shuffle* network [38]. This CLN is constructed using small SwBs, where each SwB is built using MUXes. In each SwB the outputs can be an arbitrary permutation of the inputs. In addition, as shown, we add key-configurable inverters for each wire, allowing an output to be shuffled and negated based on the key value. The CLN has N inputs, and due to its structure N is a power of 2. Numbers of SwBs in a CLN depends on the number of inputs as well as the model of $log_2 N$ networks. In all aforementioned blocking CLN, the number of SwBs is the same, i.e. $N/2 * logN$, and the only difference between them is the topology of SwBs interconnections.

The previously discussed self-routing logarithmic networks are blocking networks as they cannot propagate all permutations of their inputs to the outputs. In the result section of this paper, we illustrate that the blocking feature of these networks, eliminate a large number of permutations and significantly reduce the SAT hardness of these networks. This could change by building a non-blocking network.

According to [39], a non-blocking logarithmic network is characterized by $LOG_{N,M,P}$. In this equations N denotes the number of inputs/outputs, M is the number of extra (cascaded) stages, and P indicates there are $P - 1$ additional copies *vertically cascaded*. Exploration on N, M, and P shows that the minimum feasible values of P and M, which

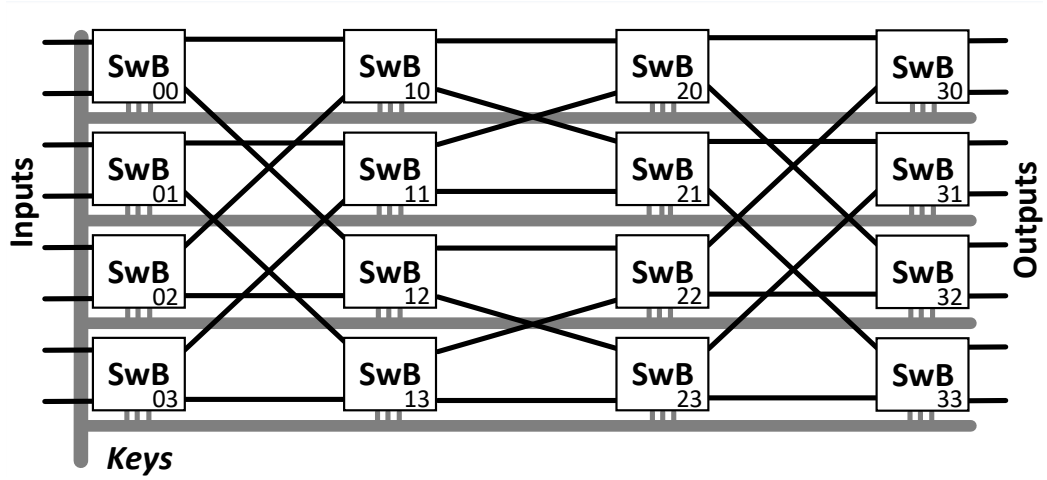


Figure 3.4: Almost Non-Blocking CLN, $LOG_{8,1,1}$.

makes the network strictly non-blocking, results in constructing a much larger network than a blocking CLN. As an instance, for $N = 64$, these values are $M = 3$ and $P = 6$. It means that a $LOG_{N,M,P}$, with $N = 64$, has more than 5× area overhead compared to a blocking CLN with the same input size, i.e. $N = 64$.

To substantially increase the permutations possibilities without incurring large area overhead, we used the near non-blocking logarithmic network suggested in [39] for constructing a key-Configurable Logarithmic-based Network (CLN). This network is able to generate not all, but *almost all* permutations, while it could be implemented using a $LOG_{N,log_2(N)-2,1}$ configuration, meaning it has only $log_2(N) - 2$ extra stages and no additional copy. Fig. 3.4, demonstrates an example of such an almost non-blocking CLN with $N = 8$. As it can be seen, the topology of SwBs interconnections is different with *shuffle*-based, shown in Fig. 3.3. This topology is a *banyan*-based interconnection that matches with our proposed $LOG_{N,log_2(N)-2,1}$.

Since an almost non-blocking CLN has only $log_2(N) - 2$ extra stages, its area and power overhead is roughly 2x compared to a blocking CLN with the same N. However, this almost non-blocking CLN is far more resistant against SAT attack compared to a blocking network. For example, an N=64 input non-blocking CLN allow only 5 iterations of SAT attack to be completed within $2{\times}10^6$ seconds, while the same size blocking network resist the SAT

attack for only ∼17 Seconds, or a much larger blocking network of $N = 512$ inputs (4 times the number of inputs, 16 times the area) complete 6 iterations of SAT attack in $2{\times}10^6$ seconds.

3.2.2 Strongly Twisted CLN into LUT/Logic

CLN provides an interconnect locking scheme that is able to generate a SAT-hard instance which significantly increases the execution time for each SAT iteration. However, in order to enhance this strength, and especially resist against other types of attacks, such as removal attack, we try to twist CLN into the logic of the gates around it. For this purpose, we suggest two methods. First, as was mentioned, we add key-configurable inverters within CLN. These inverters allow us to combine the CLN with the logic of the gates leading its inputs. In fact, both logic and interconnect locking is embedded into the CLN. For instance, suppose that one of the inputs of CLN is derived using an *OR* gate. So, we can change it to *NOR*, and configure the CLN to generate its negate on its corresponding output. These key-configurable inverters within CLN allow us to change the logic of the gates leading it. So, even removing CLN and finding the correct permutation provided by CLN will not generate correct functionality. In addition, since adding these inverters has no impact on simplification steps in DPLL, i.e. unit propagation and purification, the clause to variable ratio generated by CLN will not change.

Second, we replace the gates preceding the CLN with small Spin Transfer Torque- (STT)-based LUTs with the same input. Combining CLN with LUTs provide a *fully Programmable Logic and Routing blocks* (PLRs) that bears a resemblance to FPGA architecture. From SAT attack perspective, since each LUT will be translated to MUXes, for a LUT with R inputs, it adds up to R level to recursive DPLL tree. Moreover, since LUTs are directly connected to the output of CLN, these extra R level will be added to the large recursive DPLL tree of CLN. Hence, by massively increasing the size of recursive DPLL tree of CLN using small LUTs, PLR boosts the security of Full-Lock against SAT.

It should be noted that we use STT-based LUTs that are similar in functionality to

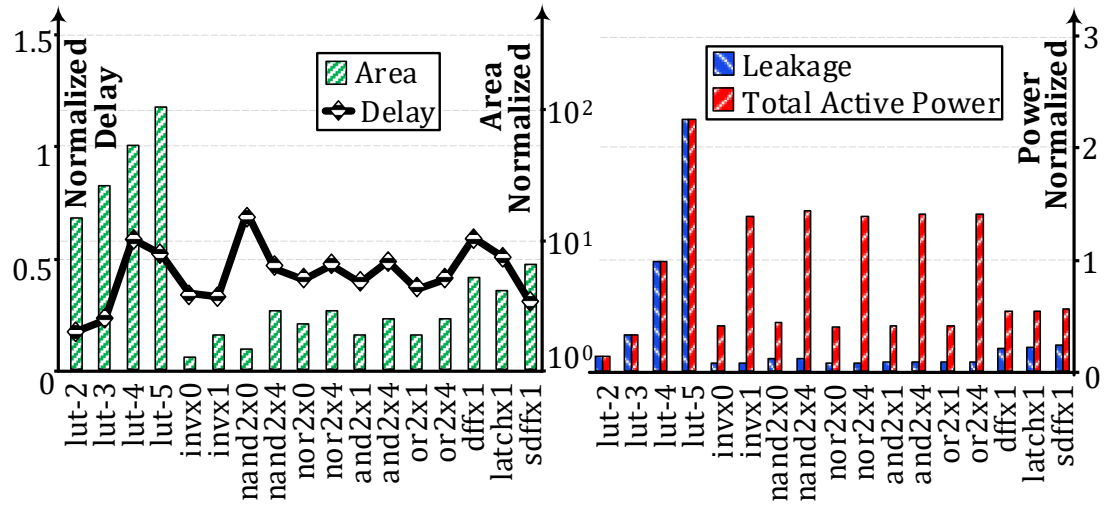
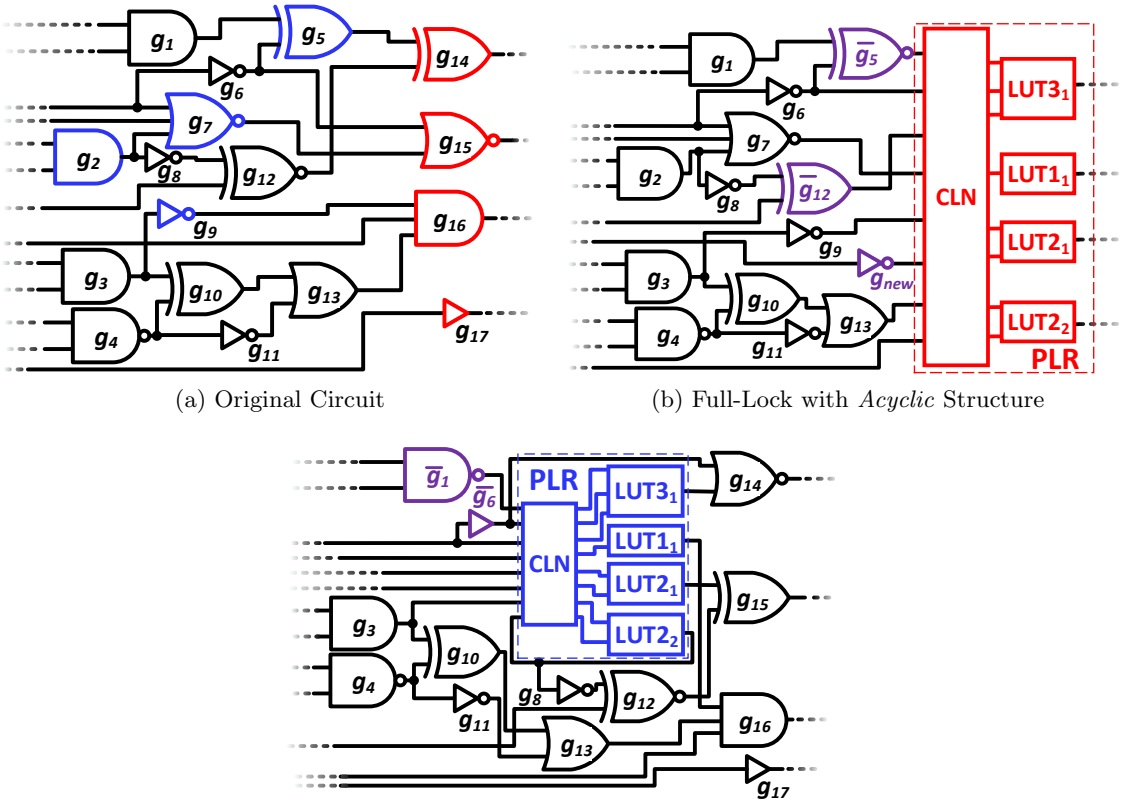


Figure 3.5: Power, Delay, and Area of STT-LUT and Standard Cells in 28nm CMOS.

FPGAs, however, they provide significantly higher speed running at GHz frequency, near zero leakage power, high thermal stability, and highly integrative with CMOS [15]. Since, each gate, located at the output of *CLN*, will be replaced with a LUT with the same input size, investigation on sizes of gates in different benchmarks such as ISCAS-85 and MCNC, shows that the maximum fan-in size is 5. It means that the largest required LUT has 5 inputs. Hence, using STT-based LUTs with a maximum size of 5 relatively has no delay overhead compared to CMOS-based basic gates. In addition, the power and area overhead is considerably low in these LUTs with size less than 5. As shown in Fig. 3.5, LUTs with size 2, 3, 4, and 5, have negligible overhead compared to CMOS-based basic gates. In addition, the size of all gates leading the CLN can be decreased to be 2. For instance, an *AND3* gate can be changed to two *AND2* while the outputs of one of them is an input for the second one. Hence, the overhead of STT-LUT can be even lower while only LUTs with size 2 is required.

3.2.3 Inserting SAT-hard *PLR*s into Design

Using these PLRs provides a big advantage compared to other locking schemes. Since inserting a PLR in a circuit provides a SAT-hard instance in the circuit, it is not required to employ a specific insertion to enhance the strength of PLRs. However, due to the



(a) Original Circuit

(b) Full-Lock with *Acyclic* Structure

(c) Full-Lock with *Cyclic* Structure

Figure 3.6: PLR Insertion Example.

topological structure of circuits it may be beneficial to have an insertion policy. But, we demonstrate that even using random insertion/replacement strategy for these PLRs creates extremely large recursive DPLL tree that makes the circuit resilient against SAT.

Additionally, in comparison with Cross-lock [27] that is a layout-based interconnect locking scheme, Full-Lock has no restriction on selection of wires and logic gates to replace them with PLRs. In Cross-lock, since they used high-density cone-based selection strategies, such as k-cut and wire-cut, to decrease the possibility of using removal attack, it has a restriction in selecting the wires to insert the crossbar. However, since we strongly twisted the CLN into the logic of the gates leading and preceding the selected wires, even removing the CLN using removal attack does not generate correct functionality. Hence, there is no limitation for wire selection in Full-Lock.

Fig. 3.6 demonstrates two simple examples that how Full-Lock inserts PLRs in the

circuit. As shown in Fig. 3.6(a) and (b), the selected gates are highlighted in red, i.e. g_{14}, g_{15}, g_{16}, and g_{17}. Since these gates have no impact on each other, replacing them with PLR, including CLN and LUTs, does not generate any cycle in the design. However, Fig. 3.6(a) and (c) show that replacing the gates, which are highlighted in blue, i.e. g_2, g_5, g_7, and g_9, generates cycle in the circuit. Additionally, some of the leading gates of CLN is changed (negated), all highlighted in purple, i.e. $\overline{g_5}$, $\overline{g_{12}}$, g_{new} in Fig. 3.6(b), and $\overline{g_1}$, $\overline{g_6}$ in Fig. 3.6(c), which shows that how twisting leading gates into CLN is working. For instance, g_5 in Fig. 3.6(a), an *XOR*, has been replaced with $\overline{g_5}$ in Fig. 3.6(b), an *XNOR*. In this case, CLN will recover the functionality of this gate using key-configurable inverters that are embedded into CLN.

3.3 Experimental Results

To show the efficiency of Full-lock, it is evaluated using different SAT-based attacks, including SAT for acyclic [7, 40], cycSAT for cyclic [26], and AppSAT for approximate-based [21], all implemented in C++, and were run on a Dell PowerEdge R620 equipped with Intel Xeon E5-2670 2.50GHz and 64GB of RAM.

3.3.1 Blocking *vs.* almost non-Blocking CLN

As was mentioned previously, Since not all but almost all permutations can be generated using non-blocking CLN, $LOG_{N,log_2(N)-2,1}$, it is far more resistant against SAT attack compared to a blocking network, especially with less power/performance/area overhead. We evaluate a shuffle-based CLN and an almost non-blocking with different sizes using SAT. As it can be seen in Table 3.2, increasing the CLN size, exponentially increases SAT execution time for either blocking or almost non-blocking. However, the SAT execution time is at least one order of magnitude higher in almost non-blocking. In addition, SAT is not able to break almost non-blocking CLN with a size larger than $N = 64$, however, for blocking CLN, it is easily broken for all sizes less than $N = 512$.

Table 3.2: SAT RunTime on *shuffle*-based Blocking CLN.

CLN Size (N)	4	8	16	32	64	128	256	**512**
Shuffle-based Blocking CLN								
SAT Iterations	7	8	9	13	15	27	28	**TO**
SAT Execution Time $_{(Seconds)}$	0.01	0.04	0.22	1.22	17.4	154.7	2329.3	**TO**
Almost non-Blocking CLN								
SAT Iterations	14	18	25	32	**TO**	**TO**	**TO**	**TO**
SAT Execution Time $_{(Seconds)}$	0.01	0.15	2.35	79.18	**TO**	**TO**	**TO**	**TO**

***TO**: Timeout* $= 2 \times 10^6$ *Seconds*

Table 3.3: PPA and SAT-Resilience of Blocking and almost non-Blocking CLNs.

CLN	Area (um^2)	Power (nW)	Delay (ns)	SAT-Resilient
Shuffle ($N = 32$)	10.1	448.1	0.82	✗
$LOG_{32,3,1}$	17.8	2137.5	0.98	✗
Shuffle ($N = 64$)	22.8	1071.1	0.89	✗
$LOG_{64,4,1}$	**38.6**	**8451.4**	**1.06**	✓
Shuffle ($N = 128$)	50.8	2503.6	0.93	✗
Shuffle ($N = 256$)	113.6	5791.4	0.99	✗
Shuffle ($N = 512$)	**254.3**	**2308**	**1.04**	✓

Since CLNs is the main part of PLRs as a SAT-hard instance that have medium length clauses while translated to CNF, the execution time of each iteration is significantly high, particularly for large sizes that cannot be broken using SAT. For blocking CLN with size $N = 512$ and non-blocking with size $N = 64$, after $2{\times}10^6$ Seconds, the number of completed iterations in SAT is only 7 and 5, respectively. It means that, on average, each iteration at least takes 2.8×10^5 Seconds in blocking and 4×10^5 in almost non-blocking CLNs.

Table 3.3 demonstrates power/area/delay of blocking and almost non-blocking CLNs for different sizes using Synopsys generic 32nm educational libraries. As it can be seen, the incurred overhead by the smallest almost non-blocking CLN, which is resilient against SAT ($N = 64$), is approximately one-third of the smallest SAT-resilient blocking CLN ($N = 512$) in terms of power consumption. Additionally, the overhead imposed by CLN is significantly low compared to area and power of even small-scale benchmark circuits.

3.3.2 Full-Lock Security Against Various Attacks

As was mentioned previously, in Full-Lock, the gates and their driving wires will be selected randomly to be replaced with PLRs. After selecting the required wires and their leading gates, Full-lock replaces them with PLR. Furthermore, the logic of some gates leading the selected wires will be negated. One or few PLR(s) can be added into the design based on the design criteria in terms of power/area/delay or security.

Security Against SAT-based Attack

Since random insertion is implemented for inserting PLRs in Full-Lock, it may generate cycle into the design. So, cycSAT has been used instead of SAT to support having potential cycles in locked circuits. In addition, to check resiliency against approximate-based attack, the cycSAT is enabled using AppSAT to extract the approximate key and corresponding error rate. Table 3.4 shows cycSAT execution time while different numbers of PLRs with different sizes have been inserted into ISCAS-85 and MCNC benchmark circuits. As it can be seen, for all circuits, having three PLRs contain 32×32 CLNs makes all locked circuit resistant against SAT. However, for each benchmark circuit, even smaller PLRs can break cycSAT.

In order to show the SAT-hardness of PLRs, we explore different sizes/numbers of PLRs to find the smallest size and the smallest number of PLRs (the lowest power/area overhead) that is required to provide resiliency against SAT. Table 3.5 shows the best solution of Full-Lock in terms of area/power/delay for each benchmark circuits. As shown, in all benchmark circuits, Full-Lock needs smaller/fewer PLRs compared to the required numbers of crossbar in Cross-Lock. As an instance, in *apex4*, only having two PLRs with a 32×32 CLN and another PLR with a 8×8 CLN can break SAT while its timeout is set to 2×10^6 Seconds. However, for the same circuit, Cross-Lock inserts 11 32×36 crossbars to make it resilient against SAT.

In addition, in order to show that PLRs are SAT-hard instances that significantly increase the number (M) and computational complexity (T^{Avg}_{DPLL}) of DPLL calls in each SAT

Table 3.4: SAT Runtime on Full-Lock with Different Sizes of PLRs.

Circuit	16×16				32×32		
	1	2	3	4	1	2	3
c432	28.8	1506.8	TO	TO	TO	TO	TO
c499	40.7	786.2	TO	TO	TO	TO	TO
c880	34.1	847.6	TO	TO	TO	TO	TO
c1355	64.9	1158.3	TO	TO	TO	TO	TO
c1908	45.5	1022.6	TO	TO	TO	TO	TO
c2670	79.8	1766.2	11791.5	184993.6	TO	TO	TO
c3540	67.2	429.6	7924.7	TO	TO	TO	TO
c5315	66.8	887.2	5748.1	TO	TO	TO	TO
c7552	90.3	1109.4	7638.6	66808.2	273367.4	TO	TO
apex2	38.4	633.1	TO	TO	TO	TO	TO
apex4	40.1	348.9	3670.9	18539.1	58467.6	380449.5	TO
i4	55.8	1604.8	TO	TO	TO	TO	TO
i7	84.6	1330.8	TO	TO	TO	TO	TO

***TO**: Timeout* $= 2 \times 10^6$ *Seconds*

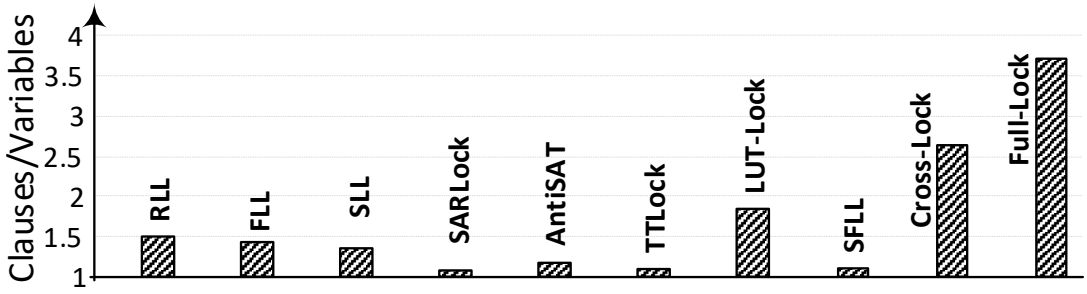


Figure 3.7: Average Clauses to Variables Ratio for Different Logic Locking Schemes.

iteration, we calculate the average clauses to variables ratio using MiniSAT for different logic locking schemes during deobfuscation. As it can be seen in Fig. 3.7, clauses to variables ratio in Full-Lock is 3.77. However, for all other methods this value is much lower. Across all logic locking schemes, LUT-Lock and Cross-Lock have higher clauses to variables ratio. Since LUT-Lock uses key-programmable LUTs for obfuscation, the translated CNF is MUX-based. However, since they have no back-to-back connection, the depth of MUX tree is low, which results in reducing the value of this ratio. The only technique with a close clauses to variables ratio is Cross-Lock, which is an interconnect locking with a tree of MUX. However, this ratio is almost 4 (3.77) in Full-Lock.

Table 3.5: PLRs Size in SAT-resilient Full-Lock compared to Cross-Lock.

Circuit	# Gates	# I/Os	Full-Lock	Cross-Lock [27]
c432	160	36/7	2×16×16 + 1×8×8	1×32×36
c499	202	41/32	2×16×16 + 1×8×8	1×32×36
c880	386	60/26	2×16×16 + 1×8×8	1×32×36
c1355	546	41/32	2×16×16 + 1×8×8	2×32×36
c1908	880	33/25	3×16×16	2×32×36
c2670	1193	157/64	1×32×32	3×32×36
c3540	1669	50/22	3×16×16 + 1×8×8	3×32×36
c5315	2307	178/123	3×16×16 + 2×8×8	3×32×36
c7552	3512	206/107	1×32×32 + 1×16×16	3×32×36
apex2	610	39/3	2×16×16 + 1×8×8	2×32×36
apex4	5360	10/19	2×32×32 + 1×8×8	11×32×36
i4	338	192/6	2×16×16 + 1×8×8	1×32×36
i7	1315	199/67	2×16×16 + 2×8×8	3×32×36

Security Against Removal Attack

As was mentioned previously, Cross-lock [27] as a layout-based interconnect locking scheme, used high-density cone-based selection strategies, such as k-cut and wire-cut, to decrease the possibility of using removal attack, which restricts in selecting the wires to insert the cross-bar. However, since the logic of the gates leading each CLN can be negated, even having the possibility of removing CLN, and finding the functionality of LUTs does not produce correct functionality, which shows that Full-Lock has no vulnerability against removal attacks.

Security Against Algebraic Attack

CLN can be expressed as an affine transformation function of the data input X, of the form $y = A \cdot X + B$, where A is an $N \times N$ matrix and B is an $N \times 1$ vector, with all elements dependent on the key input. Although recovering A and B is not equivalent to finding the key input, it may enable the successful deobfuscation of CLN. Since Full-Lock replaces the the preceding gates of selected wires with LUTs, it cannot be transformed to an affine function. So, it is safe against SAT-based algebraic attacks.

Chapter 4: Conclusion

To counter hardware security threats, such as IC overproduction, Trojan insertion, Reverse Engineering, Intellectual Property (IP) theft, and counterfeiting, logic locking, as a proactive technique, has been introduced that obfuscates and conceals the functionality of IC/IP using additional key inputs that are driven by an on-chip tamper-proof memory. Shortly after introducing the primitive logic locking solutions, a very strong Boolean attack, the Satisfiability (SAT) attack. It was shown that the SAT attack could break all previously proposed primitive locking mechanisms in almost polynomial time. To thwart the strength of SAT attack, researchers have investigated many directions, such as formulating locking solutions that significantly increase the number of required SAT iterations, or formulating the locking solutions such that it is not translatable to a SAT problem.

Recent obfuscation schemes have leveraged reconfigurable logics to alleviate various hardware security threats. In this report, first we introduced LUT-Lock, which demonstrates that how using reconfigurable logics, e.g. Look-Up-Tables (LUTs) can provide resilience against such powerful attacks, i.e. SAT attack and its derivations. LUT-Lock obfuscates a netlist while embedding several key features that make the obfuscation a hard problem for SAT attack. To develop this defense mechanism, we have identified several key features that increase the difficulty of obfuscation for SAT attacks. We illustrated how by utilizing each feature during the obfuscation, the SAT problem becomes harder. We propose LUT-Lock algorithm which combines all features, providing the best defense against SAT attacks. Then we demonstrated that how routing and logic blocks can be used for building SAT-hard solutions, which is thoroughly discussed in this report, to significantly increase the run-time of each iteration of the SAT solver. We explored the characteristics and principles of designing this SAT-hard obfuscation solutions, using reconfigurable logic and routing blocks, where the goal is to exponentially increase the time required for each

iteration of the SAT attack. As a strong representative member of this class of obfuscation techniques, we introduced Full-Lock. The Full-Lock is constructed using a set of cascaded fully programmable logic and routing blocks (PLR) networks that replace parts of the logic and routing in the desired netlist. The PLRs are SAT-hard instances designed to construct a desired ratio between the number of clauses and the number of variables with PLRs are translated to their Conjunctive Normal Form (CNF). The cascaded and non-blocking design of PLR pushes the SAT solver's algorithm to build a very deep decision tree and to spend significant time in hopeless regions of the decision tree, causing a significant increase in each iteration of SAT attack.

Bibliography

[1] D. Mitchell, B. Selman, and H. Levesque, "Hard and Easy Distributions of SAT Problems," in *Association for the Advancement of Artificial Intelligence (AAAI)*, vol. 92, 1992, pp. 459–465.

[2] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah, "Solving Difficult SAT Instances in the Presence of Symmetry," in *Proceeding of the Design Automation confernece (DAC)*, 2002, pp. 731–736.

[3] A. Yeh, "Trends in the Global IC Design Service Market," *DIGITIMES research*, 2012.

[4] M. Rostami, F. Koushanfar, and R. Karri, "A Primer on Hardware Security: Models, Methods, and Metrics," *Proceeding of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[5] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending Piracy of Integrated Circuits," *Computer*, vol. 43, no. 10, pp. 30–38, 2010.

[6] K. Z. Azar, F. Farahmand, H. M. Kamali, S. Roshanisefat, H. Homayoun, W. Diehl, K. Gaj, and A. Sasan, "{COMA}: Communication and Obfuscation Management Architecture," in *International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019)*, 2019, pp. 181–195.

[7] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the Security of Logic Encryption Algorithms," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2015, pp. 137–143.

[8] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC Piracy using Reconfigurable Logic Barriers," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 66–75, 2010.

[9] J. Rajendran, Y. Pino, O. Sinanoglu, R. Karri, "Security Analysis of Logic Obfuscation," in *Proceeding of the Design Automation Conference (DAC)*, 2012, pp. 83–89.

[10] 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*, vol. 64, no. 2, pp. 410–424, 2015.

[11] M. Yasin, B. Mazumdar, J. Rajendran, O. Sinanoglu, "SARlock: Sat Attack Resistant Logic Locking," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2016, pp. 236–241.

[12] Y. Xie and A. Srivastava, "Mitigating Sat Attack on Logic Locking," in *Int'l Conference on Cryptographic Hardware and Embedded Systems (CHES)*, 2016, pp. 127–146.

[13] M. Yasin, B. Mazumdar, O. Sinanoglu, J. Rajendran, "Security Analysis of Anti-SAT," in *Asia and South Pacific Design Automation Conf. (ASP-DAC)*, 2017, pp. 342–347.

[14] T. Winograd, H. Salmani, H. Mahmoodi, K. Gaj, and H. Homayoun, "Hybrid STT-CMOS Designs for Reverse-Engineering Prevention," in *Proceeding of the Design Automation Conference (DAC)*, 2016, p. 88.

[15] H. M. Kamali, K. Z. Azar, K. Gaj, H. Homayoun, and A. Sasan, "LUT-lock: A Novel LUT-based Logic Obfuscation for FPGA-Bitstream and ASIC-Hardware Protection," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2018, pp. 405–410.

[16] G. Kolhe, H. M. Kamali, M. Naicker, T. D. Sheaves, H. Mahmoodi, S. M. P. Dinakarrao, H. Homayoun, S. Rafatirad, and A. Sasan, "Security and Complexity Analysis of LUT-based Obfuscation: From Blueprint to Reality," in *Proceeding of the International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[17] H. M. Kamali and A. Sasan, "Much-swift: A high-throughput multi-core hw/sw co-design k-means clustering architecture," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*. ACM, 2018, pp. 459–462.

[18] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-Secure Logic Locking: from Theory to Practice," in *Proceeding of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017, pp. 1601–1618.

[19] K. Z. Azar, H. M. Kamali, H. Homayoun, A. Sasan, "Threats on Logic Locking: A Decade Later," in *Proceeding of the Great Lakes Symposium on VLSI (GLSVLSI)*. ACM, 2019, pp. 471–476.

[20] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal Attacks on Logic Locking and Camouflaging Techniques," *IEEE Transactions on Emerging Topics in Computing*, no. 1, pp. 1–1, 2017.

[21] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately Deobfuscating Integrated Circuits," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2017, pp. 95–100.

[22] X. Xu, B. Shakya, M. M. Tehranipoor, and D. Forte, "Novel Bypass Attack and BDD-based Tradeoff Analysis against All Known Logic Locking Attacks," in *Int'l Conference on Cryptographic Hardware and Embedded Systems (CHES)*, 2017, pp. 189–210.

[23] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. sasan, "SMT Attack: Next Generation Attack on Obfuscated Circuits with Capabilities and Performance Beyond the SAT Attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, vol. 2019, no. 1, pp. 97–122, 2019.

[24] S. Roshanisefat, H. M. Kamali, and A. Sasan, "SRCLock: SAT-Resistant Cyclic Logic Locking for Protecting the Hardware," in *Proceeding of the Great Lakes Symposioum on VLSI (GLSVLSI)*, 2018, pp. 153–158.

[25] Y. Xie and A. Srivastava, "Delay Locking: Security Enhancement of Logic Locking against IC Counterfeiting and Overproduction," in *Proceeding of the Design Automation Conference (DAC)*, 2017, p. 9.

[26] H. Zhou, R. Jiang, and S. Kong, "CycSAT: SAT-based attack on cyclic logic encryptions," in *Proceeding of the International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 49–56.

[27] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, "Cross-Lock: Dense Layout-Level Interconnect Locking using Cross-bar Architectures," in *Proceeding of the Great Lakes Symposium on VLSI (GLSVLSI)*, 2018, pp. 147–152.

[28] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Full-Lock: Hard Distributions of SAT Instances for Obfuscating Circuits using Fully Configurable Logic and Routing Blocks," in *Proceeding of the Annual Design Automation Conference (DAC)*, 2019, p. 89.

[29] R. Karam, T. Hoque, S. Ray, M. M. Tehranipoor, and S. Bhunia, Swarup, "Robust Bitstream Protection in FPGA-based Systems through Low-Overhead Obfuscation," in *International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2016, pp. 1–8.

[30] G. E. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," in *Design Automation Conference (DAC)*, 2007, pp. 9–14.

[31] V. Kolla, T. Nagateja, and R. Vaddi, "Robust and Energy Efficient non-Volatile Reconfigurable Logic Circuits with Hybrid CMOS-MTJs," in *International Conference on Emerging Electronics (ICEE)*, 2016, pp. 1–5.

[32] P. C. Cheeseman, B. Kanefsky, and W. M. Taylor, "Where the Really Hard Problems Are." in *IJCAI*, vol. 91, 1991, pp. 331–337.

[33] H. M. Kamali, "Using multi-core hw/sw co-design architecture for accelerating k-means clustering algorithm," *arXiv preprint arXiv:1807.09250*, 2018.

[34] G. Tseitin, "On the Complexity of Derivation in Propositional Calculus," *Studies in Constructive Mathematics and Mathematical Logic*, pp. 115–125, 1968.

[35] M. Soos, K. Nohl, and C. Castelluccia, "Extending sat solvers to cryptographic problems," in *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2009, pp. 244–257.

[36] H. M. Kamali and S. Hessabi, "A fault tolerant parallelism approach for implementing high-throughput pipelined advanced encryption standard," *Journal of Circuits, Systems and Computers*, vol. 25, no. 09, p. 1650113, 2016.

[37] G. -J. Nam, F. A. Aloul, K. A. Sakallah, and R. A. Rutenbar, "A Comparative Study of two Boolean Formulations of FPGA Detailed Routing Constraints," *IEEE Transactions on Computers*, vol. 53, no. 6, pp. 688–696, 2004.

[38] H. S. Stone, "Parallel Processing with the Perfect Shuffle," *IEEE transactions on Computers*, vol. 100, no. 2, pp. 153–161, 1971.

[39] D.-J. Shyy and C.-T. Lea, "Log/sub 2/(N, m, p) Strictly Nonblocking Networks," *IEEE Transactions on Communication*, vol. 39, no. 10, pp. 1502–1510, 1991.

[40] S. Roshanisefat, H. K. Thirumala, K. Gaj, H. Homayoun, and A. Sasan, "Benchmarking the Capabilities and Limitations of SAT Solvers in Defeating Obfuscation Schemes," in *IEEE IOLTS*, 2018, pp. 275–280.