THE EVOLUTION OF LOGIC LOCKING:
TOWARDS NEXT GENERATION LOGIC LOCKING COUNTERMEASURES

by

Hadi Mardani Kamali

A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Electrical and Computer Engineering

Committee:

_____	Dr. Avesta Sasan, Dissertation Director
_____	Dr. Kris Gaj, Committee Member
_____	Dr. Brian Mark, Committee Member
_____	Dr. Fei Li, Committee Member
_____	Dr. Monson H. Hayes, Department Chair
_____	Dr. Kenneth S. Ball, Dean, The Volgenau School of Engineering

Date: _____ Summer 2021
George Mason University
Fairfax, VA

The Evolution of Logic Locking: Towards Next Generation Logic Locking Countermeasures

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Hadi Mardani Kamali
Master of Science
Sharif University of Technology, 2013
Bachelor of Science
K. N. Toosi University of Technology, 2011

Director: Dr. Avesta Sasan, Associate Professor
Department of Electrical and Computer Engineering

Summer 2021
George Mason University
Fairfax, VA

Dedication

This thesis is dedicated to my wife, Kimia, who has been a constant source of support and encouragement during the challenges of graduate school and life. This work is further dedicated to my parents, who have always loved me unconditionally. I am enormously grateful and indebted to them for their continuous love and support.

Acknowledgments

First and foremost, I would like to take this opportunity to express my deepest gratitude to my supervisor Dr. Avesta Sasan for all his extensive guidance, advice, and continuous support during my doctoral studies. In particular, I am truly grateful to Avesta for providing me valuable and insightful research opportunities, for his patience and understanding throughout the thesis that has set an example of excellence as a professional researcher and mentor for me. In addition, I cannot thank Avesta enough for all the influential insights and valuable experiences that I have learned from him, for his attention to details and in-depth knowledge that has helped me grow as a researcher, and for his encouragement, concern, and interest towards my success.

I would also like to thank my other committee members, Dr. Kris Gaj, Dr. Brian Mark, and Dr. Fei Li for their insightful comments and feedbacks which have helped me to address the research questions in a much broader aspect.

My special gratitude to my family, particularly my Mom and Dad for their unconditional love and constant support. The holidays would have been so lonely without them. I am enormously grateful and indebted to my lovely parents for their kindness, dedication, and the education they offered me, and for having always supported me, believed in me, and encouraged me to pursue my life-long dreams and goals.

Last but not least, heartfelt gratitude and appreciation go out to my wife, Kimia, for all his love, constant support, understanding, and patience during the challenges of graduate school and life. You have always been my constant source of support and encouragement during all days and nights of research and hard work in my academic education. You were always there caring for me making all the stressful days and nights full of joy and memory. Kimia, without your unconditional love, encouragement, and support my academic achievements and completion of this research would not have been possible.

Table of Contents

List of Tables

List of Figures

Abstract

THE EVOLUTION OF LOGIC LOCKING: TOWARDS NEXT GENERATION LOGIC LOCKING COUNTERMEASURES

Hadi Mardani Kamali, PhD

George Mason University, 2021

Dissertation Director: Dr. Avesta Sasan

The globalization of the design and implementation of integrated circuits has drastically increased, particularly in the past two decades. This is when high-tech companies try (1) to reduce the cost of manufacturing, (2) to access technology that is inclusively available by a limited number of suppliers, (3) to reduce time to market, and (4) to meet the market demand. However, it results in emerging many security threats and trust challenges. Some of these threats include Hardware Trojan insertion, reverse engineering, and IP theft.

To combat these threats, numerous *Design-for-Trust* (*DfTr*) techniques have been proposed, one of them is *logic obfuscation*, *a.k.a* logic locking. In logic locking, the designer adds post-manufacturing programmability into the design controlled by programmable values referred to as the *key*. The key value is driven from an on-chip tamper-proof non-volatile memory (tpNVM), and it will be initiated after fabrication via a trusted party.

The security and the strength of the primitive logic locking techniques have been called into question by various attacks, especially by the *Boolean satisfiability* (SAT) based attack. To thwart the SAT attack, over the past few years, researchers have investigated different directions, such as point function techniques, cyclic-based locking, and behavioral logic locking. However, many of them are vulnerable to newer attacks.

The main aim of this thesis is to open a new direction as a means of logic locking. Unlike almost all previous logic locking solutions that rely on XOR-based locking, we will investigate and evaluate non-XOR-based logic locking solutions, including LUT-based and MUX-based logic locking. We first introduce LUT-Lock as a LUT-based logic locking technique, which relies on some heuristic placement strategies. LUT-Lock is resilient against the existing attacks, especially the SAT attack. However, our comprehensive design space exploration on LUT-based logic locking shows its inefficiency (in terms of overhead) compared to other techniques making this form almost impractical.

Then, we introduce Full-Lock as a new MUX-based routing locking solution. We show how MUX-based routing blocks could create SAT-hard instances while the overhead is considerably lower than the LUT-based locking solution. Although Full-Lock guarantees the resiliency against state-of-the-art attacks, we introduce a new attack, called *CP&SAT*, in which a satisfiability-based routing optimization will be introduced showing how routing-based locking techniques are still vulnerable. With this in mind, we introduce a security-enhanced routing locking technique, called InterLock. Interlock mitigates the weakness of existing routing-based obfuscation techniques against the proposed *CP&SAT* attack. In *InterLock*, the routing modules are *intercorrelated* with actual logic gates. Hence, since the logic is truly twisted with routing all controlled by the key, the adversary cannot convert and model the routing modules using the satisfiability-based routing optimization techniques, and then the *CP&SAT* attack is no longer applicable to them. We implement *InterLock* based on three different technologies: (1) transmission-gate (Tgate) CMOS, (2) programmable-via using anti-fuse elements (PVIA), and (3) three-independent-gate field-effect transistors (TIGFET). It helps us to provide a better illustration of the area/delay overhead of routing-based locking. We also show that by implementing in the lower level of abstraction, the area/delay overhead of *InterLock* could be even below $\sim$10% to make the design resilient against the prevailing attacks at a reasonable area overhead.

Since the availability of design-for-testability (DFT) structure, i.e. scan chain pins, is a mandatory requirement of the powerful SAT attack or its derivatives, we also take a step further, and by introducing *SCRAMBLE*, we evaluate the possibility of using MUX-based routing blocks as a means for locking the DFT. By locking the DFT structure, the SAT attack fails to be applied on SCRAMBLE-locked circuits. We also investigate the modeling/mapping of the logic using small-size memories optimized using the input-multiplexing technique. We will show how the integration of logic in memory and routing blocks will resist different de-obfuscation attacks at low overhead with no test compromising.

Apart from locking the DFT structure, we also propose a key-trapped design-for-security (kt-DFS) architecture, which is a DFT blockage mechanism that limits/blocks any unauthorized access to the scan chain. DFT blockage techniques provide resiliency against a wide range of de-obfuscation attacks at lower overhead compared to DFT locking techniques. In kt-DFS, we introduce a new scan chain secure cell, which is designated for safeguarding the logic locking key against any form of key leakage. We will evaluate and compare kt-DFS with other state-of-the-art logic-locking-oriented DFS architectures in terms of overhead, test coverage, and leakage.

Chapter 1: Introduction and Motivation

The cost of building a new semiconductor fabrication site was estimated to be $5.0 billion in 2015, with large recurring maintenance costs, and sharply increases as technology migrates to smaller nodes. To reduce these costs, most of the manufacturing and fabrication is pushed offshore. Over the past two decades, the ever-increasing outsourcing of different stages of the integrated circuits (ICs) has primarily changed the supply chain. Outsourcing and the involvement of numerous stakeholders in different stages of the supply chain dramatically reduce the cost and time-to-market of the chip [8]. However, as demonstrated in Fig. 1.1, getting the benefit of globalization will significantly reduce the control of original manufacturers and IP owners/vendors over the supply chain. For example, manufacturing in offshore entities, known as 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 before production, and unavoidable access to the fabricated chips during testing. This results in emerging security vulnerabilities, including but not limited to reverse-engineering, hardware Trojan insertion, counterfeiting, IP piracy, and over-production [9, 10].

During the past two decades, a wide range of design-for-trust (DfTr) techniques have been introduced, from passive to active, including watermarking, IC metering, IC camouflaging, and logic obfuscation [11–14]. 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

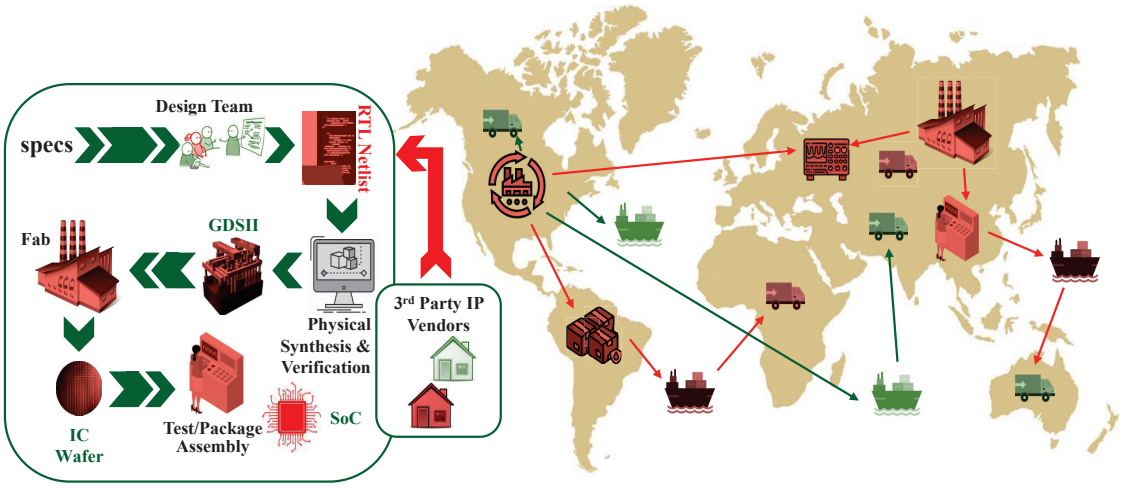


Figure 1.1: IC Supply Chain Flow.

a foundry has access to all masking layers and is not trapped by structural ambiguity for being able to logically extract a netlist.

Amongst the existing countermeasures, logic obfuscation, *a.k.a.* logic locking, could be selected as the most prominent and proactive DfTr solution that is widely accepted and studied, and it is shown that it could combat all the aforementioned threats. Logic locking provides the possibility of adding limited post-fabrication programmability into the circuit by adding/inserting some specific gates, known as key gates [14, 15]. The key gates will be driven using programming value, referred to as *key*. The key values would be initiated after the fabrication via a trusted party, and in most cases, it would be stored within a tamper-proof non-volatile memory (tpNVM) [16]. When the initiated key value is the correct one, the circuit functions correctly. However, when the key value is not correct, the functionality of the circuit will be corrupted. The main aim of inserting key gates controlled by the key values is to hide the circuit's functionality from the source of vulnerabilities (untrusted stakeholders). Hence, by using the logic locking, the original manufacturer or IP owners/vendors can regain control of the supply chain (ownership/secrets) while the source of vulnerability has no information about the correct key/functionality.

Although logic locking delivers strong protection against IP piracy and IC overproduction, during the last decade, numerous attacks have been introduced in the literature that shows the vulnerability of logic locking when the technique is not well-designed. The most well-known attack on logic locking is the satisfiability-based attack on logic locking, known as the SAT attack. Getting inspired by the miter circuit used for formal verification, the SAT attack tries to find some specific inputs called *distinguishing input patterns* (DIP) which rule out the incorrect keys. The SAT attack is considered as the turning point in logic locking, and the main aim of almost all state-of-the-art logic locking countermeasures is to be resilient against this powerful attack. Also, many of the existing attacks are derivatives of the traditional SAT attack, showing the importance of this attack in this area.

In many of the existing logic locking countermeasures, the key or correct functionality of the locked circuit could be revealed by one of the existing attacks, including the SAT attack or one of its derivatives. Over time, the introduction of newer attacks is along with the introduction of a newer countermeasure that provides the resiliency against the attack. Similarly, the introduction of newer countermeasures followed by newer and more advanced attacks on the countermeasure, showing that this *cat and mouse game* has continued since the introduction of the logic locking.

Based on the key types used for logic locking, the existing approaches could be categorized into three main groups: (1) XOR-based logic locking techniques in which the XOR gates are used as the key gate. In XOR-based, XOR gates are added/inserted, and one of their inputs would be controlled by the key. (2) multiplexer- (MUX-) based logic locking techniques in which MUXes are used as the means of the locking, and in MUX-based, the selector of the MUX would be controlled by the key. (3) look-up-table- (LUT-) based logic locking techniques that use LUTs for locking purposes, and in LUT-based, the initialization (configuration) of the LUTs are considered as the programming value (key) and would be initiated via a trusted party.

Unlike most state-of-the-art logic locking techniques that are XOR-based techniques, in this thesis, we will leverage a new breed of logic locking, which relies on the exploiting of

look-up-tables (LUTs) and key-based multiplexer- (MUX-) based locking. As a part of this research, in Chapter 3, we first introduce LUT-Lock [17], which is new LUT-based locking technique. LUT-lock locks a netlist while embedding several key features that make the LUT-based locking a hard problem for the state-of-the-art attacks with particular attention to the SAT attack. To develop this defense mechanism, we have identified several key features that increase the difficulty of logic locking for SAT attacks. We illustrate how by utilizing each feature during the logic locking, the SAT problem becomes harder. We propose LUT-Lock algorithm which combines all features, providing the best defense against SAT attacks. We also provide a comprehensive analysis on LUT-lock by investigating three key design factors: (1) LUT size, (2) number of LUTs, and (3) replacement strategy as they have a considerable influence on design criteria, i.e., Power-Performance-Area (PPA) and Security (PPA/S).

Then, in Chapter 4, we introduce and explore the characteristics and principles of designing a new direction of SAT-hard logic locking solutions, which rely on MUX-based routing locking. In this new direction, the goal is to exponentially increase the time required for finding each DIP in the SAT attack. As a strong representative member of this class of logic locking techniques, we introduce *Full-Lock* [18]. Full-Lock is constructed using a set of cascaded fully programmable logic and routing blocks (referred to as the PLR) networks that replace parts of the logic and routing in the desired netlist. The PLRs are SAT-hard instances designed to construct the 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 the SAT attack.

Although Full-Lock provides a high resiliency against the existing attacks, particularly the SAT attack and its derivatives, the existence of routing optimization algorithms, particularly satisfiability-based routing optimization algorithms, shows why solely focusing on the MUX-based routing locking might not be enough as the means of logic locking. Hence, we

investigate the possibility of attacking routing-based locking techniques, and in Chapter 5, we introduce a new attack called canonical-and-prone-based SAT (CP&SAT) attack [19], in which cardinality constraint formalization has been engaged as a pre-processing technique to optimize and simply key-based routing modules in a routing-based locked circuit. In the CP&SAT attack, after simplifying the routing modules using cardinality constraint, the SAT attack would be invoked with no scalability/complexity issue.

To combat against our proposed CP&SAT attack, we extend the structure of routing-based locking introduced in Full-lock to build a new logic locking technique that (1) could get the benefit of complexity/hardness provided by MUX-based routing locking, and (2) could be resilient against simplification/optimization used in CP&SAT. Thus, in Chapter 6, we introduce Interlock [19], which is an intertwined routing and logic locking technique, in which the logics are embedded within the key-based MUX-based routing modules. We also evaluate the implementation overhead of Interlock at the transistor-level with three different technologies to show the efficiency of this new approach at lower overhead.

Since the SAT attack and its derivatives like CP&SAT attack requires to have access to the design-for-testability (DFT) structure of the IC, i.e. scan chain pins, in Chapter 7, we propose SCRAMBLE [20], which is a comprehensive augmentation model for logic locking implemented using MUX-based routing blocks. The main aim of SCRAMBLE is to engage routing module and in-memory computation for locking the DFT structure. After locking the DFT structure using SCRAMBLE, the SAT attack, and all other de-obfuscation mechanisms that require DFT access will fail to recover the correct functionality of the locked IC. Similar to Interlock which engages twisted logic and routing locking, SCRAMBLE employs small-size memories for logic concealment. Integrating in-memory and routing-based locking schemes provides robustness at a much lower overhead.

Instead of scan-based logic locking mechanisms, more recent studies have focused on lightweight design-for-security (DFS) architectures that block the DFT access for unauthorized access. The scan blockage techniques mostly consist of two main parts: (1) a secure scan chain cell that securely stores the logic locking key when it is loaded into the circuit

after power ON, and (2) a blockage circuitry which blocks scan pins while there exist the possibility of key leakage. Although the existing scan blockage techniques provide robustness at much lower overhead compared to the scan-based logic locking techniques, in Chapter 8, we will investigate the architectural drawbacks of these schemes. Then, we propose a new key-trapped DFS (kt-DFS) structure, to overcome such limitations and drawbacks. We demonstrate that the scan-blockage techniques, once implemented appropriately, can provide robustness with no compromising at much lower overhead compared to other logic locking countermeasures.

To conclude this thesis, in Chapter 9, we will discuss the current status of different breeds of logic locking, including LUT-based, routing-based, scan-based, and scan blockage. We conclude this thesis by drawing the future outlook for these categories of logic locking. Regarding LUT-based, we demonstrate how the state-of-the-art models could be extended to not only mitigate the overhead of this category and make this group practical in real cases, it also enhances the robustness against the existing attacks. Regarding the routing-based on the other hand, for deeper scrutiny and wider assessment, it is required for the countermeasures of this group to be evaluated against more attacks. In terms of scan-based countermeasures, either locking or blockage, we evaluate the pros and cons of the existing approach to determine how it still could be extended in future studies.

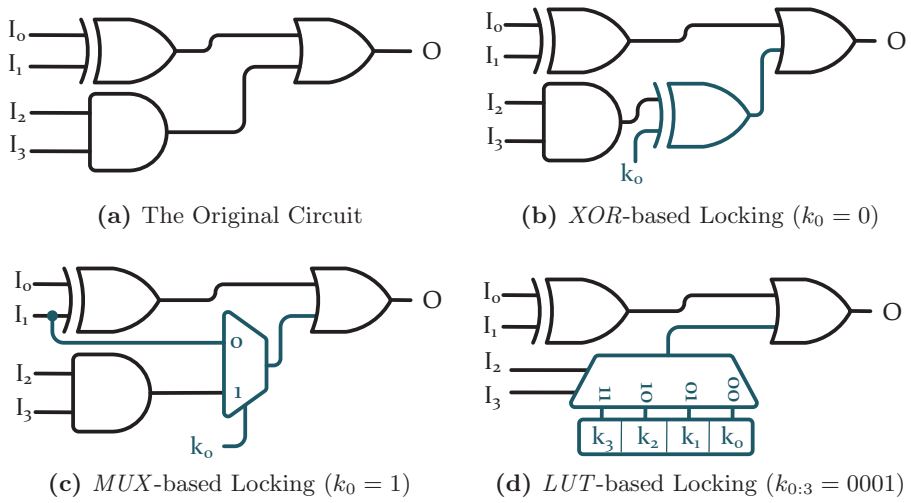
Chapter 2: Definitions, Background, and Previous Work

In this chapter of this research, we first provide the basic definitions of logic locking. Based on the metrics, characteristics, and assumptions defined in this section, the logic locking countermeasures and attacks could be categorized and evaluated. Then it is followed by the different logic locking groups, their characteristic, and their advantages/disadvantages. Since a plethora of this research focuses on utilizing reconfigurable logic/routing using LUTs/MUXes, we conclude this chapter by reviewing the previous work on LUT-based and routing-based locking techniques.

2.1 Basic Definitions of Logic Locking

Logic locking is the capability of adding post-fabrication programmability using key gates. Based on the type of the key gates used for logic locking, we can categorize them as: (1) XOR-based, (2) MUX-based, and (3) LUT-based. As their names imply, they are using XORs/MUXs/LUTs for locking, respectively. Fig. 2.1 depicts a simple example of each of these models. During the last decade, different logic locking solutions commonly have engaged these models with different structures/functions for locking purposes. Based on the location, structure, count, intercorrelation, etc., of these gates, the countermeasures provide different levels of resiliency against the existing attacks.

Logic locking could be implemented at different levels of abstraction. Fig. 2.2 demonstrates a simple example of logic locking in different levels of abstraction. For instance, at layout-level as shown in Fig. 2.2(a), the metal-insulator-metal (MIM) structure that connect two adjacent metal layers has been engaged as key-based programmable unit for routing-based locking [21]. In general, moving from layout-level to HLS- or architecture-level will mitigate implementation effort; However, at a lower level of abstraction, finding



(a) The Original Circuit

(b) *XOR*-based Locking ($k_0 = 0$)

(c) *MUX*-based Locking ($k_0 = 1$)

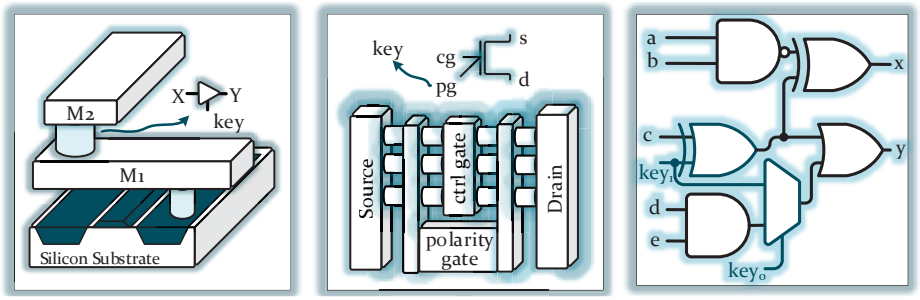
(d) *LUT*-based Locking ($k_{0:3} = 0001$)

Figure 2.1: Common Basic Gates used for Logic Locking.

a solution at lower overhead is more possible. At the moment, more than 90% of existing logic locking techniques are introduced and implemented at the gate-level, which could be done as a post-synthesis stage in the supply chain.

One important property of logic locking techniques is the *corruptibility*. Corruptibility means that when an incorrect key will be applied to the locked circuit, the value of how many of the primary output (PO) pins will be corrupted. Based on the location, structure, count, intercorrelation, etc., of the key-based *XOR*s/*MUX*s/*LUT*s that are engaged for any logic locking technique, the corruptibility will change. Corruptibility directly affects the resiliency of the countermeasure against the existing attacks. For instance, if the corruptibility is low, which means that an incorrect key only affects a few numbers of PO pins, it allows the adversary to look for a specific way for only those POs affected by logic locking. For a well-design logic locking countermeasure, the corruptibility must be high to avoid such vulnerabilities.

The key of logic locking will be initiated and stored in tpNVM after the fabrication via a trusted party. At power UP of a locked IC, as a part of the boot process per each power UP, the content of tpNVM must be read and loaded into temporary registers connected to the locked logic. Fig. 2.3 shows a simple example of key initialization structure when logic



(a) Layout-level (b) Transistor-level (c) Gate-level

```
case STATE is
   when "0000"  =>
      if (key[0] = '1') then
         STATE <= "1010";
      else STATE <= "0110";
   when "0001" =>
      if (key[1] = '0') then
         STATE <= "1110";
      else STATE <= "0111";
   when "0110" =>
      if (key[2] = '1') then
         STATE <= "1011";
      else STATE <= "0010";
```

(e) RTL-level

```
void shift_reg
(dp <4> in, cp <4> key, dp <4> out) {
   volatile reg<4> shift_temp;
   for (i = 4 ; i>0 ; i--) {
      #pragma HLS unroll
      shift_temp[i-1] =
               shift_temp[i] ^ key[i]
}
   out[3] = shift_temp[3];
   out[2] = shift_temp[2];
   out[1] = shift_temp[1];
  out [0] = shift_temp[0];
}
```

(f) HLS-level

Figure 2.2: Logic Locking Examples at Different Level of Abstractions.

locking is in place. This part of the design consists of (1) tpNVM that consists of the logic locking key, (2) tpNVM wrapper which serializes the logic locking key via parallel-in serial-out (PISO) module, and (3) temporary registers that stores the logic locking key while the IC is power ON.

2.2 Models and Assumptions in Attacks on Logic Locking

Based on the threat model and assumptions considered in the attacks on logic locking, they could be categorized into different sub-groups [22]. Some of the attacks require to have access to one additional activated version of the chip (oracle). This group of the attacks could be referred to as *Oracle-guided* attacks. On the other hand, those attacks with no need for having access to the oracle could be called *Oracle-less* attacks. During the last decade,

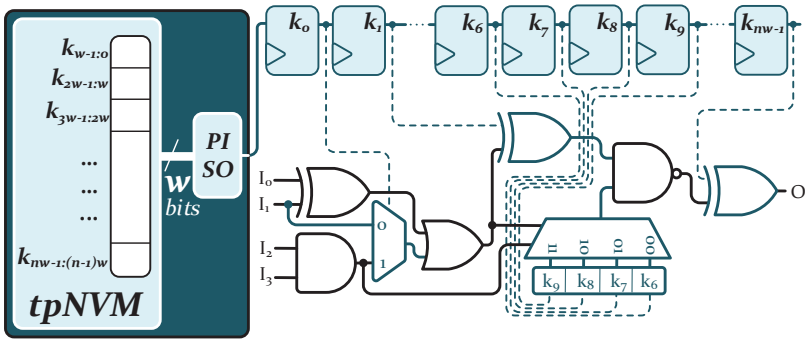
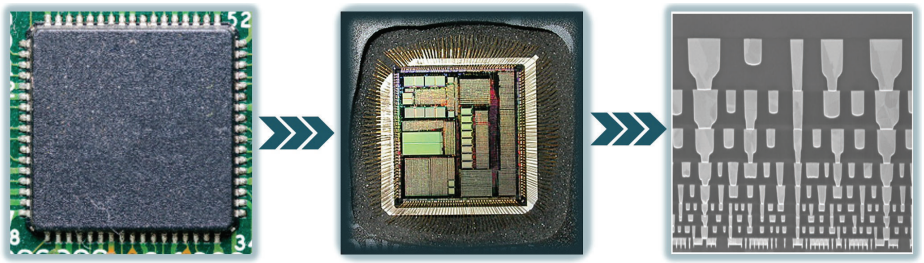


Figure 2.3: Logic Locking Key Initialization from tpNVM.

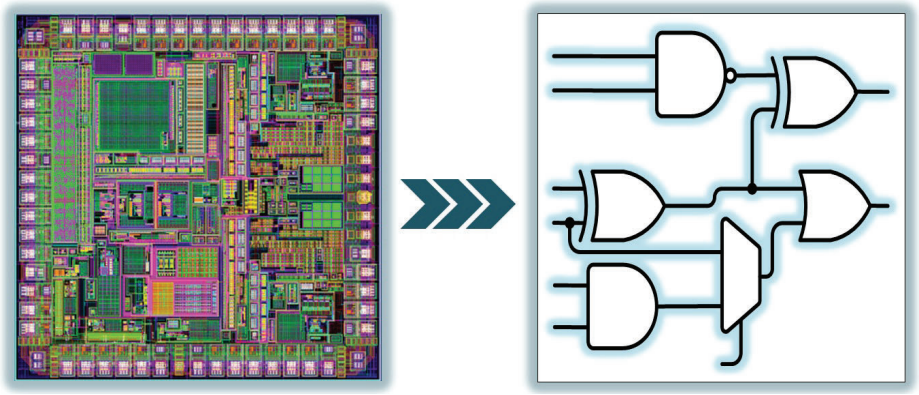
most of the attacks are members of *Oracle-guided* attacks. However, in many cases, the adversary cannot have one additional activated chip, and the fulfillment of this requirement is almost impossible. So, the adversary has to rely on only *Oracle-less* attacks.

Many of the attacks on logic locking are *invasive*. They require access to the netlist of the chip. Acquiring the netlist of the chip could be accomplished in two different scenarios: (1) The adversary is located in the foundry, and they receive the GSDII of the chip, which is locked. (2) The adversary as an end-user can obtain the packaged IC from the field/market, and then reconstructs the netlist through physical reverse engineering. Fig. 2.4 demonstrates the main steps of physical reverse engineering, including de-packaging, delayering, imaging, image (of metal layers) processing, and re-constructing the netlist. In both cases, the adversary does not have access to the logic locking key. When (s)he is at the foundry, only netlist the GDSII of the locked netlist will be available. Also, as an end-user during the physical reverse engineering, since the key is stored in tpNVM, it will be wiped out in the de-packaging stage.

Unlike *invasive* attacks, there exists a very limited number of *semi-invasive* and *non-invasive* attacks on logic locking, in which the adversary relies on electro-optical or optical probing, such as electro-optical probing (EOP) and electro-optical frequency management (EOFM). Such attacks focus on pinpointing and probing the logic gates and flip-flops of the chip. So, regardless of the logic locking technique used in the chip, this group of attacks



(a) The Packaged IC (b) After De-packaging (c) Imaging per Metal Layer



(d) Extracting Layout from Images (e) Extracting the Netlist from the Layout

Figure 2.4: The Main Steps of Reverse Engineering.

would be able to reveal the security assets like logic locking key.

The availability of scan chain architecture for testability/debug purposes in ICs opens a big door for the attackers to assess and break logic locking techniques. Hence, many of the attacks on logic locking assume that the *scan chain is OPNE*. Fig. 2.5 shows a simplified scan architecture with two chains. Assuming that the scan chain is *OPEN*, SE, SI, and SO pins would be available. So, the adversary can reach (control and observe) each combinational part, e.g. CL_1 and CL_2 in Fig. 2.5, whose FFs are part of the scan chain. The scan chain access allows the adversary to divide the attack on locked circuit into a bunch of much smaller sub-problems (each CL), and assess them independently. However, it is very common for an IC to limit/restrict access to the scan chain for security purposes. But, even while *the access to the scan chain is NOT OPEN* (e.g. SO pins are burned), some other attacks have studied and demonstrated the possibility of retrieving the correct key/functionality of the locked circuit via primary inputs/outputs (PI/PO).

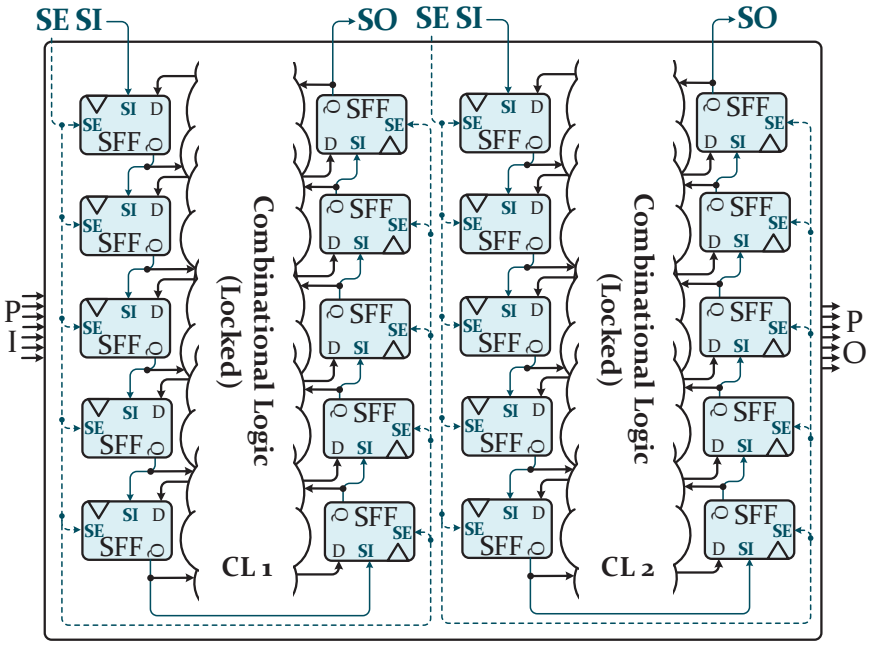


Figure 2.5: Scan Chain Architecture in ICs.

2.3 Logic Locking: Previous Countermeasures and Attacks

Starting 2008, numerous logic locking techniques have been introduced in the literature each trying to introduce a new countermeasure against the existing threats. In this section of the research, we categorize them into different groups, and briefly describe each one of them.

2.3.1 Primitive Countermeasures

The first group of countermeasures introduced in the literature are primitive techniques, including EPIC *a.k.a.* random logic locking (RLL) [14], strong logic locking (SLL) [15], fault-based logic locking (FLL) [23], and reconfigurable barriers [24]. For example, in EPIC (RLL) [14], as its name implies, XOR-based key gates will be inserted at some arbitrarily chosen location in the circuit. All primitive techniques are XOR-based and implemented at gate-level. Since a locked circuit initiated with an incorrect key corrupts the PO by propagating errors at POs, in SLL [15] and FLL [23], some features of automatic test pattern

generation (ATPG) tools and testability specification, such as controllability/observability, and faults propagation/masking have been used for selecting the location of XOR-based key gates. However, using these features results in the reduction of corruptibility in SLL and FLL compared to that of RLL. Reconfigurable barriers [24], on the other hand, investigates the possibility of using reconfigurable modules, like look-up-tables (LUTs) as a means of locking.

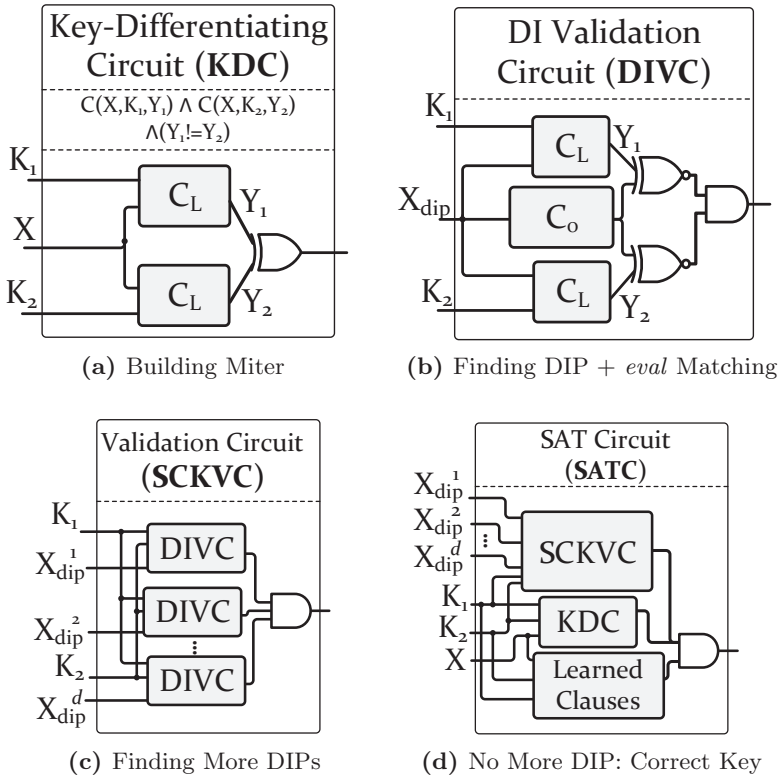
2.3.2 The SAT Attack: The Game Changer

The introduction of primitive logic locking solutions, such as RLL (random-based insertion) and SLL (no-sensitizable insertion) [14, 15, 23, 24], was considering a reliable proactive solution against hardware security threats. However, in 2015, considering and assuming that the access to the scan chain of the circuits is *OPEN* for the test/debug purposes, a new and powerful attack based on Boolean satisfiability (SAT) was formulated that completely threatened the security of the existing logic locking schemes [1, 2].

In the SAT attack, as an oracle-guided attack, the adversary has access to (1) one successfully reverse-engineered yet locked netlist, and (2) the activated/functional circuit with *OPEN* access to its scan chain architecture. By getting inspiration from the miter circuit in the formal verification process, in the SAT attack, a SAT solver is employed iteratively to rule out the incorrect keys. As shown in Fig. 2.6 and Algorithm 1, in each iteration of the SAT attack, the SAT solver finds a specific input, called discriminating (distinguishing) input patterns (DIP), that distinguish between two sets of keys, where at least one set of them is incorrect. By applying each DIP to the activated/functional circuit,

Algorithm 1 SAT Attack on Locked Circuits [1–3]

1: $SCKVC = 1$;
2: $KDC = C_L(X, K_1, Y_1) \wedge C_L(X, K_2, Y_2) \wedge (Y_1 \neq Y_2)$;
3: **while** $((X_{DI}, K_1, K_2) \leftarrow SAT_F(SAT_C) = T)$ **do**
4: $\quad Y_f \leftarrow C_O(X_{DI})$;
5: $\quad DIVC = DIVC \wedge C_L(X_{DI}, K_1, Y_f) \wedge C_L(X_{DI}, K_2, Y_f)$;
6: $\quad SAT_C = SAT_C \wedge DIVC$;
7: $KeyGenCircuit = DIVC \wedge (K_1 = K_2)$
8: $Key \leftarrow SAT_F(KeyGenCircuit)$



(a) Building Miter **(b)** Finding DIP + *eval* Matching

(c) Finding More DIPs **(d)** No More DIP: Correct Key

Figure 2.6: The SAT Attack Iterative Flow [1–3].

the adversary is able to invalidate (rule out) the incorrect set(s) before the next iteration. This process continues until the SAT solver cannot find a new DIP. At this point, any key that generates the correct output for the set of found DIP is the correct key. In general, the SAT attack could eliminate all incorrect keys within a few iterations (a few minutes), leading to retrieving the correct functionality of the circuit.

2.3.3 Post-SAT Logic Locking Countermeasures

After 2015, the introduction of the SAT attack significantly changed the direction of logic locking. Due to the strength of this technique, the main aim of almost all post-SAT countermeasures is to be resilient against the SAT attack. The main strength of the SAT attack comes from two important factors: (1) The pruning power of each DIP (each iteration of the SAT attack) is very high. In fact, the portion of incorrect keys that would be ruled out

per each iteration is big leading to termination (successful attack on logic locking) within a few iterations (few minutes). (2) The access to the scan chain is *NOT* restricted, which helps the adversary to apply the SAT attack for each combinational logic part of the circuit separately (independently).

Considering these two factors, there are two main groups of countermeasures that have been introduced in the literature to show how a logic locking technique could be built to defeat the SAT attack. One group tries to either weaken the pruning power of DIPs or introduce a solution that could not be formulated by the SAT attack (behavioral locking). However, the main focus of the second group of countermeasures, on the other hand, is to restrict any unauthorized access to the scan chain to completely invalidate the possibility of engaging the SAT the attack.

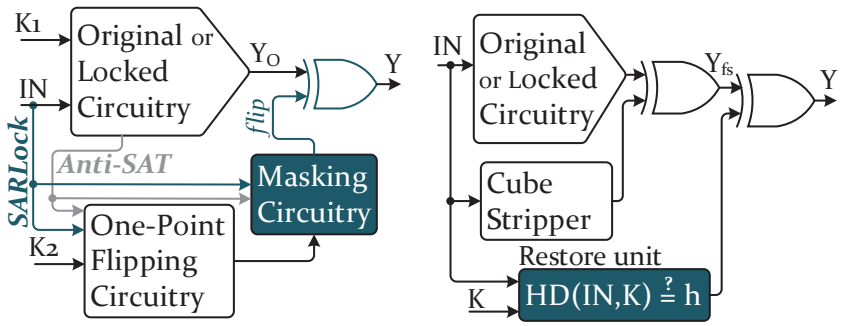
2.3.4 Weakening/Disabling the SAT Attack

As mentioned previously, the first group of countermeasures tries to weaken/disable the SAT attack. Since having access to the scan chain does not provide any advantage for the adversary in this group of countermeasures, there is no concern for the designer to leave the scan chain architecture *OPEN* [4, 25–33]. These countermeasures could be categorized into different sub-groups: (1) point-function structure, (2) cyclic logic locking, (3) and behavioral locking.

Point Function Logic Locking

The main aim of point function techniques is to minimize the number of available input patterns showing that a specific key is incorrect[1]. SARLock and Anti-SAT are the very first logic locking techniques in this category [25, 26]. As demonstrated in Fig. 2.7(a), the main structure of point function techniques relies on a *flipping* circuitry that corrupts the PO(s) only for a very limited number of input patterns (e.g. 1) per each incorrect key. Also, a *masking* circuitry has been engaged in point function techniques to re-flip the impact

[1]The best case is *ONE* input pattern per each incorrect key.



(a) Anti-SAT and SARLock **(b)** Stripped-based Point Function

Figure 2.7: The Structure of Point Function Techniques.

of flipping circuitry for the correct key to guaranteeing the correct functionality when the correct key is applied. Also, these techniques could be combined with primitive techniques, called *compound* techniques. For instance, as shown in Fig. 2.7(a) the upper part could be the original or locked version of the circuit. In case of locked, it is suggested to be locked primitive techniques.

Point function techniques could be applied on stripped functioned version of the circuit, known as stripped function logic locking [27]. In such techniques, the original/locked part is modified, and in at-least one minterm, the (affected) POs are flipped. This is done using *cube stripper* module as demonstrated in Fig. 2.7(b). The *restore* unit builds the flipping and masking circuitry. In the stripped function techniques, for each incorrect key, there exists a very limited number of input patterns (e.g. 1) *plus* one extra input pattern (caused by the stripped function) that corrupt the POs. Point function techniques could be categorized as XOR-based techniques. Except SFLL-HLS that is at architecture-level, all techniques in this category are implemented at gate-level.

Similar to a brute force attack, the SAT attack faces an exponential runtime when the point function is applied on the circuit. However, as demonstrated in Fig. 2.7, the part(s) of the logic added for locking purpose are completely decoupled from the original part of the circuit. Hence, these techniques suffer from various structural vulnerabilities

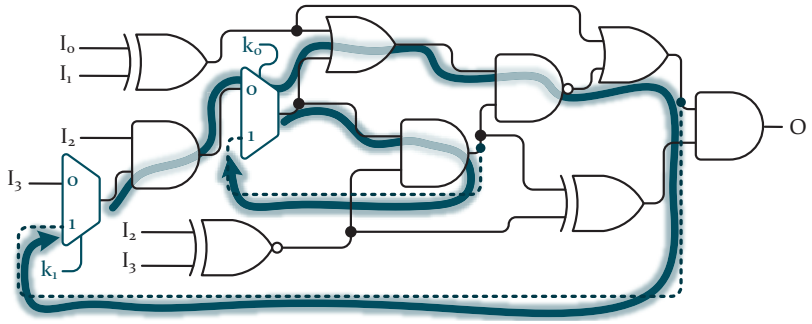


Figure 2.8: An Example of Cylic Locking using 2-to-1 MUXes.

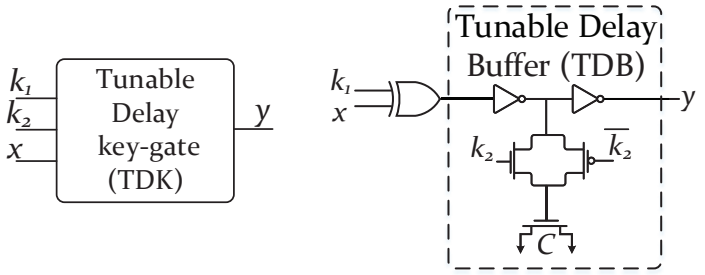
that were eventually exploited to break them. Attacks like signal probability skew (SPS), removal, bypass, and functional analysis (FALL) attack [34–37] exploit these structural characteristics of the point function techniques to break them. Also, as demonstrated in Fig. 2.7, since the point function sub-circuitry will be added for limited (e.g. 1) number of POs, the corruptibility of this breed of logic locking is very low. Due to the low output corruptibility, the error probability is exponentially small. Hence, the adversary could also rely on approximate key with an extremely low error rate, which could be found by approximate-based SAT attacks [38, 39].

Cyclic-based Logic Locking

As its name implies and as shown in Fig. 2.8, cyclic logic locking will add key gates that control the possibility of adding/removing combinational cycles into the circuit. Having combinational cycles will add difficulties for the CAD tools to deal with (like synthesis and timing analysis) such structures. Many CAD tools do not allow the designers to have combinational cycles in the circuit. However, the designers would be able to handle the combinational cyclic paths during the physical design in a manual manner, like false paths. Hence, combinational cycles are used commonly as a means of logic locking recently. Also, since the SAT attack only receives directly acyclic graphs (DAGs) as its input, for those circuits containing the cycle, the SAT attack cannot be used. Accordingly, cyclic-based logic locking received a significant attention in recent years.

Different techniques have been used for cyclic logic locking, such as (1) adding false cycles [28], (2) adding cycles as a part of the original functionality (like stateful) [29, 31], (3) exponentially increasing the number of cycles *w.r.t.* the number of feedbacks [29, 32], (4) engaging new modules and technologies for cycle generation [30], and (5) inserting cycle pairs [40]. Since re-routing is required to generate the combinational cycles, all cyclic logic locking techniques use key-based MUX gates. Also, in some cases, key-based XOR gates are used to build the model. All existing cyclic logic locking techniques are implemented at the gate-level. Since re-routing involve many logic cones in the circuit, the corruptibility of this group of logic locking would be high.

The promise of secure cyclic locking was shortly after broken by CycSAT attack [41]. In CycSAT, the key combinations that result in formation of cycles are found in a pre-processing step. These conditions are then translated into problem augmenting CNF formulas, denoted as cycle avoidance clauses, satisfaction of which guarantee no cycle in the netlist. The cycle avoidance clauses are then added to the original SAT circuit CNF and the SAT attack is executed. Also, inability to analyze all cycles in the prepossessing step of CycSAT [41] results in missing cycles in the pre-processing step of CycSAT, leading to building a stateful or oscillating circuit, trapping the SAT stage of the CycSAT attack. However, BeSAT [42] remedies this shortcoming by augmenting the CycSAT attack with a run-time behavioral analysis. By performing behavioral analysis at each SAT iteration, BeSAT detects repeated DIPs when the SAT is trapped in an infinite loop. Also, when SAT cannot find any new DIP, a ternary-based SAT is used to verify the returned key as a correct one, preventing the SAT from exiting with an invalid key. Recently, the work in [43], IcySAT, shows how explicitly banning cyclic keys as used in BeSAT can end up performing exponential computation on small and simple cyclic circuits. They propose an algorithm that can produce non-cyclic conditions in polynomial time with respect to the size of the circuit, avoiding the potentially exponential runtime of BeSAT.



k_1k_2	f(x)	Delay
00	$y = x$	d_0
01	$y = x$	d_1
10	$y = \overline{x}$	d_0
11	$y = \overline{x}$	d_1

(a) TDK Gate (b) Layout of TDK (c) TDK Specifications

Figure 2.9: Overall Structure of TDK used in DLL [4].

Behavioral Logic Locking

A SAT attack works perfectly fine if the logic locking is of Boolean nature. This is because any Boolean logic could be easily transformed into its Conjunctive Normal Form (CNF) and be converted into a satisfiability assignment problem. Hence, in a group of logic locking techniques, the locking mechanism is designed to control aspects of circuit operations that could not be translated to CNF as required by a SAT solver. The delay-locking (DLL) scheme proposed in [4] is a good instance of such locking mechanism. For the purpose of locking, DLL uses a tunable delay key-gate (TDK) which is illustrated in Fig. 2.9. TDK consists of a conventional key-gate (XOR/XNOR) with a tunable delay buffer (TDB). The Goal of DLL is introducing setup and hold violation if the correct key is not applied. In this case, the locking flow attempts to change both logical and behavioral (timing) properties. A functionally-correct but timing-incorrect key will result in timing violations, leading to circuit malfunctions.

Considering that timing is not translatable to CNF, the SAT solver remains oblivious to the keys used for timing locking. However, authors in [44] introduce a new attack called the satisfiability modulo theory (SMT) attack. As shown in Fig. 2.10, SMT attack engages the capability of SMT solvers, in which a combination of theory solver and SAT solver could

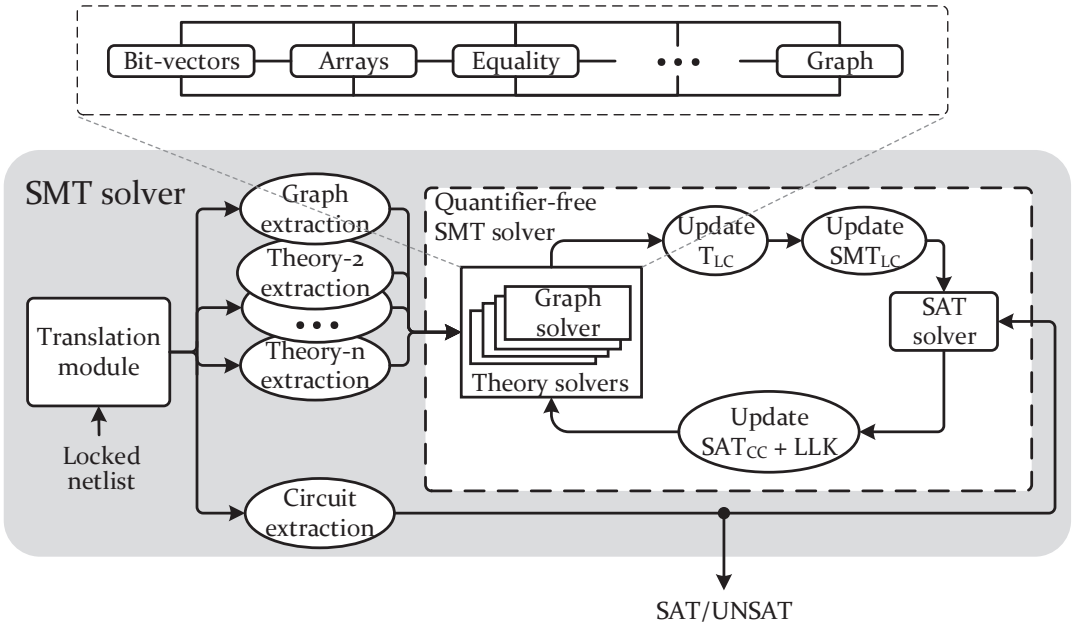


Figure 2.10: Overall Architecture of SMT Attack for Behavioral Logic Locking.

be used for solving much more complex problems. The SMT attack could easily deploy a graph theory solver, provide timing constraints to the theory solver (in terms of required min and max delay to meet the hold and setup time), and use the theory solver in parallel with the internal SAT solver to break both logic and delay locking. They additionally show that the theory solver could be initiated as a pre-processor (Eager SMT approach) or as a co-processor (Lazy SMT approach) to break the same problem, showcasing the strength of SMT attack. Similarly, TimingSAT is another attack on DLL [45], in which similar to many prior SAT-based attack, it is deploying a pre-processor for analysis of graph timing, and generating helper clauses for the subsequent call to the SAT attack.

2.3.5 Restricting Unauthorized Scan Chain Access

Unlike all previously discussed countermeasures that focus on combinational part(s) of the circuit and assume that the scan chain is available for the adversary, some other techniques focus on protecting the key instead of the protecting the design [46, 47]. A very specific

group of this category is trying to secure scan architecture []. Restricting the access to the scan chain could be accomplished by limiting/blocking the access to the scan chain pins, i.e. scan-enable (SE), scan-in (SI), and particularly scan-out (SO). This restriction has been evaluated in two different ways: (1) scan chain locking, and (2) scan chain blocking.

Scan Chain Locking

Scan chain locking techniques directly target the scan chain structure to be locked. By using these techniques, the scan chain pins, SE/SI/SO would be out of access, and the adversary loses the chance of direct/independent controlling/observing the combinational parts of the circuit. Some of the scan chain locking techniques lock the scan structure statically [20, 48–50]. However, there also exist some techniques which dynamically lock the scan chain using LFSR and/or PRNG [51–53]. Unlike all combinational techniques that could be done in all abstraction layers, since scan chain locking techniques must be applied on the scan chain structure, they only could be done after design-for-testability (DFT) synthesis. So, these techniques could be implemented at gate-level, transistor-level, or layout-level. All the existing techniques are implemented at gate-level. Since the scan locking techniques only lock the scan chain structure, even while an incorrect key is initiated in the circuit, it only affects the scan chain functionality and has no impact on the functionality of the circuit at POs. Hence, to add the desired ambiguity to the circuit, they need to be combined with one of the previously discussed combinational logic locking techniques. In many cases, they are combined with primitive techniques.

Scan Chain Blocking

Similar to scan chain locking techniques, some approaches $BLOCK$ the access to the scan chain pins, particularly SO [54–59]. The blockage will happen based on a sequence of specific operations in the scan chain structure. For example, assuming that (part of) the key is loaded into the scan chain after activation, switching SE to 1 (shift mode) might be perilous. Hence, shift operation would be limited after the activation. Compared to

scan chain locking techniques, they incur less area overhead. However, they have some limitations during the test phase, such as limiting the functional test and increasing the test time and complexity. Also similar to scan chain locking techniques, they need to be combined with one of the previously discussed combinational logic locking techniques. Some techniques use RLL with high corruptibility [57], some other techniques use SLL [54, 58], and one scan blockage technique is combined with true-RLL (TRLL) [59].

Attacks on Scan Chain Locking/Blocking

Shortly after the introduction of the primitive studies on scan chain locking and blocking, new derivatives of the SAT attack demonstrate the feasibility of breaking these schemes, called *sequential SAT attacks* [60–62]. Sequential SAT attacks first try to convert each circuit to its combinational counterpart and then apply the SAT attack. This prepossessing step could be done using unrolling with a specific depth (e.g. u times), or using a bounded-model-checker (BMC). So, for any given u, SAT solver will find u input patterns, denoted as discriminating input *sequence* (X_{DIS}), that have at least one difference at one of u output vectors with two different keys. This process continues until no further X_{DIS} is found within the boundary of b. After reaching the boundary, the algorithm checks three criteria (Unique Completion (UC), Combinational Equivalence (CE), and Unbounded Model Check (UMC)) to determine if the attack can be terminated [60].

The sequential SAT was first introduced in [60] that is only applicable to static-based scan locking techniques. The work in [62], improved and accelerated the primitive sequential SAT attack [60] via implementing several tweaks and dynamic optimization techniques in the attack procedure. However, it still only works on static locking techniques. ScanSAT [61] is another sequential SAT attack, which is able to break dynamicity in DOS architecture. In scanSAT, by relying on the fact that the LFSR structure of DOS architecture (and its polynomial) are known to the adversary, it would be able to find the seed as well as update frequency parameter, which is the only secret in DOS architecture. So, it could derive all the keys that are dynamically generated on the chip. Another state-of-the-art sequential

SAT attack is DynUnlock [63], which has a similar approach to ScanSAT, and shows how it is possible to break the dynamicity and find the seed of the PRNG in EFF-Dyn [52].

Furthermore, locking or blocking the scan chain architecture will impose some critical issues for the test/debug of the design-under-test. For example, disabling the shift operation in scan blocking forces the tester to rely on PI/PO for the functional test, which significantly increases the test flow complexity. Also, the structure of some scan chain locking/blocking techniques forces the designer to engage a trusted party (the tester should have the correct key) for the functional test; however, it is a hard assumption to maintain in many practical cases. In some other cases, due to the structure of the secure scan chain, a system reset is required for each test pattern to initiate the state of the circuit, which significantly increases the test time.

2.4 Previous Work on LUT/Routing Locking

Unlike almost all previously discussed countermeasures that use XOR-based logic locking, MUX-based (could be used for routing locking) and LUT-based locking has been previously visited by few researchers[2]. Engaging reconfigurable logic like look-up-tables was first evaluated in [24], where LUTs as the reconfigurable barriers are used for locking purposes. In [24], some placement strategies have been introduced for LUT insertion. For instance, controllability and observability are two metrics used for part of their selection heuristic, expressed as don't-care conditions. Another parameter in their strategies is the gate-level distance with PI/PO. Although power, performance and area (PPA) overhead is thoroughly evaluated in [24], the claim on the security of these schemes is made solely base on inability to readout the content of LUTs after reverse engineering, and the proposed placement strategies are not resilient against the SAT attacks.

The work in [64] proposed another LUT-based locking technique with three different LUT placement algorithms, which relies on spin-transfer-torque (STT) magnetic technology

[2]The only group that exploits non-XOR (MUX-based) is cyclic-based, where key-programmable MUXes are added to generate feedbacks/cycles.

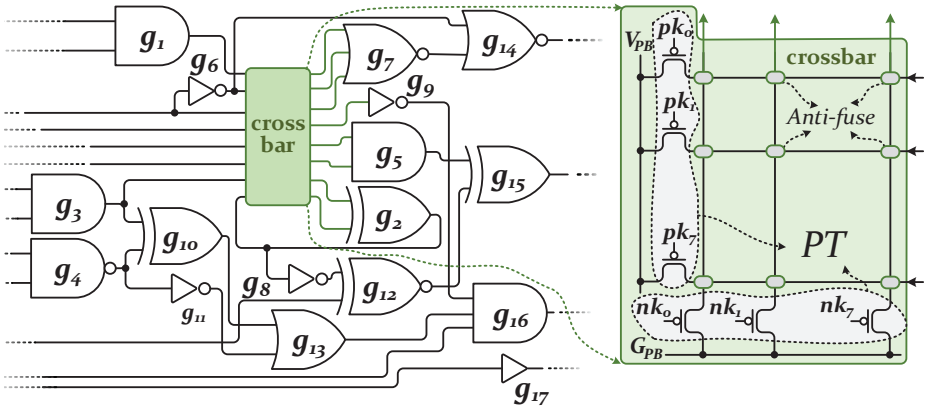


Figure 2.11: Circuit Locked by Cross-Lock [5] with an 8×8 Crossbar Network.

for implementing LUTs. Three different placement algorithms are: (1) dependent selection in which there is the dependency between reconfigurable units (reachable), (2) Independent selection in which the gates are randomly selected such that they may or may not connected (directly or indirectly) through any design path, and (3) parametric-aware selection in which the placement aims to minimize the impact and possibly avoid violating timing requirement (inserting in timing paths with the highest positive slack). This work further focuses on the PPA impact of their strategies and illustrates that utilizing STT-based LUTs could reduce the PPA impact. However, the proposed solution does not consider its resiliency against the SAT attack.

Routing-based locking was first introduced and investigated in Cross-Lock [5]. In Cross-Lock [5], key-based switching network are built using programmable-vias (PVIA) constructed using one-time-programmable (OTP) elements made of anti-fuse [21]. PIVAs are used to implement $n \times m$ crossbars. The main approach for PVIA programming is connecting two complementary (NMOS and PMOS) programming transistors (PTs) to the two ends that connect the device terminals to programming supplies [21]. Routing locking in Cross-lock has been performed by inserting PVIA-based $n \times m$ crossbars. Fig. 2.11 shows a small circuit locked by Cross-Lock with $n = m = 8$.

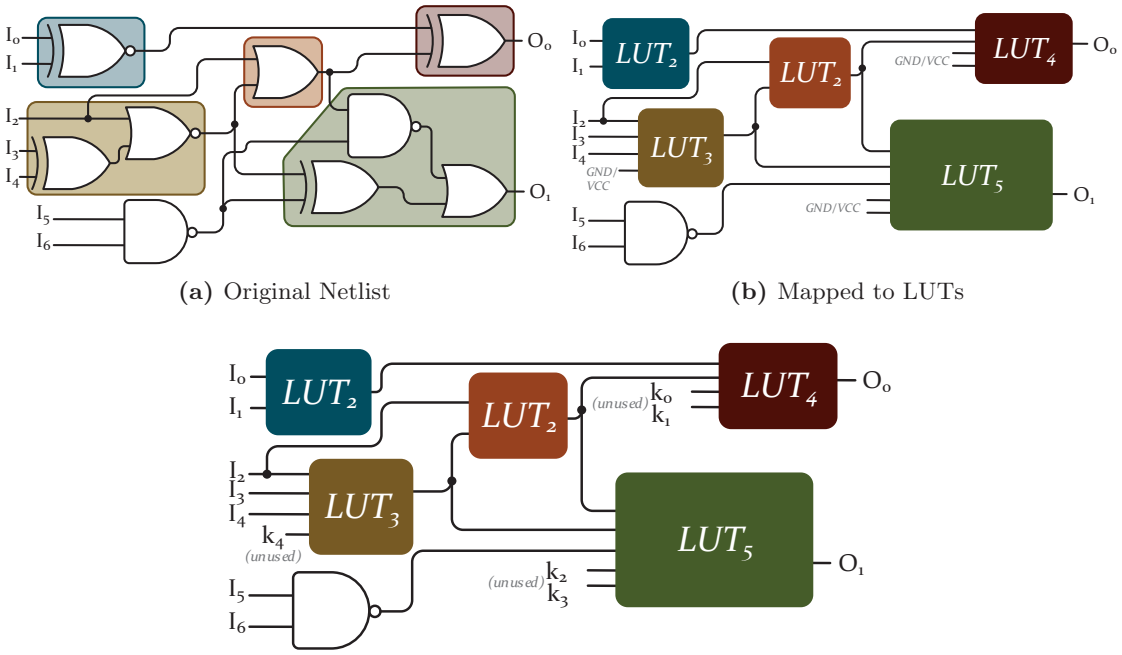
Chapter 3: LUT-Lock: SAT-resilient LUT-based Locking

Compared to XOR-based and MUX-based logic locking, the usage of Look-up-tables (LUTs) as a means of locking could have a wider range of technologies as the application. For instance, since the main building blocks of FPGAs are built using LUTs, LUT-based logic locking could be used more efficiently in FPGAs. FPGAs are inherently more secure for their post-silicon reconfigurability. However, the FPGA hardware security relies on the protected and non-intruded mapping of the intended bitstream into FPGAs. In certain cases, it is difficult to protect the bitstream both during the initial configuration in untrusted third-party systems as well as during remote and in-field reconfiguration [65]. A successful attack may result in an unauthorized transfer of a bitstream to a third-party, reverse-engineering, injection of a hardware Trojan, and cloning or theft of embedded IPs [65, 66].

Although high-end FPGAs are typically equipped with bitstream encryption, there are many cases where encryption alone is not enough [66]: (1) Not all FPGA families are equipped with implementations of cryptographic algorithms [65], especially for small and low-energy FPGAs. (2) When the power and delay overhead of the bitstream encryption process is not tolerable, a developer may choose not to use encryption. (3) Many FPGA-based products, to support new services or to enhance the existing ones, require frequent updates which are mostly accomplished remotely. Despite the first time safely programming, for an in-field update or a remote upgrade, the encrypted bitstream, and the keys are vulnerable to leakage [65]. (4) After deployment, FPGAs are susceptible to physical attacks. The long-term in-field usage makes it possible for an attacker to extract the encryption keys via various side-channel attack mechanisms [67]. So, it is essential to implement additional security measures to prevent the usability of a leaked bitstream. Such threats validate the need for implementing logic locking as an extra security measure to protect the bitstream.

3.1 LUT-based Locking in FPGA

In FPGA solutions, the hardware resources are fixed and are designed independently of a given netlist. Hence by nature, state-of-the-art FPGAs provide a large pool of resources to apply 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 [65] depicts the utilization of Altera Cyclone V after mapping a diverse set of benchmarks of various scales and complexity to this FPGA and reported that FPGA utilization is typically low. This phenomenon was coined as FPGA-Dark-Silicon [65]. These unmapped and unutilized LUTs are freely available and could be used for logic locking purposes. Hence, LUT-based locking in FPGAs could be considered as utilizing unused LUTs or using larger than needed LUTs, where the connectivity and impact of additional logic are controlled using keys. An example of the process of using LUTs in FPGA for logic locking is illustrated in Fig. 3.1, where some of the 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 new inputs (keys) for locking purposes. It also could be taken from output of an internally implemented (non)linear feedback shift register (LFSR) or a physical unclonable function (PUF) [68]. Also, by changing the ordering of inputs based on the key values (generated by PUF), the locked circuit possibilities will increase considerably. For instance, assuming that a PUF is used, each FPGA has unique PUF responses. By knowing the PUF responses 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. These missing key values serve as the locking key in LUT-based locking. Also note that the bitstream, in this case, is unique for each FPGA, as each FPGA has unique PUF responses. In this case, even if the bitstream is leaked, the PUF responses remain unknown, making the problem similar to ASIC flow, where after reverse-engineering the locked netlist is available, but the keys are unknown.



(a) Original Netlist

(b) Mapped to LUTs

(c) Locking Unutilized LUTs

Figure 3.1: LUT-based Logic Locking using Unutilized LUTs.

3.2 LUT-based Locking in ASIC

In ASICs, utilizing LUTs for logic locking can lead to 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 be replaced with basic gates in a netlist. Also, 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 [64, 69, 70] 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 logic locking in ASICs is straightforward: selected cells are removed and replaced by LUTs. The functionality of the cell remains hidden from the manufacturer. LUTs are then programmed after fabrication

in a trusted testing facility.

3.3 Different Placement Strategies used in LUT-Lock

Since none of the previous LUT-based locking techniques evaluated the security of LUT-based locked circuits against the existing threats, our proposed LUT-Lock algorithm combines several key features, each introduces a specific strategy for the placement of LUTs in the circuit, which helps to enhance its ability to resist against the 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 chapter, we illustrate how by adding each key feature, the resiliency of the locked netlist against the SAT attack increases, proving that the resiliency gained from adding these features are orthogonal to one another.

3.3.1 FIC: Focusing on the Fan-In Cone of Primary Outputs

Since the SAT attack works on a structure-like binary decision tree, deepening the search tree for the SAT solver will increase the complexity significantly. Having a deeper tree dependent on the key values makes the SAT problem much harder to be solved. Hence, moving towards dependent LUT insertion would be more promising. Hence, mapping the LUTs such that they place in the same fan-in-cone (FIC) would increase the strength of the locking against the SAT attacks. To achieve this, we limit the LUT insertion to the fan-in cone of the smallest possible set of primary outputs (best case being single PO), and we refer to this algorithm as FIC. It should be noted that FIC-based LUT replacement still corrupts other outputs, as the intersection of fan-in cones of different outputs is not empty. Hence, we expect to face high (enough) output corruption even while we choose a limited number of FICs of POs. Also, to have more control on the number of POs affected by LUTs, in LUT-Lock, we start with replacing the closest cells to the selected output first and then we move towards PIs.

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

(1) Total Positive Slack (TPS) of all timing paths leading to that primary output(s) should be large (timing paths with highest positive slack). 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 the 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. These two conditions mean that we require wider FICs than deeper ones. For large circuits, we define two coefficients (α and β) for prioritizing these two conditions to generate a cumulative weight which helps to select the best candidate output(s). 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.

Fig. 3.2 demonstrates different features of LUT-Lock we introduce in this chapter, and Fig. 3.2(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 (deeper paths). So, in this simple example, we choose g_8 as the PO, and its FIC would be the candidates for locking. When using breadth-first-search (BFS) for gate selection (moving from POs towards PIs), 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.

3.3.2 HSC: Focusing on Higher Skew Gates in FIC

Our investigation on the hardness of many tested LUT placement strategies revealed that the cells with lower controllability are better candidates for LUT-based locking. The controllability could be defined as the effort/hardness metric for controlling the logic value of a wire in a circuit. Hence, it is evident that the controllability of the wires located near the POs would be much harder than other wires located near the PIs. Since we move from POs towards PIs in LUT-lock, we already choose gates with higher controllability. However, for sibling gates, located at the same depth, we need to calculate controllability to prioritize

them for the placement strategy. Hence, we could define the controllability using the probability as the skew probability (SPS) $|P_r(0) - P_r(1)|$, in which $P_r(1)$ and $P_r(0)$ denoted as the probability of being 1 or 0 at the output of the gate, respectively. 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 (feature) of our LUT-lock algorithm is to enhance the FIC to perform the gate selection based on its measure of the gate's output (higher) SPS. 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 locking, its fan-in gates will be added to the list of gates that could be visited in the next search for gate replacement, and the gates with the highest SPS will be selected among all gates in the list. HSC replacement flow is illustrated in Fig. 3.2(b). In the first invocation of HSC, the fan-in-cone of gate G_8, for satisfying the FIC requirements, is selected and is locked. 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. 3.2(b), G_4 with SPS of 0.5 is selected over G_5 and G_2 with SPS of 0.25 and *zero*, respectively. For the 3rd gate selection, HSC appends the fan-in gates of G_4 (Here are primary inputs and will be ignored!) as candidate gates for the replacement along with G_2 and G_5. Hence, among these 2 gates, G_5 is selected for having the higher SPS.

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

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 *almost zero*. Separating the fan-in cones of different outputs could be achieved by replicating the common gate, however, this will result in a large area overhead. To have more control over the corruption of the POs without exploding the area, we introduce another sub-algorithm (feature), in which we give more priority to candidate gated with lower fan-outs. We refer to this gate selection strategy as MFO-HSC.

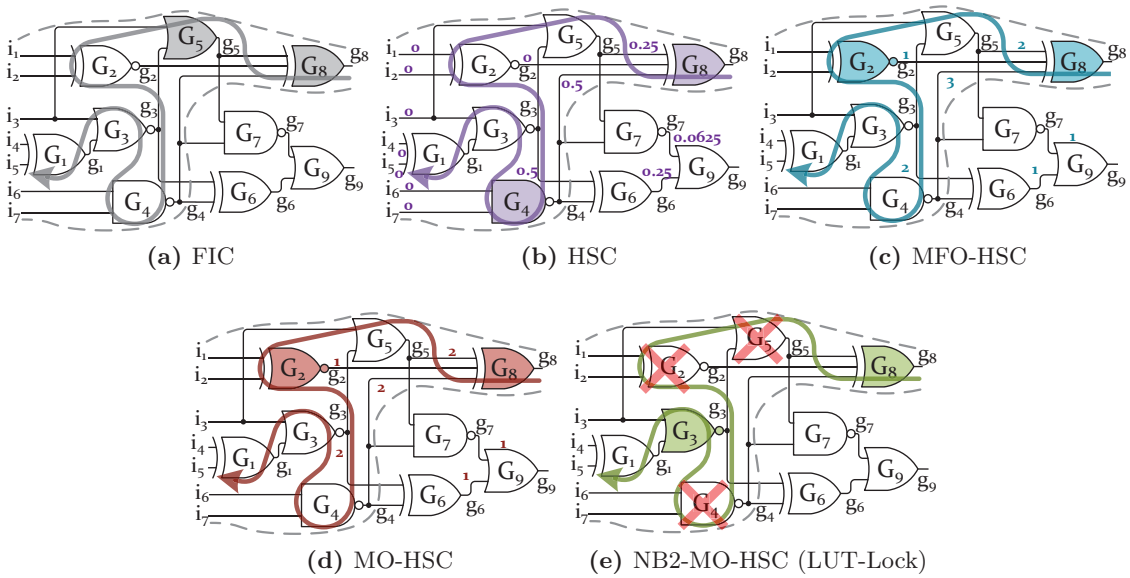


Figure 3.2: Different Placement Strategies in LUT-Lock.

In the MFO-HSC algorithm, a BFS search is first deployed (FIC), visiting all candidate gates at the current cone, 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 (the previous feature). When a gate is locked, its fan-in gates are added to the list of candidate gates that will be considered in the next gate selection. Fig. 3.2(c) depicts how the MFO-HSC works; Similar to FIC, the fan-in cone of g_8 is selected for locking and G_8 is locked. 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 next candidate gate is only G_5.

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

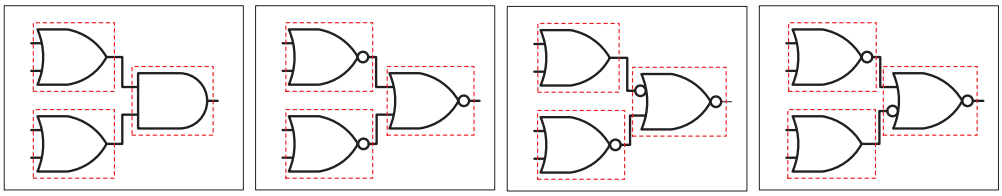
Based on our observation in MFO-HSC, some gates have more than one fan-out, but they only affect one output. For instance, as it can be seen in Fig. 3.2(c), the fan-out of g_4 is 2. However, it affects only g_9. This observation led us to introduce a similar but more efficient

sub-algorithm (feature) to choose gates with an impact on minimum outputs, 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 (direct or indirect) to each candidate gate. MO-HSC requires additional parsing and processing, however, it further provides better control on outputs corruption. Similar to MFO-HSC, the tie between two candidate gates (for affecting an equal number of outputs) is broken using the SPS of respective gates. Each time a gate is selected for its locking, the fan-in of the gate is added to the list of candidate gates to be considered for the next gate selection. Similar to the FIC algorithm, each gate replacement candidate should pass the timing check, otherwise ignored. An example of the MO-HSC has been illustrated in Fig. 3.2(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 the smaller number of outputs.

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

The back-to-back locking of the 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 shown in Fig. 3.3, the back-to-back locking of the function $(A \vee B) \wedge (C \vee D)$, using 2-input LUTs (replacing all three gates with 2-input LUTs), could have 4 different combinations of programmable logic based on De Morgans's Laws with the same function. So, instead of having *one* correct key, we will have *four* different keys all provide the correct functionality. When LUT placement focus on FIC, placement of each LUT in the fan-in of the logic cone with a direct connection with other LUTs, creates another set of possibilities leading to the 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, locking more gates may even reduce the locking strength. This is illustrated in Fig. 3.4, where the execution time of the SAT solver and some generated keys per each inserted LUT for the benchmark C5315 of ISCAS-85 is illustrated. The LUTs are placed back-to-back, hence,



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

Figure 3.3: De-Morgan's law: Four Different Conversion with the Same Function.

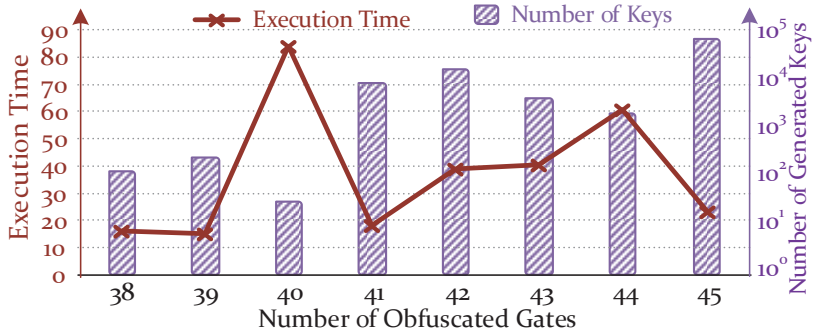


Figure 3.4: Number of Valid Keys in back-to-back LUT placement (c5315).

insertion of each LUT increases the number of keys. Fig. 3.4 focuses on the insertion of the 38th to the 45th LUT. As shown, the insertion of 41st and 42nd LUT produces a large number of new (correct) keys (around 10^4). Hence, the SAT solver execution time doesn't increase. On the other hand, the replacement of gate 40 produces far fewer new keys (in the 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 the exploitation of De Morgan's gate conversion laws. So, we introduce another sub-algorithm (feature), NB2-MO-HSC, which implements this restriction by avoiding back-to-back locking, 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 the 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. 3.2(e), the application of NB2-MO-HSC results in the selection of G_8 and G_3 for the LUT insertion.

3.4 LUT-Lock Flow: Implementing NB2-MO-HSC

LUT-Lock relies on all previously mentioned strategies combined in NB2-MO-HSC. In summary, it is the combination of (1) no back-to-back, (2) minimum impact on POs (minimum fan-out), (3) on gates with higher SPS, and (4) located in selected FICs. The Algorithm 2 captures the detailed implementation of the proposed LUT-Lock locking 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 behavior and also more possible candidates during each iteration, we embed MO-HSC in the LUT-Lock algorithm.

3.5 The Efficacy of LUT-Lock against the SAT Attack

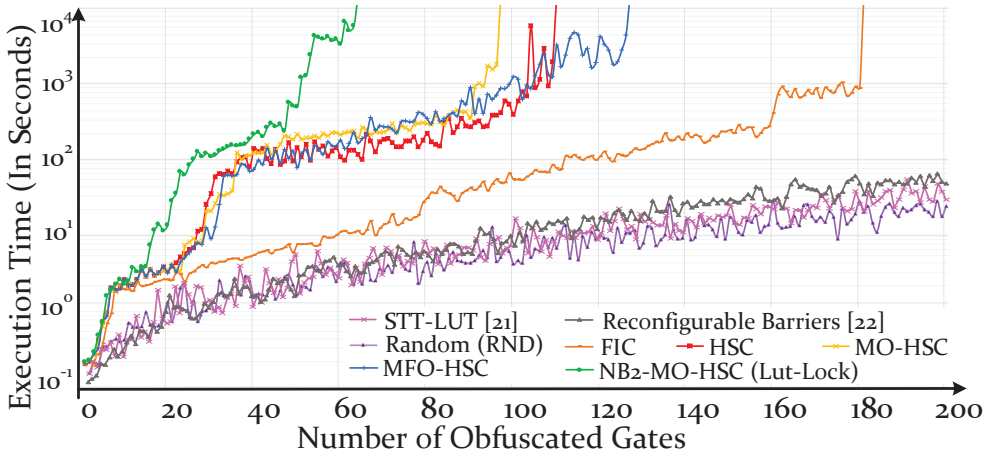
To evaluate the efficacy (Robustness) of the LUT-Lock algorithm, we used a set of desktops equipped with Intel Core-i5 processor and 8GB of RAM. For a fair comparison, and to reduce the impact of the operating system background processes, we dedicated one machine to each SAT solver at a time, and installed 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 LUT-Lock. We employed the Lingling-based SAT attack described and developed by [1]. We measured the SAT solver execution time by increasing the number of locked gates (the number of LUTs) from 1 to 200. A run-time limit of 1.1×10^4 seconds was set for the SAT solver.

To show the effectiveness of each key feature of the proposed algorithm, we compared

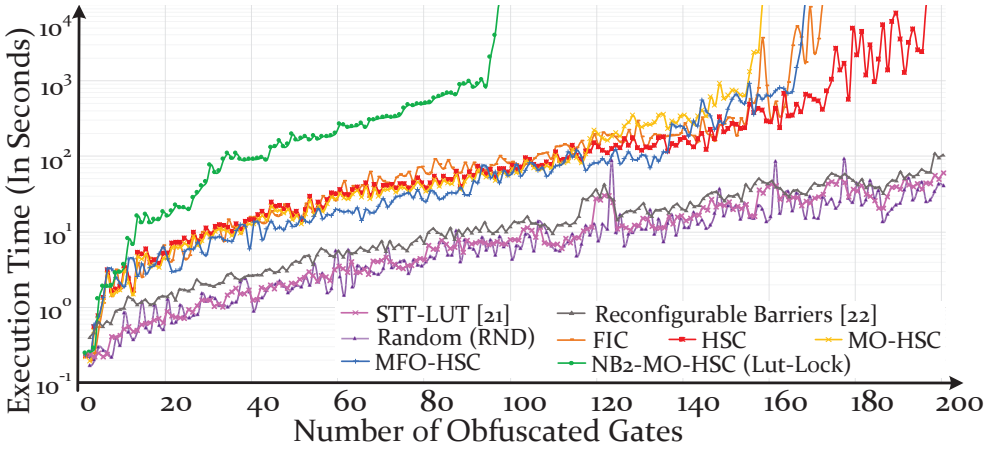
Algorithm 2 LUT-Lock: Implementing NB2-MO-HSC

```
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_locked < 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;
```

the execution time of the SAT attack on circuits which are locked based on different features (sub-algorithms). We also compare the effectiveness of the proposed LUT-Lock with the previous work, STT-LUT [64] and reconfigurable barriers [24]. As illustrated in Fig. 3.5 the SAT attack execution time increases as the replacement algorithm evolves from Random replacement to FIC, then to HSC, then to MFO-HSC, then to MO-HSC, and then to MB2-MO-HSC, illustrating the orthogonal improvement of added features in providing resiliency against the SAT attack. The LUT-Lock algorithm, which implements the NB2-MO-HSC replacement policy, combines all key features and provides a close to an exponential increase in the execution time of the SAT attack with respect to the number of locked gates.



(a) on ISCAS-85 c5315



(b) on ISCAS-85 c7552

Figure 3.5: Execution time of the SAT Attack on LUT-Lock vs. Previous Work.

Fig. 3.5 shows that although the execution time of the SAT attack increases steadily, it faces small variation. The variation in the execution time is the result of the 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 3.3.5), compared to the rate of growth in the number of possible keys. A poor selection of candidates for locking results in faster growth in the number of valid keys, reducing the overall effectiveness of locked netlist against the SAT attack. As illustrated, the LUT-Lock (NB2-MO-HSC) has the least variation, as it eliminates the explosion of the set of valid keys by preventing back-to-back gate replacement.

Table 3.1: Exponential Regression of the SAT Attack Exection Time on LUT-Lock.

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

Table 3.2: The SAT Attack Execution Time on LUT-Locked ISCAS-85 Benchmarks.

Circuit	1%		2%		3%		5%		10%	
	RND	LUT-Lock	RND	LUT-Lock	RND	LUT-Lock	RND	LUT-Lock	RND	LUT-Lock
c2670	0.18	0.88	0.5	1.39	0.93	1.92	2.41	24.6	3.48	Timeout
c3540	0.6	1.24	1.07	6.12	2.25	988.2	2.66	Timeout	5.29	Timeout
c5315	0.5	9.05	1.21	115.01	1.66	941.02	3.93	Timeout	12.04	Timeout
c7552	0.79	28.432	2.61	182.9	3.71	492.04	11.1	Timeout	264.9	Timeout

As shown in Fig. 3.5, the SAT resiliency of the prior work is very close to that of random (RND) replacement, showing slow growth in the SAT attack execution time with respect to the number of LUTs. However, in LUT-Lock, it shows a much faster exponential increase in difficulty (the SAT execution time). As illustrated, in both c5315 and c7552 benchmarks, with only 20 LUTs, the LUT-Lock locked netlist is as resilient as the netlist locked by [64] and [24] when they use $10X$ (200 gates) LUTs.

Table 3.1 captures the fitted function (exponential regression) of execution time for different sub-algorithms and LUT-Lock, where x denote the number of locked gates. As illustrated in this Table, the LUT-Lock (NB2-MO-HSC) poses an exceptionally more challenging SAT problem compare to other locking scheme. Table 3.2 compare the execution time of SAT attack, across selected number of ISCAS-85 benchmarks, once locked 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.

3.6 From Theory to Reality: LUT-lock Overhead

When LUT-based locking is in place, like LUT-lock, as we demonstrated, the design would be partially mapped to LUTs. For instance, if a 2-input AND gate have to be locked

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. From the SAT attack perspective, on the other hand, to model the LUT-based locking, each LUT is substituted with a (2+)-level MUX. Unlike the preliminary LUT-based locking techniques that are vulnerable to the SAT attack, LUT-lock can provide robustness against this powerful attack. However, it would have more chances to being resilient when we increase the number of LUTs inserted/replaced into the circuit. So, the impact of LUT-lock on the overhead is required to be considered meticulously.

When a LUT with size u is used to be replaced with a gate, each u-input LUT can provide all 2^{2^u} possible functions, which increases the key length along with the search space to find the correct key configuration for LUT. However, increasing the size of the LUTs imposes large area and performance overheads, even while it is built using STT. To evaluate the efficiency (robustness vs overhead) of the LUT-based locking, we investigate some key factors that play important role in both robustness and overhead. These factors are (1) LUT size, (2) number of LUTs inserted for locking, and (3) replacement strategy.

Fig. 3.6 illustrates the overall impact of these three key factors on the SAT attack execution time. As shown, even for random (RND) strategy, for LUT size larger than 8, locking only ∼1% gates of each circuit is sufficient to provide the SAT resiliency. Further, it is evident that *NB2-MO-HSC* remarkably increases the SAT execution time, which shows its effectiveness on the SAT resiliency. However, increasing the size of LUTs (on the X-axis) significantly increases the hardness of locking regardless of the replacement strategy and the number of LUTs locked.

3.6.1 LUT Size vs. Number of LUTs

One of the straightforward approaches for LUT-based locking is to either increase the number or the size of LUTs to enhance the security against the SAT attack. For example, instead of using a LUTu for the u-input gate, a LUTu+ (i.e., LUT$u+1$, LUT$n+2$,...) could be used. When LUTs are replaced by MUXs for SAT modeling leads to a $\log_2(u)$-level MUX-based structure. Thus, by increasing the size of the LUTs, the SAT attack replaces them

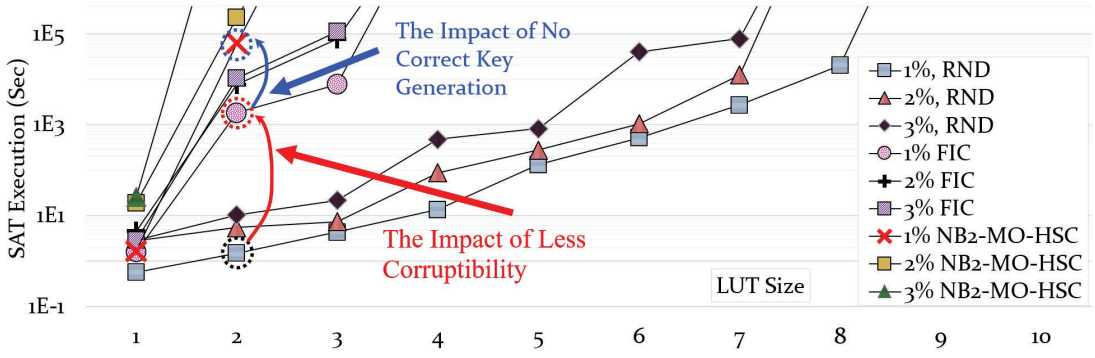


Figure 3.6: The SAT Attack Execution Time for Different Values of the Key Factors.

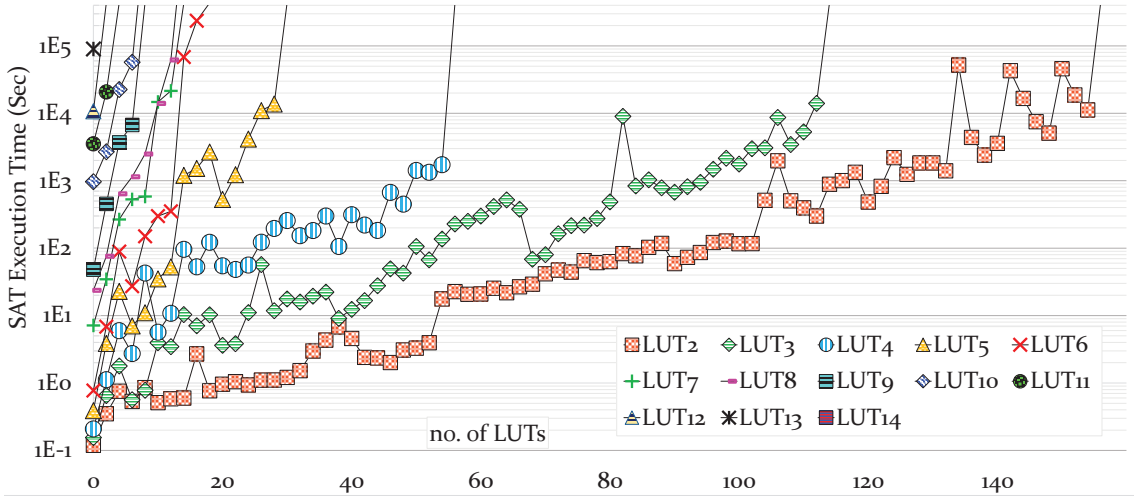


Figure 3.7: The SAT Attack Execution Time: LUT Size vs. Number of LUTs.

with more deeper MUX trees, and consequently, the attack time gets exponentially longer to exploit the value of keys for LUTs, making large LUT resilient against the SAT attack.

Fig. 3.7 demonstrates the SAT attack execution time with more details on ISCAS-85 c7552 for different sizes of LUTs, and different numbers of LUTs using the replacement strategy NB2-MO-HSC. This experiment shows that using larger sizes of LUT provides higher SAT resiliency than locking a higher number of gates with smaller sizes of LUTs. For instance, only replacing a single gate with a LUT_{13} is sufficient to make the design perfectly resilient against the SAT attack.

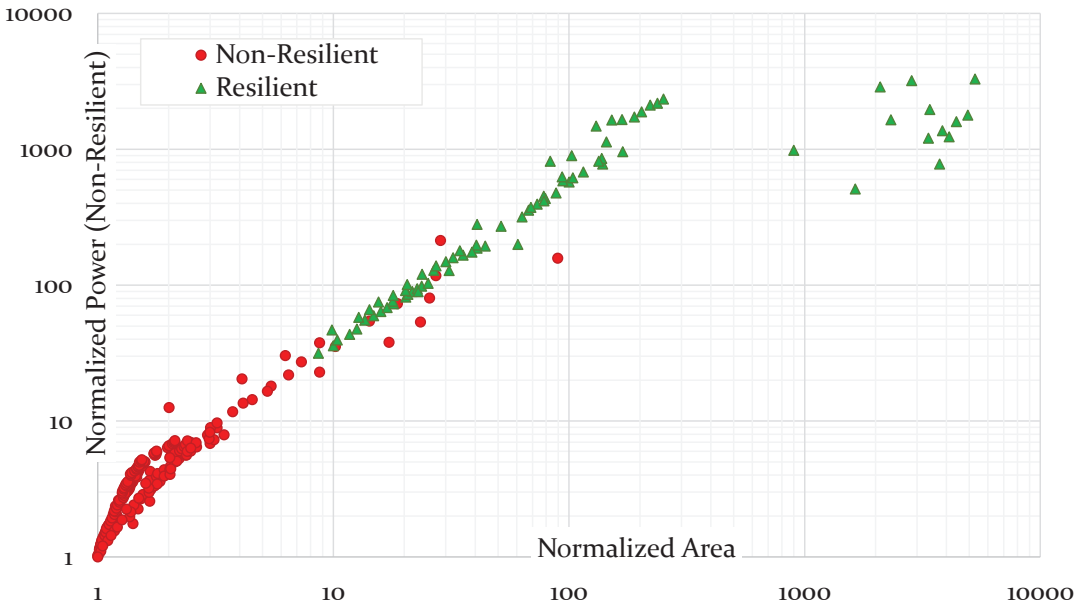


Figure 3.8: Normalized Area/Power Overhead of LUT-based locking.

3.7 More Investigation is Required on LUT-based Locking

Although Fig. 3.7 shows that LUT scale-up (enlarging the LUTs) is the most straightforward approach for LUT-based locking for yielding high resiliency against the SAT attack, Fig. 3.8 shows that precipitous increase in the number and the size of LUT can render inefficient solutions. Every possible combination from Fig. 3.7 is synthesized using Synopsys Design Compiler to obtain area and power overhead which are demonstrated in Fig. 3.8. As shown, making the circuits resilient against the SAT attack incurs at least 10x area and power overhead, making LUT-based locking an idealistic solution at the moment for hardware security.

Therefore, to make LUT-based locking a realistic and resilient solution with reasonable overhead, different directions need to be investigated. One possible direction is to design and build a customized LUT structure with more focus on the security of the LUT at lower overhead. Exploiting the combination of MUX-based and LUT-based locking as a full configurable block also could be evaluated as a possible direction. As we demonstrated, the strategy of LUT replacement plays a crucial role to provide resiliency at lower overhead.

Hence, more investigation on LUT placement strategies is inevitable in this topic. Also, all previous LUT-based locking techniques use LUT-per-gate replacement which incurs significant overhead, and it could be replaced with LUT-per-cone replacement. Undoubtedly, these factors could be the main contributors to the future of LUT-based locking techniques, which will show us the feasibility/idealism of this breed of locking countermeasures.

Chapter 4: Full-Lock: Moving towards Routing-based Locking

The inefficiency of LUT-based locking, particularly in terms of overhead, motivates us to open a new direction by investigating other non-XOR-based logic locking techniques, i.e. MUX-based logic locking. In MUX-based logic locking as discussed in Section 4.4, key-based MUXes are added, and the selector of the MUXes are controlled with the key. So, in MUX-based locking, the routing/wiring of the circuit would be locked. In this chapter, we introduce and evaluate a new MUX-based (routing-based) locking technique, which could open a new direction for building SAT-hard solutions. Unlike point function techniques which exponentially increase the number of iteration required for the SAT attack, the main aim of the proposed routing-based locking is to significantly increase the run-time of *each iteration* of the SAT solver. The only existing solution that somewhat could be enumerated as a member of this category is Cross-lock [5]. In Cross-lock, a one-time programmable interconnect mesh is used to lock the routing of a netlist, and the resulting locked netlist substantially increases the runtime of each iteration of the SAT attack. However, we will illustrate that locking solution in [5], 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 chapter, it is possible to construct locked circuits that translate into far harder SAT circuits than Cross-lock.

In this chapter, we explore the characteristics and principles of designing this new category of SAT-hard logic locking 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 direction of logic locking techniques, we introduce *Full-Lock*. 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 the 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.

4.1 A New Perspective of SAT Hardness

As previously mentioned in 2.3.2, in each of the SAT attack iteration, the SAT solver would be invoked to find a *discriminating input patterns* (DIP), which results in ruling out a set of incorrect key value(s). Hence, many SAT-resilient locking schemes tried to weaken the pruning power of one DIP, making sure each DIP can only rule out a very limited number of (the best case is one) incorrect key. This forces the number of needed iterations to be exponentially increased with respect to the number of keys. Consequently, it exponentially increases 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 the SAT solvers comes from their conflict-driven clause learning (CDCL) ability. In each iteration of the SAT attack, a new SAT problem will be defined, in which there exist many clauses each contains a set of literals. The goal of the SAT solver is to find a satisfying assignment 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

to a safe point before the conflict.

4.1.1 Recursive DPLL in the SAT Solver

Davis-Putnam-Logemann-Loveland (*DPLL*) algorithm (or one of its derivatives), which is used to perform CDCL, is illustrated in Algorithm 3. Each SAT iteration invokes the recursive DPLL function. Besides, DPLL is a recursive function that 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 the SAT solver (a SAT attack iteration).

Algorithm 3 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)
```

Based on the SAT attack algorithm discussed in Section 2.3.2, the runtime of the SAT attack could be obtain from:

$$T_{Attack} = \sum_{i=1}^{N} T(i) = \sum_{i=1}^{N} (t + T_{DPLL}(\Phi)) \tag{4.1}$$

We call locked circuit SAT resilient if the SAT attack faces a very large runtime on it. As we discussed previosly in Section 2.3.4, the most intuitive way as the first solution is

to weakening the DIP and increasing the number of iteration (N) to a very large number [25–27]. In spite of very shallow DPLL recursive tree, for having a very large N, these solution exhibit resistance against SAT attack. However, this type of logic locking solutions, as suggested previously is prone to SPS [25], Approximate-based [38, 39], bypass [36], and possibly removal attack [35].

Based on the discussion on DPLL, an alternative solution is smaller N but larger recursive trees. Hence, as illustrated in equation 4.2, the attack time could also increase beyond acceptable if the number of recursive calls (M) grows to a very large number.

$$T_{Attack} = \sum_{i=1}^{N}(t + T_{DPLL}(\Phi)) \simeq \sum_{i=1}^{N}\sum_{j=1}^{M}(T_{DPLL}^{Avg}) \simeq MN \times T_{DPLL}^{Avg} \tag{4.2}$$

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

Motivated from this discussion, in this chapter we present 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 MUX-based locking instance(s) in the circuit.

4.2 Full-Lock: SAT-hard Routing Locking

Many SAT-hard problems (instances) are introduced annually in SAT competitions. 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, features and tricks used in these SAT-hard problems could be used in designing SAT-hard circuit (SATC) problems.

In [6], the SAT hardness of formulas produced using a 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 Algorithm 3 increases the chances of one satisfying assignment to be found early in the search using *unit propagation* or *purification*. Note that these two steps are used to simplify the size of the 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 [71]. Fig. 4.1 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 clause 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 to solve a 3-SAT instance which is constructed with 100 variables and 450 clauses. From this discussion, an locked 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).

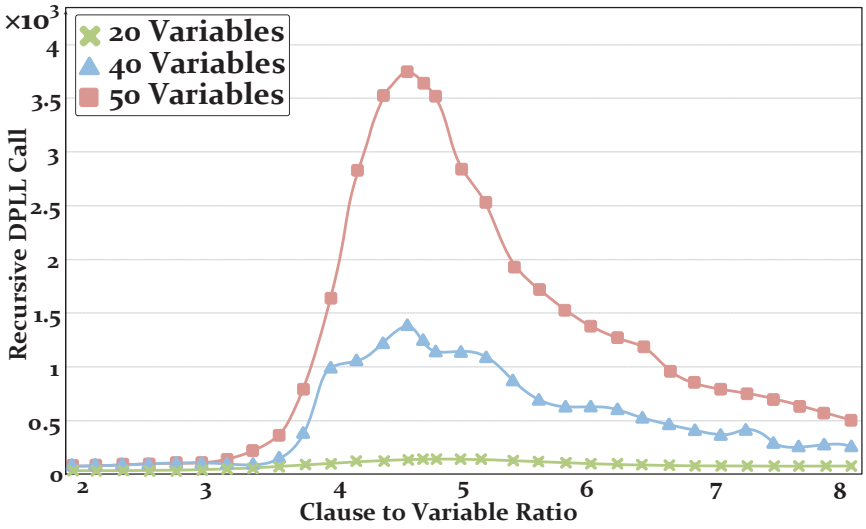


Figure 4.1: Recursive DPLL Call for Different Clause to Variable Ratio [6].

4.2.1 Logarithmic Networks for SAT-Hardness

Since the SAT solvers receive the inputs (SAT problems) in CNF, the Boolean representation of the circuit must be translated into CNF before the attack. This translation could be done using Tseytin transformation [73]. Table 4.1 lists the Tseytin transformation [73] 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 clause to variable ratio is 1 and 4/3 in MUX and XOR/XNOR respectively. Despite 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, a MUX can make the recursive DPLL tree one level deeper, (2) unit propagation and purification steps in the DPLL algorithm provide more simplified and smaller formula using enhanced Gaussian elimination while the contribution of XOR/XNOR gates is much higher [75]. 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.

Table 4.1: Tseytin Transformation of Basic Logic Gates.

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

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 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 variables 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 the switching network is cascaded while pushing up the clause to variable ratio to the desired range. This is consistent with the finding in the [76], in which investigating Boolean formulations of global detailed interconnect constraints, authors concluded that the CNF of symmetric switching networks is a hard problem for the SAT solvers. Besides, using N-by-M switch-boxes, with back-to-back interconnection, illustrated in Fig. 4.2 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 [7].

In Full-Lock, we achieve this by constructing a logarithmic-based key-programmable routing block (KeyRB) for locking the wires. For this purpose, we create small and lightweight switch-boxes (SwB) that are implemented easily using only MUXes/inverters. 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

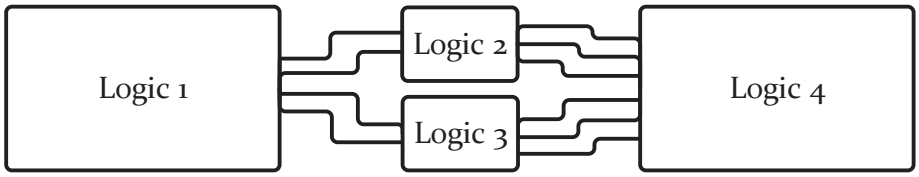


Figure 4.2: N-by-M switch-boxes for Building Hard Satisfiable Instances [7].

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 existing switching networks, self-routing logarithmic networks, log_2N networks, provide 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 *banyan*, *baseline*, *shuffle*, etc. Fig. 4.3 demonstrates a simple implementation of a 8×8 keyRB using the blocking *shuffle* network [77]. This keyRB is constructed using small SwBs, where each SwB is built using MUXes/inverters. In each SwB, the outputs can be an arbitrary permutation of the inputs. Also, as shown, we add key-configurable inverters for each wire, allowing the outputs of each keyRB to be shuffled and negated based on the key value. The keyRB has N inputs, and due to its structure, N is a power of 2. The numbers of SwBs in a keyRB depend on the number of inputs as well as the model of log_2N networks. In all aforementioned blocking keyRB, the number of SwBs is the same, i.e. $N/2 * logN$, and the only difference between them is the topology of SwBs interconnections.

4.2.2 Moving towards non-Blocking Logarithmic Networks

The previously discussed self-routing logarithmic networks are blocking networks as they cannot propagate all permutations of their inputs to the outputs. In Section 4.4, 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.

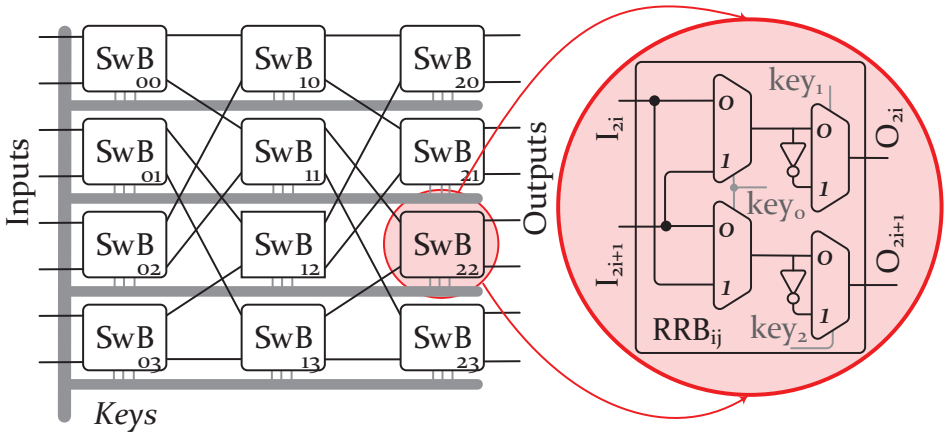


Figure 4.3: Shuffle-based Blocking Logarithmic-based *keyRB* with $N = 8$.

According to [78], 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 makes the network strictly non-blocking, results in constructing a much larger network than a blocking keyRB. 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 keyRB 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 [78] for constructing the logarithmic-based key-programmable routing block (keyRB). 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. 4.4, demonstrates an example of such an almost non-blocking keyRB with $N = 8$. As it can be seen, the topology of SwBs interconnections is different with *shuffle*-based, shown in Fig. 4.3. This topology is a *banyan*-based interconnection that matches with our proposed $LOG_{N,log_2(N)-2,1}$.

Since almost non-blocking keyRB has only $log_2(N) - 2$ extra stages, its area/power

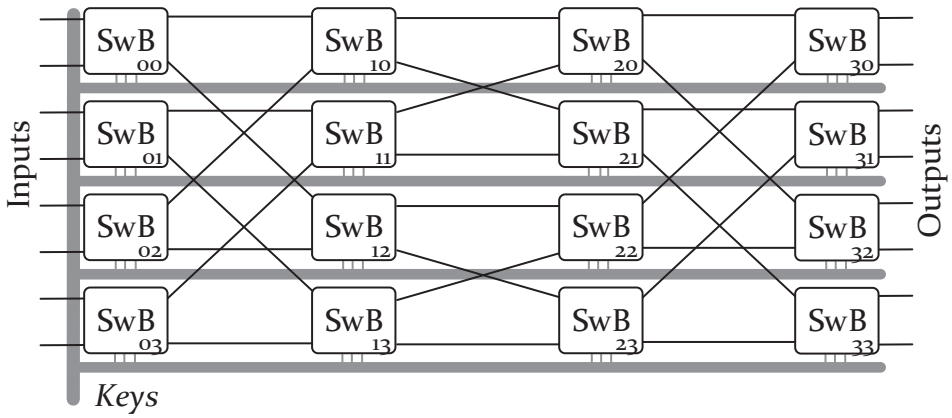


Figure 4.4: Almost Non-Blocking Logarithmic-based KeyRB with size 8 ($LOG_{8,1,1}$).

overhead is roughly 2x compared to a blocking keyRB with the same N. However, this almost non-blocking keyRB is far more resistant against SAT attack compared to a blocking network. For example, an $N = 64$ input non-blocking keyRB 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.

4.2.3 Strongly Twisted KeyRB into LUT/Logic

The keyRB provides an interconnect locking scheme that can generate a SAT-hard instance which significantly increases the execution time for each SAT iteration. However, to enhance this strength, and especially to be resilient against other types of attacks, such as removal attacks, we try to twist keyRB 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 keyRB. These inverters allow us to combine the keyRB with the logic of the gates leading to its inputs. So, both logic and interconnect locking are embedded into the keyRB. For instance, let us suppose that one of the inputs of keyRB is derived using an *OR* gate. Using these inverters, we can change it to *NOR*, and configure the keyRB to generate its negate

on its corresponding output. These key-configurable inverters within keyRB allow us to change the logic of the gates leading it. So, even removing keyRB and finding the correct permutation provided by keyRB will not generate the correct functionality. Also, 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 keyRB will not change.

Furthermore, we replace the gates preceding the keyRB with small spin-transfer-torque-(STT)-based LUTs with the same input. Combining keyRB with LUTs provides 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 u inputs, it adds up to u layer to the recursive DPLL tree. Moreover, since LUTs are directly connected to the output of keyRB, these extra u layers will be added to the large recursive DPLL tree of keyRB. Hence, by massively increasing the size of a recursive DPLL tree of keyRB 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 FPGAs, however, they provide significantly higher speed running at GHz frequency, near-zero leakage power, high thermal stability, and highly integrative with CMOS [79]. Since, each gate, located at the output of keyRB, 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. Also, the power and area overhead is considerably low in these LUTs with size less than 5. As shown in Fig. 4.5, LUTs with sizes 2, 3, 4, and 5, have negligible overhead compared to CMOS-based basic gates. Besides, the size of all gates leading the keyRB can be decreased to be 2. For instance, an *AND3* gate can be changed to two *AND2* while the output 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 are required.

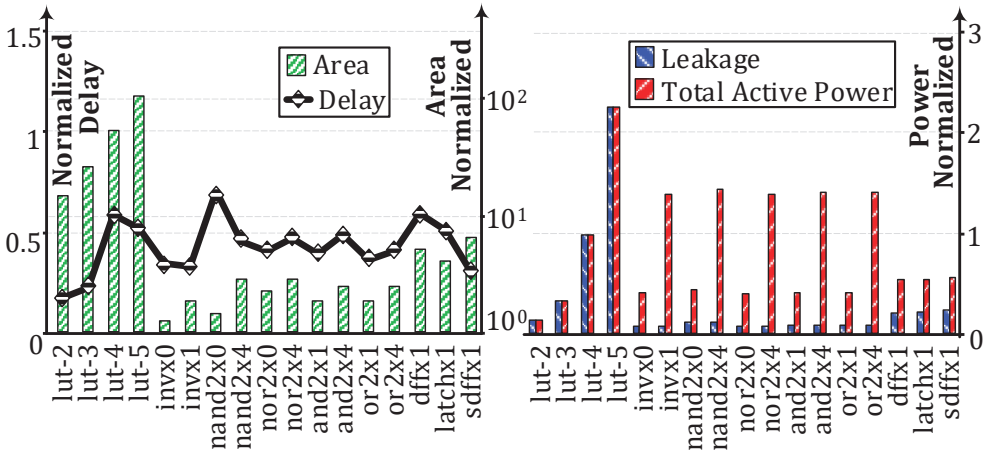


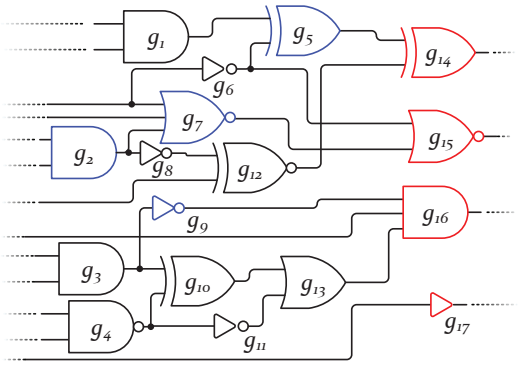
Figure 4.5: Power, Delay, and Area of STT-LUT and Standard Cells in 28nm CMOS.

4.3 Inserting SAT-hard *PLR*s into Design

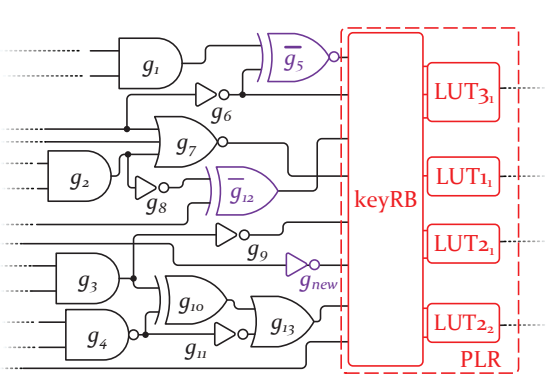
Using these PLRs provides a big advantage compared to other locking schemes. Since inserting a PLR in a circuit provides a SAT-hard instance in the circuit, it is not required to employ a specific insertion to enhance the strength of PLRs. However, due to the topological structure of circuits, it may be beneficial to have an insertion policy. But, we demonstrate that even using random insertion/replacement strategy for these PLRs creates an extremely large recursive DPLL tree that makes the circuit resilient against SAT.

Additionally, in comparison with Cross-lock [5] that is a layout-based interconnect locking scheme, Full-Lock has no restriction on the 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 keyRB into the logic of the gates leading and preceding the selected wires, even removing the keyRB using a removal attack does not generate correct functionality. Hence, there is no limitation for wire selection in Full-Lock.

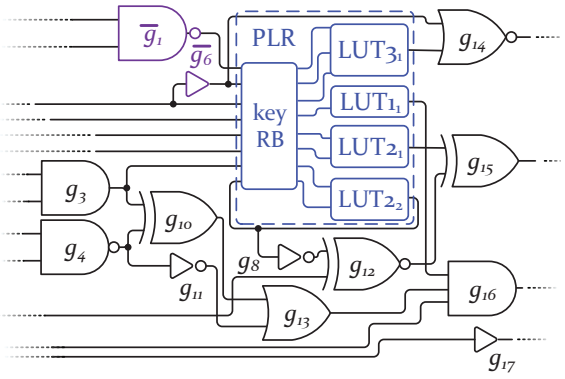
Fig. 4.6 demonstrates two simple examples that how Full-Lock inserts PLRs in the circuit. As shown in Fig. 4.6(a) and (b), the selected gates are highlighted in red, i.e. g_{14},



(a) Gate-level of Original Circuit



(b) PLR with *Acyclic* Structure



(c) PLR with *Cyclic* Structure

Figure 4.6: An Example of Routing-based Locking using Full-Lock Example.

g_{15}, g_{16}, and g_{17}. Since these gates have no impact on each other, replacing them with PLR, including keyRB and LUTs, does not generate any cycle in the design. However, Fig. 4.6(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 keyRB is changed (negated), all highlighted in purple, i.e. $\overline{g_5}$, $\overline{g_{12}}$, g_{new} in Fig. 4.6(b), and $\overline{g_1}$, $\overline{g_6}$ in Fig. 4.6(c), which shows that how twisting leading gates into keyRB is working. For instance, g_5 in Fig. 4.6(a), an *XOR*, has been replaced with $\overline{g_5}$ in Fig. 4.6(b), an *XNOR*. In this case, keyRB will recover the functionality of this gate using key-configurable inverters that are embedded into keyRB.

4.4 Robustness/Overhead Evaluation of Full-Lock

To show the efficiency of Full-lock, it is evaluated using different SAT-based attacks, including SAT for acyclic [1, 3], cycSAT for cyclic [41], and AppSAT for approximate-based [38, 39], all implemented in C++, and were run on a Dell PowerEdge R620 equipped with Intel Xeon E5-2670 2.50GHz and 64GB of RAM. Assuming that the circuits are cycle-free before locking, since routing-based locking techniques might add cycles into the circuit (combinational cycles), a cyclic-based SAT attack is required (cycSAT I in this case) to be applied on Full-Lock.

4.4.1 Blocking *vs.* almost non-Blocking KeyRB

As was mentioned previously, Since not all but almost all permutations can be generated using non-blocking keyRB, $LOG_{N,log_2(N)-2,1}$, 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 keyRB and an almost non-blocking with different sizes using SAT. As it can be seen in Table 4.2, increasing the keyRB 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 keyRB with a size larger than $N = 64$, however, for blocking keyRB, it is easily broken for all sizes less than $N = 512$.

Since keyRBs 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 keyRB 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 keyRBs.

Table 7.6 demonstrates power/area/delay of blocking and almost non-blocking keyRBs for different sizes using Synopsys generic 32nm educational libraries. As it can be seen, the

Table 4.2: SAT Execution Time on *shuffle*-based Blocking KeyRBs.

KeyRB Size (N)	4	8	16	32	64	128	256	**512**
Shuffle-based Blocking KeyRB								
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 KeyRB								
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 4.3: Overhead/SAT-Resiliency of Blocking vs. almost non-Blocking KeyRBs.

keyRB	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**	✓

incurred overhead by the smallest almost non-blocking keyRB, which is resilient against SAT ($N = 64$), is approximately one-third of the smallest SAT-resilient blocking keyRB ($N = 512$) in terms of power consumption. Additionally, the overhead imposed by keyRB is significantly low compared to area and power of even small-scale benchmark circuits.

4.4.2 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.

Table 4.4: Execution Time of the SAT Attack on Full-Lock.

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*

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 4.4 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 keyRBs 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 4.5 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 keyRB and

Table 4.5: PLRs Size in SAT-resilient Full-Lock compared to Cross-Lock.

Circuit	# Gates	# I/Os	Full-Lock	Cross-Lock [5]
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

another PLR with a 8×8 keyRB 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 iteration, we calculate the average clauses to variables ratio using MiniSAT for different logic locking schemes during the attack. As it can be seen in Fig. 4.7, 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 locking, 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.

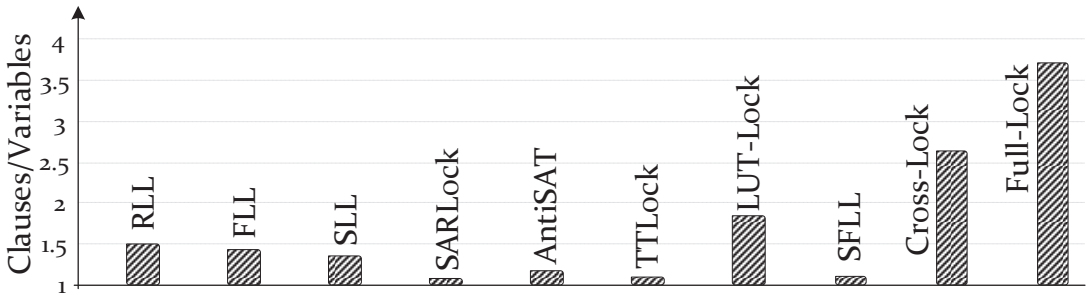


Figure 4.7: Average C2V Ratio for Different Logic Locking Schemes.

Security Against Removal Attack

As was mentioned previously, Cross-lock [5] 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 crossbar. However, since the logic of the gates leading each keyRB can be negated, even having the possibility of removing keyRB, 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

keyRB 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 attack of keyRB. 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 5: CP&SAT: A New Attack on Routing-based Locking

As we discussed in Section 4.1, Full-Lock as a symmetric routing-based locking technique shows how the direction of adding difficulties to the SAT attack could be changed (becoming deeper) via deepening the DPLL tree. This deepening could be achieved if the logic locking technique forces a relationship between the number of clauses and the number of variables to maximize the penalty associated with incorrect variable assignment symmetrically across the search tree. As we discussed in Section 4.1, with having a deeper DPLL tree, the runtime formula of the SAT attack could be re-written as follows:

$$T_{Attack} = \sum_{i=1}^{N}(t_i + T_{DPLL}(\Phi_i)) \simeq \sum_{i=1}^{N}\sum_{j=1}^{M}(T_{DPLL}^{Avg}) \tag{5.1}$$

In Chapter 4, we demonstrated that the main aim of this technique as a symmetric routing locking technique is to extremely increase the number (M) and the computational complexity (T_{DPLL}^{Avg}) of recursive calls in DPLL algorithm, which occurs when the DPLL tree is extremely deep/large enough.

5.1 Canonical Prune-and-SAT Attack

As of today, there is still no successful attack on routing-based locking techniques. Each iteration of SAT solving on routing-based locked circuits faces an ultra-deep and complex DPLL tree. Hence, the SAT attack cannot even find a satisfying assignment(s) to complete the attack process. However, in this Chapter, we propose *canonical prune-and-SAT* (*CP&SAT*) attack on the routing-based locking techniques. In the *CP&SAT*, we first model

the key-programmable routing blocks (keyRB(s)) as numerical bound problems, and then a bounded variable addition (BVA) algorithm has been engaged as a pre-processing step to reduce the size and complexity of numerical bound problems. By using the BVA algorithm, the CNF corresponded to each keyRB will be reduced dramatically in terms of the number of clauses. Then, the re-encoded CNF will be solved using the traditional SAT attack.

5.2 Threat Model in *CP&SAT* Attack

The *CP&SAT* attack will be performed based on the conventional threat model for logic locking [1, 2, 15], where:

1. The adversary has access to the successfully reverse-engineered yet locked netlist. Hence, (s)he has all the necessary information about the netlist, such as the locking technique, the key gates, the key inputs, etc. Specifically in routing locking, the location of the keyRBs could be determined by the adversary.

2. The adversary has access to an activated/unlocked chip, in which the correct key is embedded into a secure tpNVM.

3. With having scan chain access on the activated/unlocked chip, the adversary can apply the SAT attack on each combinational part of the circuit, independently.

5.3 Attack Flow

The proposed *CP&SAT* attack is composed of three main steps: (1) modeling the keyRB(s) to be presented as a numerical bound problem, where for each output of keyRB, a sub-CNF will be extracted from the CNF of the whole circuit; (2) re-encoding the sub-CNF corresponded to each keyRB output using bounded variable addition (BVA) algorithm; (3) merging the updated (reduced) sub-CNFs into the CNF of the whole circuit, running the traditional SAT attack, and match the key for the correct routing.

5.3.1 Modeling keyRB as a Numerical Bound Problem

Extensive analysis on the application of Boolean satisfiability in detailed routing constraints [7, 76, 80–82] shows that the SAT solvers can consider simultaneously the routability constraints for all nets, leading to potentially faster convergence to a solution. However, this only happens when an appropriate *encoding approach* has been chosen to represent routing constraints as a SAT problem before solving. Many studies have investigated and compared different encoding approaches [76,81]. Using the key observations provided in these studies, in the first step of the proposed *CP&SAT* attack, we encode the sub-CNF related to each keyRB output using *one-layer linear encoding*. To describe the logic-equivalent model, for each output of a $n \times n$ keyRB, the one-layer linear encoding replaces the original sub-CNF with a CNF describing a one-layer $n - to - 1$ multiplexer (MUX) controlled by the one-hot key. More formally, for a $n \times n$ keyRB, the sub-CNF of each keyRB output, which is encoded using one-layer linear encoding, will be as follows:

$$\bigwedge_{\substack{M \subseteq 1,\ldots,n, \\ |M|=1}} \Big(\bigvee_{i \in M} x_i k_i \Big) \tag{5.2}$$

In which x_i denotes the wire that is connected to the i^{th} input of the keyRB, and k_i denotes the one-hot key that connects the i^{th} input of the keyRB to the corresponded keyRB output when it is 1, and M is the search space for each keyRB output. The Eq. 5.2 is the most special case of encoding of numerical bounds [83]. The numerical bound problems could be denoted as $\leq p(x_1, x_2, ..., x_n)$, meaning that among n variables p variables are allowed to be assigned true. The most special case of numerical bounds is when $p = 1$, called *at-most-1 constraint*, that is applied whenever a finite domain is encoded, and the Eq. 5.2 is one form of at-most-1 constraint encoding. According to this encoding definition, in the first step of the proposed *CP&SAT* attack, we first extract the sub-CNF related to each output of keyRB(s). Then, we use *one-layer linear encoding* for the extracted sub-CNF to be encoded as a numerical bound problem. Then, in the second step as described in the

next section, we use the BVA to re-encode and reduce the size of each sub-CNF for each output of the keyRB.

5.3.2 SAT Reduction using Bounded Variable Addition

As an integral part of SAT solving, *resolution* and *variable elimination* (VE) are two rules that would be applied on CNF before running the SAT solver to reduce the size of variables/literals [84–86]. The VE, as a proof procedure for CNF formulas, faces an exponential space complexity. Hence, to make it practical for usage, the VE must be bounded [85, 86]. In bounded VE (BVE) a variable x could be eliminated only if $|S| \leq |S_x \cup S_{\bar{x}}|$, in which $S_x(S_{\bar{x}})$ denotes a set containing clauses all contain $x(\bar{x})$, S is obtained from Eq. 5.3, and $|S| \leq |S_x \cup S_{\bar{x}}|$ means that the resulting CNF $\big(((F \setminus (S_x \cup S_{\bar{x}})) \cup S\big)$[1] will contain no more than the original CNF (F) clauses.

$$S = S_x \otimes S_{\bar{x}} = \{C_1 \otimes C_2 | C_1 \in S_x, C_2 \in S_{\bar{x}}, C_1 \otimes C_2 \neq Tautology\} \tag{5.3}$$

In *CP&SAT attack*, we engage the complementary version of BVE, called *bounded variable addition* (BVA) [83], in which either a new variable will be added to the CNF or a variable will be substituted. Similar to BVE, the same *bounding* concept must be used in BVA to decrease the size of the CNF [83]. As the simplest example of the BVA, by adding a new variable x to the following formula F with 6 clauses, the re-encoded formula F' would have one clause less.

$$F = (a \vee c) \wedge (a \vee d) \wedge (a \vee e) \wedge (b \vee c) \wedge (b \vee d) \wedge (b \vee e) \tag{5.4}$$

$$F' = (a \vee x) \wedge (b \vee x) \wedge (c \vee \bar{x}) \wedge (d \vee \bar{x}) \wedge (e \vee \bar{x}) \tag{5.5}$$

In the BVA, the number of possibilities to add or substitute a variable is extremely large.

[1] $((F \setminus (S_x \cup S_{\bar{x}})) \cup S$ means that, in CNF F, both sets of clauses, S_x and $S_{\bar{x}}$, that contain x and $\bar{x}$, respectively, must be replaced with clauses of set S that are built using Eq. 5.3.

Hence, to make it practical for any CNF (F), the BVA algorithm must be constructed based on two steps:

1. *Replaceable Matching*: Creating a pair of sets consisting of a set of literals (SET_L) and a set of clauses (SET_C) such that for all $\{l, c\} \in \{SET_L, SET_C\}$, the clauses $(c \setminus \{SET_L\}) \cup \{l\}$ are either in CNF (F) or tautological.

2. *matching-to-clauses*: Using a method that creates the sets $S_x = \{(l \vee x) \mid l \in SET_L\}$ and $S_{\bar{x}} = \{(c \setminus SET_L) \cup \{\bar{x}\} \mid c \in SET_C\}$, and removes all clauses $(c \setminus \{SET_L) \cup \{l\}$, and replaces them with $S_x \cup S_{\bar{x}}$.

By applying these two steps on F (Eq. 5.4), $SET_L = \{a, b\}$ and $SET_C = \{(a \vee c), (a \vee d), (a \vee e)\}$, F' could be generated using *matching-to-clauses* with one clause less as shown in Eq. 5.5. Now, by using this 2-step BVA algorithm, any CNF formula could be reduced provably in size (the number) of clauses while the reduced CNF is also provably equivalent with the original CNF.

Theorem 1. *For two sets as* replaceable matching $\{SET_L, SET_C\}$ *as for the CNF formula F, F' as the reduced of F could be constructed by adding a Boolean variable such that (1) F' is logically equivalent to F and (2) F' contains $|F'|+|SET_C|+|SET_L|-|SET_C|\times|SET_L|$ clauses if none of the resolvents is a tautology.*

Proof. For two sets as replaceable matching $\{SET_L, SET_C\}$, we can construct F' as follows: All clauses of $(c \setminus \{SET_L) \cup \{l\}$ must be removed from F and must be replaced with $S_x \cup S_{\bar{x}}$ that are obtained using the matching-to-clauses construction method. The number of removed clauses is $|SET_L| \times |SET_C|$, while the number of added clauses is $|SET_L| + |SET_C|$ proving (2). Since the BVA is the complement of BVE, by applying BVE on x in F', it re-produces (reverse) F, and BVE preserves logical equivalence proving (1). ■

One of the best fitting applications of the BVA algorithm is re-encoding cardinality constraints [7, 80, 83, 87], where it is necessary to encode numerical bounds ($\leq k(x_1, x_2, ..., x_p)$).

As discussed previously, the one-layer linear encoding formulates each keyRB output as the most special case of cardinality constraints ($k = 1$), called *at-most-1 constraint*. Compared to naive encoding for *at-most-1 constraint* in which the number of clauses is $n.(n + 1)/2$, by using BVA algorithm, the number of clauses would be reduced to $\sim 3n$ [83].

It is worth mentioning that numerous studies are explaining how cardinality constraints (numerical bounds) could be encoded efficiently [88–92]. Also, a few SAT solvers handle cardinality constraints by itself, such as Sat4J [93] or clasp [94]; however, since these solvers do not extract cardinality constraints from the formula, compared to the direct re-encoding using BVA, their efficiency is extremely low. Furthermore, the strongest SAT solvers tend to not support native cardinality constraints, such as MiniSAT [95] that was supporting native cardinality constraints up to version 1.12. In *CP&SAT attack*, we employ the simpleBVA proposed in [83] as a pre-processing step before running the SAT attack and after one-layer linear encoding. The simpleBVA will be used for each sub-CNF, corresponded to each keyRB output, and encoded using one-layer linear encoding, separately.

5.3.3 SAT Execution and Key Matching

After the reduction using the BVA algorithm, we update the CNF of the whole circuit using the reduced sub-CNFs corresponded to keyRB(s) outputs. Now, it is time to run the traditional SAT attack on the updated CNF. Since each keyRB might add cycles into the design, as mentioned in Section 5.2, we need to use a cyclic-based SAT attack. Assuming that the circuits are acyclic, the CycSAT-I will be used.

As was mentioned previously, in one-layer linear encoding, the actual keys of each keyRB will be replaced with a set of one-hot key controlling the MUXes (one-layer encoding). Hence, after breaking the updated CNF, the SAT attack will recover the values of the one-hot keys. These one-hot keys determine the correct wiring/interconnection for the MUXes. So, a matching step is required, in which we need to calculate the actual key for each keyRB that establishes the same (correct) wiring/interconnection built by one-hot key in MUXes.

Table 5.1: The Effectiveness of the BVA Pre-Processing Step on Routing Blocks.

Instance	Original			BVA Pre-processed		
	#Variables	#Clauses	Solve	#Variables	#Clauses	pre+Solve
keyRB-4	271	418	**0.02**	428	202	**0.01+0.22**
keyRB-8	875	1606	**0.45**	1278	718	**0.01+0.36**
keyRB-12	1544	3084	**2.48**	2188	1288	**0.01+0.54**
keyRB-16	2419	4750	**5.42**	3982	2184	**0.01+0.82**
keyRB-24	3372	7502	**54.82**	4618	3452	**0.02+1.64**
keyRB-32	6178	12510	**194.8**	8892	7258	**0.02+2.22**
keyRB-48	9891	18614	**Timeout**	12672	9918	**0.04+3.92**
keyRB-64	15043	31182	**Timeout**	23818	14772	**0.04+12.22**

Timeout $= 10^5$ *Seconds* $\approx$ *1 day*

5.4 *CP&SAT* Attack Effectiveness on Routing Locking

To evaluate our proposed *CP&SAT* attack, we engage well-known ISCAS-89 and ITC-99 combinational circuits locked using Full-Lock[2][3]. We sweep the size of keyRBs to show the efficiency of the BVA algorithm on routing-based locking. All the experiments are implemented using Python/C++ and have been carried out on many Dell PowerEdge R620 equipped with Intel Xeon E5-2670 2.50GHz and 64GB of RAM.

5.4.1 The Efficiency of the BVA

To show how the BVA algorithm efficiently reduces the CNF size of the routing-based locked circuits, Table 5.1 shows the rate of reduction that is more than a factor of two. The BVA adds/substitute the variables to decrease the number of clauses. As shown, the number of variables increases (by up to 2x); however, the number of clauses that play an important role in determining the complexity of the CNF is decreased by more than 2x. Hence, the BVA-based pre-processed CNFs are far easier for the SAT solver to be solved.

Furthermore, as can be seen in Table 5.1, increasing the size of the keyRBs does not increase the SAT runtime after BVA pre-processing exponentially (most likely quadratically).

[2]Since Cross-Lock is a weaker version of Full-Lock (It has no configurable inverters), we only report the attack results on circuits locked by Full-Lock.

[3]Since all ISCAS-89 and ITC-99 are sequential, we apply all techniques on combinational parts of these circuits, and we assume that all FFs are accesible to be read/written for both SAT and *CP&SAT* attack.

Table 5.2: SAT Attack vs. *CP&SAT* Attack on Full-Lock.

Circuit	#Gates	#I/O	Traditional SAT Attack (CycSAT-I)			proposed *canonical prune&SAT* (with inverters)			proposed *canonical prune&SAT* (detached inverters)		
			keyRB-16 (16×16)			keyRB-16			keyRB-16		
			2	3	4	2	3	4	2	3	4
b15	∼8.5K	485/519	1507.5	TO	TO	486.4	2581.5	TO	44.8	75.9	319.8
b14	∼9.5K	277/299	788.3	TO	TO	329.7	1688.8	TO	34.8	88.9	416.6
s35932	∼16K	1763/2048	856.6	TO	TO	643.8	5238.1	TO	76.4	147.9	407.4
s38417	∼18K	1464/1731	1187.4	TO	TO	482.5	2037.9	TO	58.2	100.7	366.3
b20	∼19.5K	522/512	1096.8	TO	TO	537.8	3507.9	TO	70.8	129.4	411.2
b21	∼20K	522/512	1832.4	13283	TO	984.8	8207.3	TO	134.4	207.8	550.1
b17	∼30K	1452/1512	508.2	8401.7	TO	306.8	6095.4	TO	81.7	137.4	463.8
b22	∼30K	767/757	924.6	6491.5	TO	508.2	5538.4	TO	68.5	99.1	390.5
b18	∼110K	3357/3343	1283.7	9208.1	TO	581.9	6327.8	TO	91.6	162.7	472.2

Timeout (TO) $= 10^5$ *Seconds* $\approx$ *1 day*

Since we cannot infinitely increase the size of the keyRB (due to overhead and limitation of candidate selection), we report the results on keyRB with size up to 64×64.

5.4.2 *CP&SAT* Attack on Full-Lock

To show the success of our proposed *CP&SAT* attack on routing-based locking, in Table 5.2, we illustrate the runtime of the SAT (CycSAT-I) and the *CP&SAT* attack on circuits locked by Full-Lock. As shown, in all cases, after inserting four keyRB-16 (16×16), the traditional SAT attack fails to break the locked circuits. However, when we apply the *CP&SAT* attack, all circuits locked by four keyRB-16 are broken in less than 10 minutes. When we assume that the configurable inverters of SwBs in Full-Lock are in place, the BVA algorithm within the *CP&SAT* attack does not provide a significant advantage. As shown in Table 5.2, we observed that this new attack also fails to break circuits locked with four keyRB-16 while inverters are intact. However, as described previously, we detach the inverters in Full-Lock by fixating the key values of all layers of inverters (disable the inversion) except the last layer. When the inverters are detached, the BVA could efficiently reduce the size of the locked circuit allowing us to break the locking within few minutes.

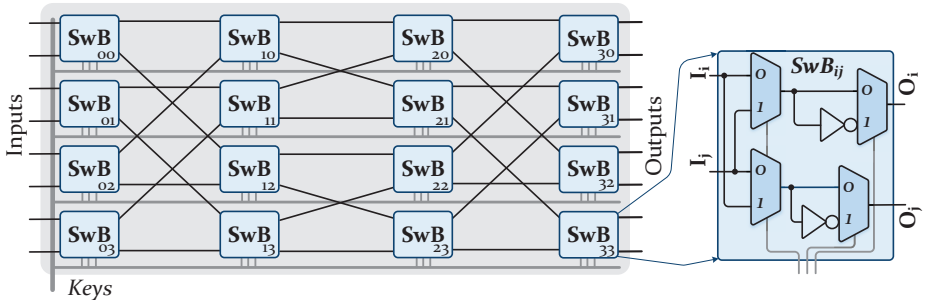
Chapter 6: Interlock: Moving towards Intercorrelated Logic and Routing Locking

In the previous Chapter, we described how prior routing-based locking techniques could be broken using the proposed *CP&SAT*. It calls into a question that *"How routing locking could be still used while it is not vulnerable to the BVA algorithm?"*. In this Chapter, we answer this question by proposing a countermeasure that improves the resiliency of this category of locking techniques against *CP&SAT* attack.

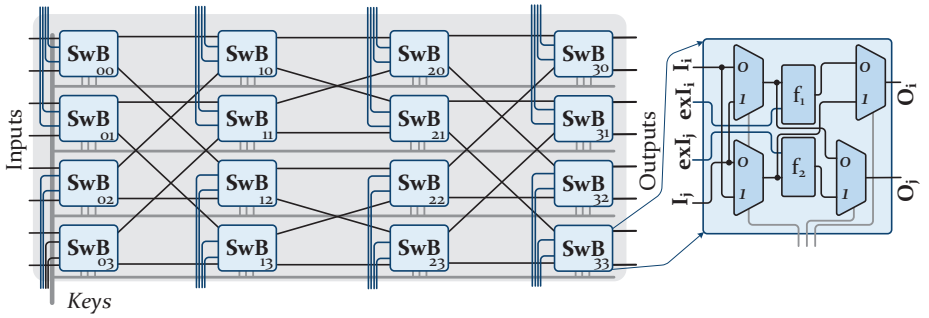
6.1 Truly-Twisted Logic & Routing Locking

To still get the benefit of routing locking, and to combat against the efficiency of BVA-based re-encoding on routing-based locking, we truly twist the keyRB with logic gates, meaning that a part of the actual logic gates will be also embedded into the keyRB.

In the *CP&SAT* attack, we explained how a keyRB could be modeled as multiple numerical bound problems before the BVA re-encoding. So, the idea is that when the routing-based locked circuit could not be translated (converted) to a numerical bound problem, the BVA is no longer applicable to it. For this purpose, inspired by the logarithmic (permutation) networks proposed in Full-Lock, we employ the same architecture for keyRBs in InterLock; however, for each layer of that hierarchy, we will add a Boolean function (logic gate). Fig. 6.1 shows our new keyRB architecture that must be used for routing-based locking. Compared to Full-Lock, for each switch-box (SwB), the configurable inverters are removed, and f_1 and f_2 are added that could be any of 2-input basic logic gates, i.e. *NAND, NOR, XNOR, AND, OR, XOR*. Also, for each SwB, we add extra inputs (exI) as one of the inputs of 2-input logic gates. For each SwB with 4 inputs (I_i, I_j, exI_i, exI_j), output O_i



(a) key-programmable Routing Block (KeyRB) in Full-Lock



(b) key-programmable Routing Block (KeyRB) in InterLock

Figure 6.1: Full-Lock vs. InterLock.

could be $\{I_i,\ I_j,\ f_1(I_i, exI_i)$, and $f_1(I_j, exI_i)\}$, and output O_j could be $\{I_i,\ I_j,\ f_2(I_i, exI_j)$, and $f_2(I_j, exI_j)\}$.

6.1.1 Different Possibilities for f_1 and f_2

In this section, we aim to explain (1) *"Why the usage of f_1 and f_2 gates in the SwBs improves the resiliency of Interlock (compared to full-lock in which only routing and inversion is implemented)?"*, and (2) *"how the selection of the logic for f_1 and f_2 affects its resiliency"*. To answer these two questions, we investigate five different scenarios: f_1s and f_2s could be (1) still inverters with no extra input (similar to Full-Lock), (2) all *NAND* (*AND*), (3) all *NOR* (*OR*), (4) all *XNOR* (*XOR*), and (5) selected randomly (any arbitrary 2-input gate).

Table 6.1 shows the runtime of the SAT attack (CycSAT-I) on an locked ISCAS-85 c7552 circuit while only one keyRB is embedded into the design based on these five scenarios. The first and the most promising observation is that, compared to Full-Lock (when the logic

Table 6.1: The SAT Attack Runtime on KeyRBs with Different Logic.

$f_{1,2}$ / Size	Full-Lock *NOT*	InterLock all *NAND*	InterLock all *NOR*	InterLock all *XNOR*	InterLock Random
keyRB-4	0.02	0.192	0.136	0.718	0.232
keyRB-8	0.437	3.083	5.905	2062	19.79
keyRB-16	5.413	522.1	558.2	Timeout	62332
keyRB-32	195.1	Timeout	Timeout	Timeout	Timeout
keyRB-64	Timeout	Timeout	Timeout	Timeout	Timeout

Timeout $= 10^5$ *Seconds* $\approx$ *1 day*

layer is still inverter), for the same-size keyRB, the InterLock (all scenarios) builds a much harder SAT problem. As shown, in Full-Lock, the smallest single keyRB that breaks the SAT attack is a 64×64 keyRB (keyRB-64). However, in InterLock, it is even smaller (keyRB-16 or keyRB-32). Furthermore, we observe that, while all f_1s and f_2s are *XNOR* (or *XOR*), keyRB-16 is enough; however, for other gate types (*NAND, NOR, AND, OR*), the smallest resilient is keyRB-32.

6.1.2 Embedding Actual Timing Paths into KeyRBs

This is a key observation that when all f_1s and f_2s are *XNOR* (*XOR*), the SAT resiliency of the keyRB is extremely higher. But, as shown in Fig. 6.1, similar to Full-lock that engaged inverters to handle the toggling of some gates preceding the keyRB, in InterLock, these extra gates must become a part of the actual logic gates to avoid far exceeding the overhead. However, it is less likely to find a set of paths that only consist of *XNOR* (*XOR*). Hence, if we select the gate of each SwB based on an actual gate in a selected timing path, all f_1s and f_2s will become the actual gates of the design. It guarantees that, in InterLock, only MUXes could be considered as the overhead, however, in Full-Lock, all inversions except one layer are surplus as an extra overhead. Hence, although InterLock adds extra logic, it even reduces the overhead compared to the Full-Lock.

To embed part(s) of the logic gates into each keyRB, we need a strategy to select the gates from the design. For the selection strategy, when the number of timing paths in

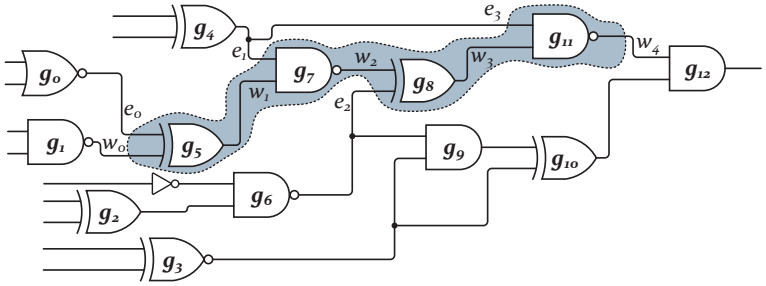
each keyRB is m, m timing paths must be selected[1] For $2log_2(m) - 2$ layers of SwBs in permutation-based network[2], the length of the timing path must be equal with $2log_2(m) - 2$. Hence, among the candidate timing paths, we select paths (or cut the paths) with length $2log_2(m) - 2$. For example, Fig. 6.2 shows how an actual timing path will be embedded into a keyRB in InterLock. For a 8×8 keyRB, we have $2log_2(m) - 2 = 2(3) - 2 = 4$ layers of SwBs. So, the timing paths must be the length of 4, and Fig. 6.2(a) shows a part of the timing path with a length of 4 that is selected to be embedded into the keyRB. Fig. 6.2(b) shows how this timing path is embedded into the keyRB. By using this approach, we embed m timing paths into a $m \times m$ keyRB allowing us to utilize the logic gates of each keyRB by 100%. Fig. 6.2(c) shows the top view of 8 different paths that are embedded into a keyRB while an arbitrary key has determined the connection between keyRB I/O.

6.2 Twisted Logic in Interlock vs. Full-Lock

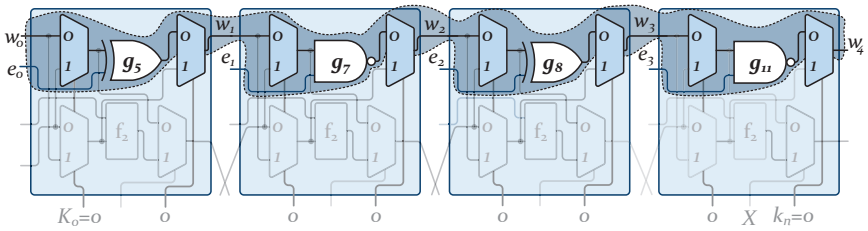
In Full-Lock, we showed that by adding a layer of configurable inverters into each SwB, the logic could be twisted with the keyRB. For example, Fig. 4.6(a) and 4.6(b) show that gates g_5 and g_12 are converted to its negated model ($\{XOR,\ XNOR\} \rightarrow \{XNOR,\ XOR\}$, and the inversion is handled within keyRB. Hence, to handle the inversion inside each keyRB, a layer of inversion is added into each SwB. However, such usage of the sequence of key-programmable inversion does not elevate the security of Full-Lock, and the KeyRB of Full-Lock could be simplified. More precisely, to attack the Full-lock, one could remove all inverters from the KeyRB (from all layers), and just add one layer of the key-programmable inverters at the end to reduce the key size while still maintaining the same function. This decouples the inversion from the routing block. Hence, in Full-Lock, the logic (inversion) and routing are not truly twisted. This allows us to simplify the KeyRB of Full-Lock before applying our *CP&SAT* attack to give maximum efficiency to the BVA. However, InterLock

[1]For a permutation-based $m \times m$ network, there are m different timing paths.

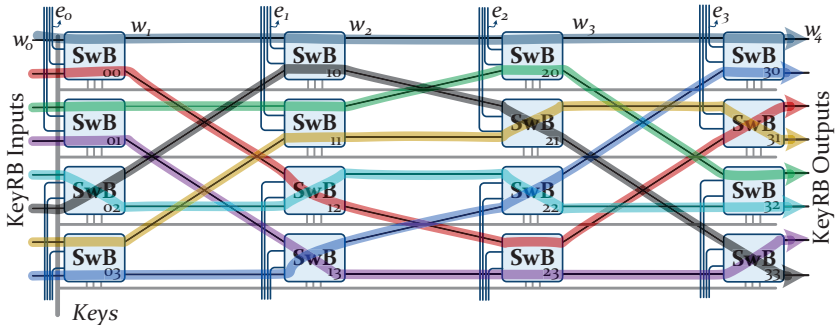
[2]The number of layers depends on the topology and being a blocking/non-blocking network. In this work, we use $2log_2(m) - 2$ layers of SwBs for $m \times m$ network that builds a near non-blocking permutation network.



(a) A Timing Path Selected to be Embedded into KeyRB



(b) Inserting the Selected Timing Path into a KeyRB



(c) 100% Utilization: m Timing Paths in a $m \times m$ KeyRB

Figure 6.2: Timing Path Embedding into KeyRB.

does not allow such simplification as f_1 and f_2 functions in each SwB are 2-input logic gates, and each input is (or could be considered as a) random and independent input.

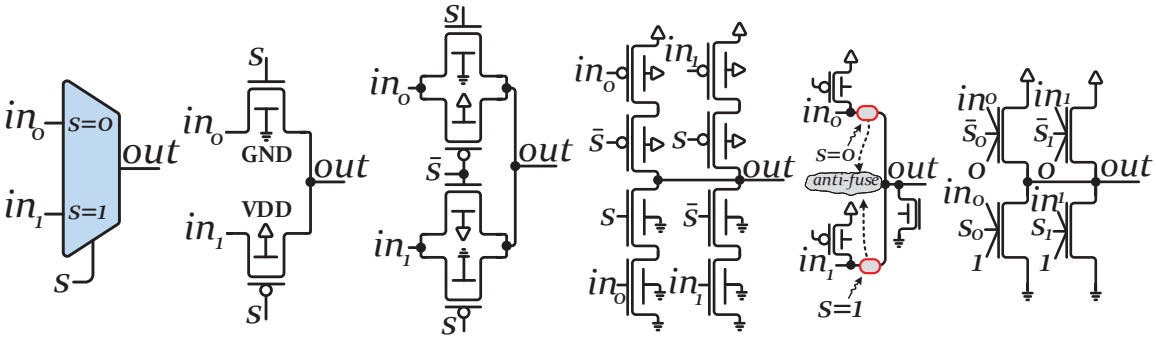
6.3 Area/Delay Overhead Exploration

At first glance, embedding routing-based locking incurs prohibited area/delay overhead. However, both Full-Lock and Cross-Lock engages some techniques to reduce the overhead to a reasonable ratio. Full-Lock shows how LUT insertion succeeding each keyRB allows

them to use a smaller size of the keyRB to guarantee the resiliency at lower overhead. Unlike Full-lock that is implemented at gate-level and based on static CMOS technology, Cross-Lock engages anti-fuse-based elements called programmable via (PVIA) elements to minimize the overhead ratio of each keyRB. To implement InterLock in this thesis, we examine both CMOS-based and PVIA-based implementation of keyRBs. Since MUXes are the only gate types that are used (overhead) for InterLock implementation, for CMOS-based implementation, amongst static logic, pass-transistor, or transmission gates, as demonstrated in Fig. 6.3(a)-6.3(d), we engage transmission-gates (Tgate) for MUXes based on tree-like structure [96, 97] that incur much lower overhead compared to static CMOS implementation. Also, as shown in Fig. 6.3(e), by using one-time-programmable elements (called PVIA elements in Cross-Lock [5]), we investigate the overhead of InterLock while implemented using anti-fuse-based (PVIA-based) 2:1 MUX. Similar to Tgate CMOS technology, we would use the tree-like structure to build keyRBs using PVIA elements.

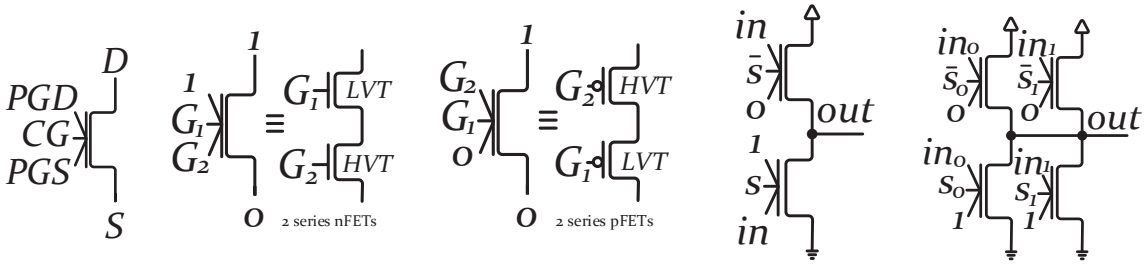
Apart from these two technologies, in this thesis, we assess the efficiency of another technology, called *Three-Independent-Gate Field Effect Transistors (TIGFET)*, for implementing MUXes in InterLock. In TIGFET technology, each transistor has *three* independent gates, and any two CMOS transistors could be modeled using only one TIGFET transistors, compacting the structure and achieving area as well as energy reduction, particularly for MUXes. Fig. 6.3(f) shows a 2:1 TIGFET multiplexer, and comparing with static CMOS each driving path consists of only one TIGFET transistors.

As shown in Fig. 6.4(a), Three terminal gates in TIGFET transistors called *Control Gate* (CG), *Polarity Gate at Drain* (PGD), and *Polarity Gate at Source* (PGS). Based on the value of these terminals, as illustrated in Fig. 6.4(b, c), one could build 2 series nFETs or 2-series pFETs. Since MUXes could be built using tristate inverters, in Fig. 6.4(d), we show how a tristate inverter could be built using TIGFET transistors. Compared to CMOS-based tristate inverter, the number of transistors is reduced by 50% in the TIGFET version. When m tristate TIGFET inverters are cascaded, a m : 1 MUX is built (e.g. 2:1 TIGFET MUX in Fig. 6.4(e)). It is worth mentioning that since the tristate inverter



(a) Symbol **(b)** PT CMOS **(c)** TGate **(d)** Static **(e)** Fuse **(f)** TIGFET

Figure 6.3: Different Multiplexer Implementation Possibilities.



(a) Transistor **(b)** 2 Series nFETs **(c)** 2 Series pFETs **(d)** Tristate INV **(e)** 2:1 MUX

Figure 6.4: 2:1 TIGFET MUX Implementation.

is used for each path of the multiplexer, the control signal (MUX selector) needs to be decoded. Hence, since all MUXes are controlled by the keys, and since we only use 2:1 MUXes, decoding selectors doubles the number of selectors (key inputs) in this technology. In our experimental results, we compare the implementation and the overhead of all three technologies.

6.4 Robustness/Overhead Evaluation of InterLock

To evaluate the proposed *InterLock*, as a countermeasure against *CP&SAT* attack, we implement keyRBs from Fig. 6.1(b) on the same benchmark circuits to acquire locked

circuits. We apply both the SAT (CycSAT-I) and our proposed *CP&SAT* attack on locked circuits by InterLock. All the experiments are implemented using Python/C++ and have been carried out on many Dell PowerEdge R620 equipped with Intel Xeon E5-2670 2.50GHz and 64GB of RAM. We evaluate the overhead of InterLock in three different technologies: (1) Transmission-Gate (Tgate) CMOS using Synopsys generic 32nm library, (2) PVIA-based MUXes that are manually added between the M2 and M3 layers as physical-only cells, and (3) Silicon NanoWire TIGFETs (TIG SiNWFETs 32nm) modeled using Verilog-A [98, 99].

6.4.1 Disabling the BVA using InterLock

To show how InterLock could be used as a countermeasure against the *Canonical prune and SAT* attack, we insert different numbers of the keyRBs from Fig. 6.1(b) into the benchmark circuits with different sizes. For the insertion of the keyRBs two strategies have been considered/applied in InterLock: (1) To minimize the performance degradation (delay overhead), the timing paths that are selected as the candidates to be embedded into keyRBs must be one of the highest positive slack timing paths, and (2) as shown in Table 6.1, since having more *XNOR*s (*XOR*s) increase the resilience of the locked circuits considerably, amongst the candidates, we select those paths that have more *XNOR*s (*XOR*s).

Table 6.2 shows the runtime of both the SAT and the *canonical prune and SAT* attack on circuits locked by InterLock. In both cases, since the locked circuit is possibly cyclic, for the SAT solving part, CycSAT-I has been used. As shown, for almost all cases, after inserting only two keyRB-16, both attacks fail to break the locked circuit.

Unlike previous routing-based locking techniques, BVA does not provide any advantage as a pre-processing step showing that truly twisting logic into the keyRBs guarantees the resistance against this new attack. Furthermore, compared to Full-Lock and Cross-Lock, twisting logic into keyRB allows us to engage smaller sizes of keyRB to guarantee the resiliency (keyRB-32/16 $\rightarrow$ keyRB-16/8). Shrinking the size of the keyRB with guaranteed security in InterLock allows the designer to considerably reduce area and delay overhead.

To illustrate that InterLock elevate the complexity of locked circuit, in Table 6.3, we

Table 6.2: The SAT Attack vs. *CP&SAT* on InterLock.

Circuit	Traditional SAT Attack (CycSAT-I)					proposed *canonical prune&SAT*				
	keyRB-8			keyRB-16		keyRB-8			keyRB-16	
	1	2	3	1	2	1	2	3	1	2
b15	98.5	3807.8	TO	74127.8	TO	85.3	3207.8	TO	69328.5	TO
b14	120.8	4229.1	TO	67203.2	TO	105.1	4028.5	TO	64328.6	TO
s35932	291.7	7126.4	TO	59372.1	TO	260.7	6992.1	TO	55221.4	TO
s38584	267.9	7624.4	TO	71375.5	TO	233.8	7168.5	TO	63298.8	TO
b20	284.4	11275.8	TO	58348.6	TO	186.7	8673.8	TO	55373.3	TO
b21	738.4	TO	TO	TO	TO	672.5	TO	TO	TO	TO
b17	320.4	6221.3	TO	77023.3	TO	271.9	5882.2	TO	74623.7	TO
b22	376.4	5209.9	TO	51042.4	TO	126.7	3862.7	TO	37621.2	TO
b18	701.9	32841.5	TO	TO	TO	630.3	30067.7	TO	TO	TO

Table 6.3: The SAT Attack Iterations on Routing Blocks.

Insance / Model	keyRB-4	keyRB-8	keyRB-16	keyRB-32	keyRB-64
Full-Lock	3-5	4-6	8-10	10-12	(5) Timeout
InterLock	8-12	16-22	30-33	(8) Timeout	(9) Timeout

compare the average number of iterations (N in Eq. 4.2) in Full-Lock with that of InterLock. Since we still get the benefit of routing-based locking, the number (M in Eq. 4.2) and the computational complexity (T^{Avg}_{DPLL} in Eq. 4.2) of recursive calls in DPLL algorithm is still extremely high in InterLock. However, as illustrated in Table 6.3, the average number of required iterations is increased by ∼3x-4x. This increase shows that $MN \times T^{Avg}_{DPLL}$ (from Eq. 4.2) is extremely higher in InterLock deepening the logic locking problem significantly.

6.4.2 Elevated Security at Lower Overhead

To perform a proof-of-concept physical design flow, we implement the keyRB in InterLock using three different 32nm technology: (1) Transmission-Gate (Tgate) CMOS, (2) PVIA-based MUXes, and (3) TIGFET SiNWFETs using Verilog-A. Table 6.4 shows the area/power/delay overhead of locked circuits via three keyRB-8, which is resilient against

Table 6.4: InterLock Overhead in: Tgate, Anti-fuse, and TIGFET.

Circuit	Original			3×keyRB-8								
				Tgate CMOS			PVIA Anti-Fuse			TIGFET		
	a (μm^2)	p (μW)	d (ns)	a %	p %	d %	a %	p %	d %	a %	p %	d %
b15	5292.7	327.6	1.23	34.5%	27.8%	12.9%	27.6%	21.9%	7.1%	24.5%	20.1%	6.4%
b14	5707.9	423.9	1.55	22.6%	17.5%	14.6%	19.5%	16.2%	8.8%	18.6%	15.4	6.8%
s35932	9283.1	729.8	1.68	30.7%	22.8%	10.9%	27.3%	20.7%	6.5%	24.8%	18.7%	5.1%
s38584	11003.2	806.6	1.77	24.1%	19.1%	19.7%	22.4%	17.3%	10.7%	20.7%	16.7%	8.3%
b20	11752.5	755.6	2.34	19.8%	15.5%	12.8%	17.6%	13.9%	7.1%	15.9%	13.4%	5.6%
b21	13007.1	922.7	2.21	16.2%	12.7%	10.7%	14.3%	11.2%	5.7%	12.9%	10.4%	4.5%
b17	15573.3	1245.1	3.58	8.9%	7.4%	7.6%	7.8%	6.4%	4.2%	7.2%	5.9%	3.4%
b22	16582.7	1319.5	3.18	6.4%	4.9%	8.5%	5.9%	4.1%	4.3%	4.8%	3.9%	3.5%
b18	57626.9	4834.1	3.81	3.7%	2.1%	3.2%	3.5%	1.8%	1.9%	2.7%	1.4%	1.6%

the SAT and the *cnonical prune-and-SAT* attack. Compared to Full-Lock which is a gate-level implementation of MUXes using static CMOS, in InterLock, Tgate-based implementation at transistor level would considerably reduce the area/delay overhead. In many cases it was expected to observe that PVIA-based MUXes could achieve the most optimum results; However, since there is no automatic flow in existing EDA tools for optimization of a large number of PVIA-based elements, the insertion has to be done manually. To do it manually, we inserted the PVIAs in a grid and push the standard cells away from this PVIA grid to successfully perform placement, and due to fine-granularity of MUXes in the circuit (small units and a large amount of usage), and since the number of PVIAs that must be used is a lot, in many cases we faced DRC violations leading us to use much lower utilization rate.

Based on our evaluation of these three different technologies, as shown in Table 6.4, TIGFET-based keyRB could bring more efficiency compared to Tgate CMOS and PVIA-based implementation. As shown, on average, TIGFET could reduce the area/delay overhead by up to 20%/56% compared to Tgate-based CMOS keyRB. However, for two important reasons, in all three technologies, on average, the overhead is less than 10%, which is completely acceptable: (1) The required number/size of keyRBs is less/smaller in InterLock, and (2) The actual timing paths selected for embedding are paths with highest positive slack time.

Chapter 7: SCRAMBLE: Logic and Routing Locking for Scan/Sequential Locking

Although the SAT attack (and many of its derivatives) only works on combinational circuits [22], having access to the DFT structure. i.e. scan chain pins, allows an adversary to apply the SAT attack, or its derivatives, on each combinational logic part of sequential circuits separately. Accordingly, the adversary can target sequential circuits using the SAT attack. Hence, few recent studies investigated the possibility of restricting the scan chain using scan chain locking/blocking [48, 50, 54, 56]. Also, considering that the access to the scan chain is restricted/locked, several studies investigated the possibility of applying the logic locking to the whole sequential circuits [100, 101], particularly FSMs [100–105]. However, further research revealed that new attacks could be formulated for these locking solutions even while access to the scan chain is restricted/locked.

In case of FSM locking [102–104, 106], a new attack, without oracle access, denoted as ***2-stage attacks on FSM*** (2-stage) was formulated [100, 105]. Also, in case of sequential (datapath) or scan chain locking [48, 54, 56, 100, 101], a new breed of SAT-based attacks, referred as ***unrolling-based SAT attack*** (UB-SAT) as well as SAT attacks integrated with bounded-model-checking (BMC) was formulated [60–62], challenging the validity of these solutions.

To defend against UB-SAT or BMC, and to break 2-stage attacks on FSMs, in this Chapter, we introduce a new state, connectivity and routing augmentation model for building logic encryption (SCRAMBLE). SCRAMBLE is designed to (a) add and maximize the *false transitions* within state transition graph (STG) when FSM is targeted for locking, (b) add and maximize the *false connections* between datapath flip flops (FFs) when sequential datapath locking is targeted, and (c) add and maximize the *false sequences'* possibilities in

scan FFs (SFFs) when the scan chain is targeted. SCRAMBLE, with 2 variants, can resist both 2-stage attacks on FSM as well as UB-SAT or BMC attacks on sequential datapath or scan chain locking.

7.1 FSM, Sequential Datapath, and Scan Chain Locking

In FSM locking [102–104,106], few extra sets (modes) of states are added to the original state transition graph (STG), such as locking/authentication mode states or black holes. The traversal sequence of locking/authentication modes is the locking/authentication key, and a correct traversal that reaches the initial state of the original STG unlocks the FSM. Also, the output generated by the correct traversal of authentication states serves as a watermark. In addition to these groups, a set of studies focuses on locking the STG without adding any extra state. However, the complexity and overhead (area) of this approach is higher compared to other schemes [100].

In sequential datapath locking or scan chain locking, XOR- or MUX-based locking has been added into timing paths or the scan chain. For instance, in scan chain locking solutions, the scan chain has been blocked or locked to prevent the scan chain loading and readout (shift/load) for protecting the logic against the SAT attacks. For example, the *Encrypt Flip-Flop* solution [48] adds key-programmable MUXes on the outputs of SFFs enabling the negation of the SFFs when the scan chain locking key is incorrect.

7.2 Attacks on FSM, Sequential, and Scan Chain Locking

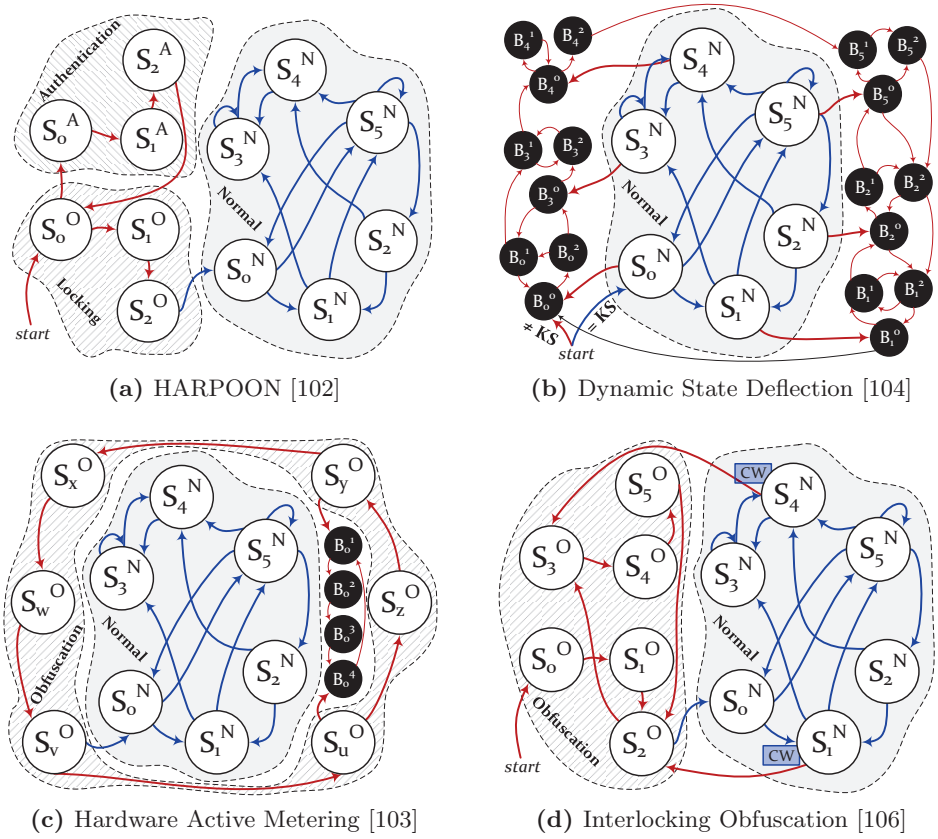
To assess the strength of FSM locking solutions, many studies evaluated the possibility of deploying 2-stage attacks, as an *oracle-less* attack, on locked FSMs [100, 105]. The 2-stage attacks on FSM are composed of: (1) ***(stage 1): topological analysis*** (described in line 2-13 of Algorithm 4), which is a detection algorithm to find FFs that are responsible for storing the state values (separating them from datapath FFs), and ***(stage 2): functional analysis***

Algorithm 4 2-stage on FSM Locking [100, 105]

```
1:  function FSM_Extract(Circuit C_L)
2:      SFF ← [];                                          ▷ State Flip Flops
3:      RS ← classify(FFs);                                ▷ Classifying FFs into Register Sets
4:      for each set ∈ RS do
5:          set ← set - notSCC(set);                       ▷ Keeps Strongly Connected Components
6:          if is_splittable(set) then
7:              RS ← {RS - set} ∪ split(set);
8:      CLFP ← find_feedback_circuits(C_L, Reg_Sets);
9:      for each set ∈ RS do
10:         set ← set - notInfDep(set);                    ▷ Keeps Intersected Influence/Dependence
11:         set ← set - InputIndependt(set);               ▷ Check Control Metrics
12:         update(CLFP);
13:         SFF ← SFF ∪ set;
14:     S_0 ← initial_state(state_regs); SQ ← [];          ▷ State Queue
15:     SQ ← SQ ∪ S_0; STF ← [];                           ▷ State Transition Table
16:     while SQ ≠ [] do
17:         state ← SQ.dequeue();
18:         for each DIP do                                ▷ DIP found by SAT
19:             if eval(state_regs, DIP, state) ∉ SQ then
20:                 SQ.enqueue(nx_state);
21:                 STF ← STF ∪ {state, DIP, nx_state, PO}
        return SQ, S_0, STF;                               ▷ States, Initial, Transition Func.
```

(described in line 14-21 of Algorithm 4) that finds the STG based on the list of FFs found in (stage 1). In such an attack, the topological analysis, which is derived from [107], identifies FFs whose input contains a combinational feedback path from their output. Then, it reduces the set of possible state FFs by (a) grouping the FFs controlled by the same set of signals, and (b) finding strongly connected components (SCC) using Tarjan's algorithm [100, 105, 108, 109]. In the functional analysis stage (stage 2), the attacker attempts to extract/re-draw the STG. This is done by first attempting to find the initial state, and then identifying the reachable states by creating a reduced binary decision diagram (BDD) or using a SAT solver. After re-drawing STG by using a 2-stage attack, in most FSM locking solutions, the adversary can readily distinguish the original part of the FSM from either extra added states or extra state transitions, leading to extracting the original FSM. Fig. 7.1 illustrates four examples of FSM locking. As shown, the original part of the FSM is easily distinguishable from extra locked states in these solutions.

In UB-SAT or BMC [60–62] on the other hand, as a much stronger attack that is applicable to all FSM locking, sequential datapath locking, and scan chain locking, the



(a) HARPOON [102]

(b) Dynamic State Deflection [104]

(c) Hardware Active Metering [103]

(d) Interlocking Obfuscation [106]

Figure 7.1: FSM Obfuscation Solutions.

adversary does not need to have access to the scan chain. In these attacks, the adversary unrolls the reverse-engineered netlist n times and then creates a double circuit similar to the SAT attack. Then, the adversary uses a SAT solver to find a sequence of n inputs and two key values such that the output of the meter (double) circuit detects a mismatch. Such an input sequence is denoted as a *discriminating input sequence* (DIS). The attacker increases the unrolling depth (n) until a termination condition is met. The overall flow of this breed of attacks is captured in Algorithm 5. By using this structure, the adversary can target and attack any sequential logic locking solution even while the access to the scan chain is restricted. Also, as an enhanced version of this group of attacks, KC2 [62], accelerates the original UB-SAT [60] by using some additional simplification steps. Some of the added features include (a) integrating BMC with SAT, (b) model conversion to BDD to simplify

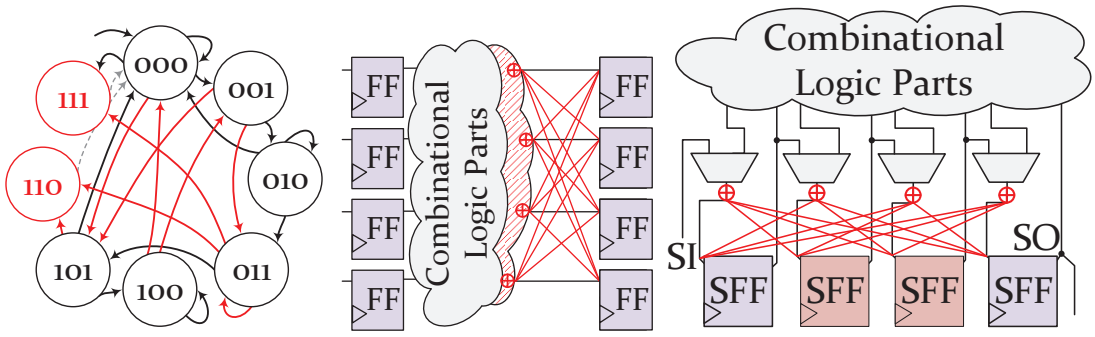
the circuit representation, and (c) constraint simplification using key-sweeping. Similarly, the ScanSAT [61] is another unrolling-based SAT attack that only focuses on scan chain locking solutions, which is applicable to both statically and dynamically scan chain locking solutions.

Algorithm 5 UB-SAT on Sequential/Scan Locking [60–62]

```
1:  function UB_SAT(Circuit C)
2:      b ← initial_boundary;
3:      Terminated ← False;
4:      MC_circuit ← C(X, K1, Y1) ∧ C(X, K2, Y2) ∧ (Y1 ≠ Y2);
5:      while !Terminated do
6:          while (X_DIS, K1, K2) ← BMC(MC_circuit, b) = T do
7:              Y_f ← C_BlackBox(X_DI);
8:              MC_circuit =∧  C(X_DIS, K1, Y_f) ∧ C(X_DIS, K2, Y_f);
9:          if not BMC(MC_circuit, b) then                          ▷ UC
10:             Terminated = True;
11:         else if not BMC(MC_comb_circuit, b) then                ▷ CE
12:             Terminated = True;
13:         else if UMC(MC_circuit, b) then                         ▷ UMC
14:             Terminated = True;
15:         else
16:             b ← b + boundary_step;
17:     KeyGenCircuit = DIVC ∧ (K1 = K2)
18:     Key ← BMC(emphKeyGenCircuit, b)
```

7.3 Proposed Scheme: SCRAMBLE

In SCRAMBLE, we engage the term *augmentation* to refer to the process illustrated in Fig. 7.2. *Augmentation* in SCRAMBLE adds false state transitions in case of FSM locking, or adds false FF-to-FF timing paths in case of sequential datapath locking, or adds false scan chain sequence in case of scan chain locking. SCRAMBLE is proposed in two variants: (1) The first variant is *connectivity* SCRAMBLE (SCRAMBLE-C) that hides the connectivity to the targeted FFs using logarithmic switching network. (2) The second variant is *logic* SCRAMBLE (SCRAMBLE-L) that hides the logic by implementing part(s) of the logic within memory. The SCRAMBLE-C could be used for locking either FSMs, sequential datapath, or scan chains, to protect the locked design against all UB-SAT and BMC attacks, such as KC2 or ScanSAT. SCRAMBLE-L, on the other hand, is mostly applicable to FSMs



(a) for FSM Locking **(b)** for Sequential Locking **(c)** for Scan Chain Locking

Figure 7.2: *Augmentation* Model in SCRAMBLE

to provide resilience against 2-stage attacks [100, 105].

7.3.1 SCRAMBLE-C

The overall structure of SCRAMBLE-C has been illustrated in Fig. 7.3. In SCRAMBLE-C, the connectivity between the targeted FFs and their *fan-in-cone*s (FiCs) (combinational logic cones) is locked. Hence, before connecting the output of corresponding FiCs to the FFs, a key-programmable routing block (KeyRB) has been inserted to control the connections. For instance, in Fig. 7.3, a KeyRB with size 8 has been inserted before a combination of FSM FFs and datapath FFs. Similar to Full-Lock and InterLock, the KeyRB, which must be inserted between the targeted FFs and their corresponding FiCs, is a key-programmable switching network that is built using self-routing logarithmic ($log_2(N)$) networks [18, 78]. As discussed previously, the $log_2(N)$ networks, compared to the existing and well-known switching networks, such as mesh, crossbar, etc., incur less area overhead. Also, we demonstrated that due to its cascading structure, $log_2(N)$ networks could improve the robustness against the SAT attack.

As shown in Fig. 7.3, the KeyRB is built using key-programmable 2x2 switch-boxes (SwBs). However, as shown in this example, in this case, one layer of the MUXes that were designated for inversion in Full-Lock has been omitted, and to add the capability of

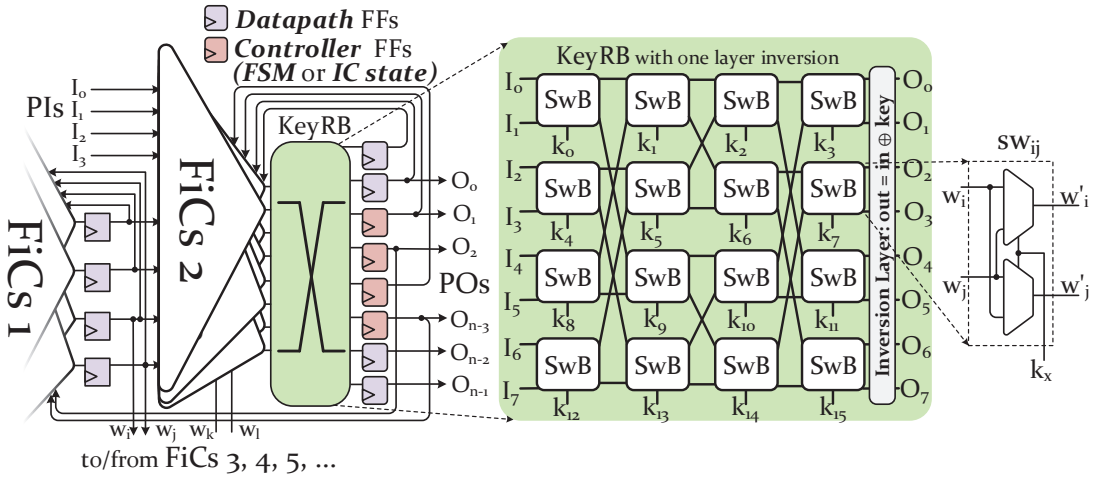


Figure 7.3: *Augmentation* using *shuffle*-based KeyRB in SCRAMBLE-C.

logic programmability, we only add one extra key-controlled (XOR) inversion layer, as the final layer of KeyRB. Based on its key, a SwB saves or changes the order of inputs while connecting them to output pins. Also, the connection between the layers of *sw*s is fixed. This inter-layer connection determines the topology of the $log_2(N)$ network. For instance, the topology of the sample KeyRB demonstrated in Fig. 7.3 is *shuffle* topology.

In SCRAMBLE-C, the KeyRB must be inserted before the targeted FFs. When FSM locking or sequential datapath locking is targeted, during either the physical design or after DFT synthesis step, the KeyRB is placed on wires that connect the outputs of FiCs to the *data-in (DI) pin* of targeted FSM FFs or datapath FFs. When scan chain locking is targeted, after DFT synthesis, the KeyRB is placed in *scan network* before the *scan-in (SI) pin* of the targeted SFFs.

Although engaging self-routing $log_2(N)$ networks provides a low-overhead routing locking solution, similar to Full-Lock, we have to address a few issues even while it targets sequential circuits: (1) What is the best size for the $log_2(N)$ KeyRB to make the sequential circuit robust against the UB-SAT attack or 2-stage attacks on FSMs; and (2) How we can build more permutations using $log_2(N)$ in FSM, sequential, and datapath locking. Hence,

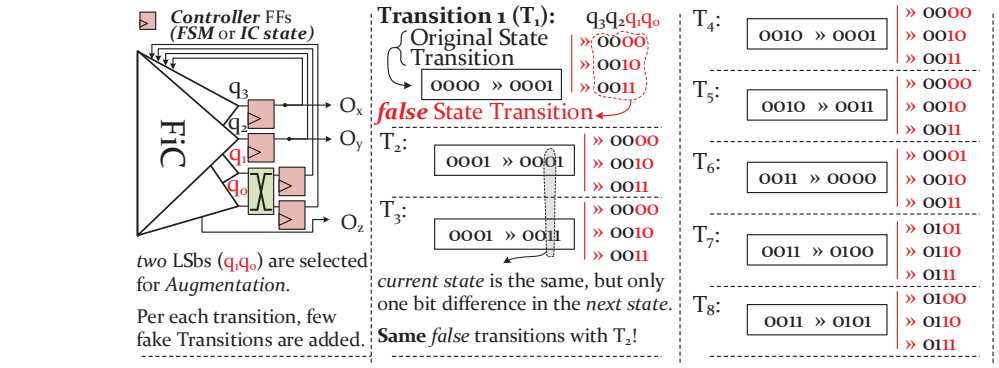
the number of wires ((N) as the inputs of KeyRB must be small enough to make the network overhead reasonable; and large enough to make the number of permutation enough. It raises two questions: (1) which N FFs must be selected? and (2) How we can minimize N?

Selection of N FFs

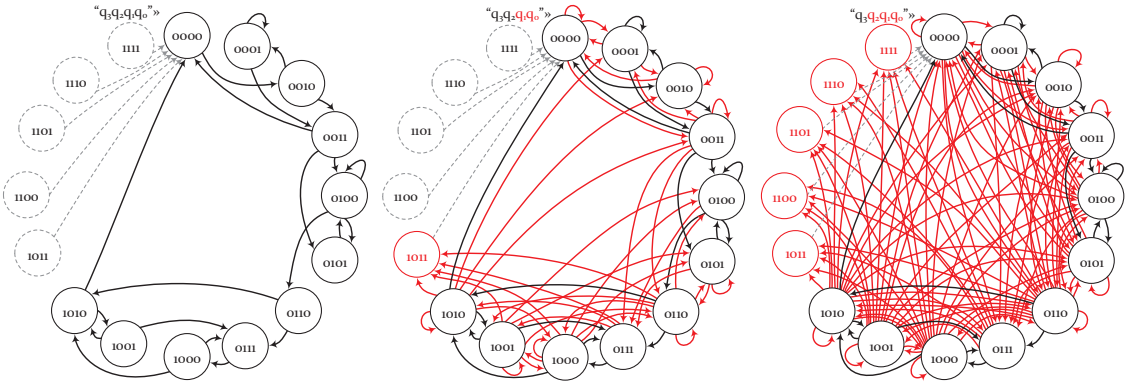
The selection of FFs (to insert KeyRB before them) in SCRAMBLE-C could significantly impact its locking strength, particularly in FSM locking. For example, let us consider the engaging of SCRAMBLE-C for an FSM presented in Fig. 7.4(b), which is generated using binary encoding of 4 FFs. In this example, if we select *two* least significant bits (LSB) FFs to insert a KeyRB with size 2 before them (circuit of Fig. 7.4(a)), the locked FSM is demonstrated in Fig. 7.4(c). Fig. 7.4(a) shows how the false transitions in Fig. 7.4(c) have been generated for some of the original transitions. As shown in 7.4(c), when 2 LSBs are selected, a limited number of false transitions are added, and only one extra (unreachable) state is visited. However, if we select *two* most significant bits (MSB) FFs, it will visit all extra states and generates a large number of false transitions demonstrated in Fig. 7.4(d). Since 2-stage attacks are applicable to FSM locking, maximizing false transitions as well as extra states makes SCRAMBLE-C more robust against this attack. Accordingly, being aware of the encoding style of FSM will impact its locking strength. For instance, in binary encoding, a synthesis tool usually encodes the states from low to high (0 to 2^{N-1}). Hence, using the N FFs representing the MSB of state value results in the inclusion of the largest number of previously unreachable states and false transitions in the locked FSM.

Reducing N by making KeyRBs near non-blocking

As previously discussed, the implementation of blocking $log_2(N)$ network revealed that even a 256-input KeyRB could be broken by the SAT attack in less than an hour. Hence, to address the blocking nature of KeyRB and to resist against UB-SAT or BMC attacks (with a small KeyRB), we expand the $log_2(N)$ network towards non-blocking via adding extra



(a) FSM circuit and transitions generation



(b) the original FSM **(c)** 2 LSBs to SCRAMBLE-C **(d)** 3 LSBs to SCRAMBLE-C

Figure 7.4: Using SCRAMBLE-C for FSM Locking.

(cascaded) stages. To move close enough towards non-blocking nature without incurring large area overhead, we used the *"near non-blocking"* structure [78]. In near non-blocking, not all but *almost all* permutations are feasible, while it could be implemented using a $LOG_2(N, log_2(N)-2, 1)$, meaning it has only $M = log_2(N)-2$ extra stages and no additional copy ($P = 1$). The KeyRB depicted in Fig. 7.3 is an example of a near non-blocking KeyRB for 8 inputs. Our implementation shows that a 32-input near non-blocking network ($LOG_2(32, 3, 1)$) is far stronger against the SAT attack compared to a 256-input blocking network $log_2(256)$, while it is 8x smaller.

7.3.2 SCRAMBLE-L

In SCRAMBLE-L, which is proposed for FSM locking against 2-stage attacks, the logic located before the targeted FFs is locked using in-memory computation. As shown in Fig. 7.5, in SCRAMBLE-L, a small part of the combinational logic in the FiCs of the targeted FFs is replaced with a one-cycle read memory, such as SRAM. As an example, FiC_2 and FiC_4 are replaced with equivalent memories. The content of the memories must provide the same output compared to that of FiC_2 and FiC_4 while the triggering input is the same. Hence, the truth table corresponded to those FiCs must be generated and stored in the memories. The memories would be initialized during boot-up from a tamper-proof NVM which serves as the key storage. To avoid additional delay incurred by memories, the selection of FFs must be done with respect to their available timing slack.

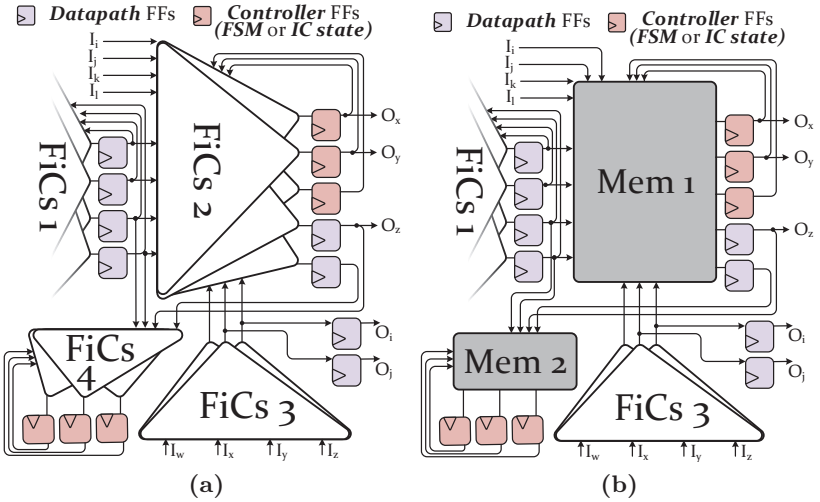
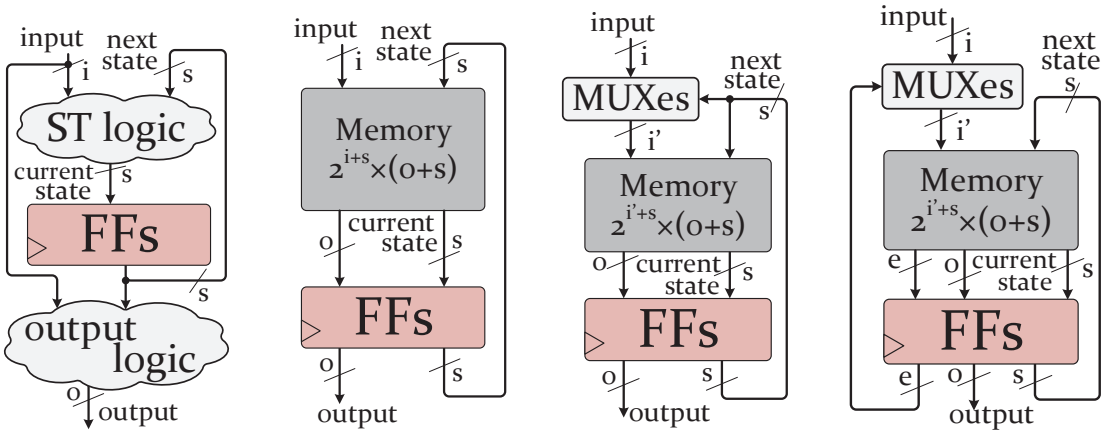


Figure 7.5: Sequential Circuits Locking using SCRAMBLE-L.

SCRAMBLE-L makes 2-stage attacks almost impractical. Considering that the adversary has no access to the contents of memories after reverse engineering, there is no equivalent logic for the memories, and the BDD- or SAT-based *functional analysis* (stage 2) cannot be accomplished on the locked circuit. Also, similar to Fig. 7.5(b), if the designer selects a combination of datapath FFs and FSM FFs, the adversary cannot distinguish



(a) Traditional **(b)** Memory-based **(c)** Memory-based with Input Multiplexing 1 **(d)** Memory-based with Input Multiplexing 2

Figure 7.6: Different FSM Implementation Models.

between them when deploying *topological analysis* (stage 1) of the 2-stage attack, resulting in the inclusion of an extremely large number of non-FSM FFs in the candidate FSM FFs. Hence, none of the existing 2-stage attacks can be applied to SCRAMBLE-L.

The big challenge with the SCRAMBLE-L is the size of the memory for implementing the selected FiCs. However, since SCRAMBLE-L is proposed for FSM locking, this problem could be easily addressed by engaging the FSM input multiplexing (FSMIM) techniques [110]. In this technique, considering that the next state and the outputs of the FSM are a function of a subset of the inputs (not all), a set of multiplexers has been used to select only those signals that affect the next state and the output. Hence, the designer is able to minimize the number of inputs to the memories (as address width), resulting in a significant decrease in the size of memory. The main difference between traditional FSM implementation, memory-based FSM, and FSMIM has shown in Fig. 7.6.

In FSMIM, multiplexers could be controlled using two different strategies: (1) using the value of the current state, (2) using code-words stored in the memory. The first option is more efficient in terms of memory size reduction; however, the second method has better efficiency in reducing the multiplexers complexity. Hence, the first option has been used in

Table 7.1: Simplification Ratio of Input Multiplexing (FSMIM).

FSM	Required Memory Size and Additional MUXes						
	Traditional	Input MUXing			Input MUXing + State Reduce		
	$Size_{Kb}$	$Size_{Kb}$	MUX	Reduction	$Size_{Kb}$	MUX	Reduction
s510	435,500	5.5	14, 5	∼99.9%	2.5	14, 7	∼99.9%
s820	195,000	255	5, 4, 3, 2	∼99.9%	38	7, 6, 4, 4, 2, 2. 2	∼99.9%
s832	200,000	262.5	5, 4, 3, 2	∼99.9%	69	5, 4, 4, 4, 2	∼99.9%
s1488	408,000	110,500	2, 2	73%	16,000	4, 4, 2, 2, 2	92.5%
s1494	408,000	110,500	2, 2	73%	16,000	4, 4, 2, 2, 2	92.5%

Table 7.2: The Effectiveness of SCRAMBLE in FSM/Sequential/Scan Obfuscation.

Variants	SCRAMBLE-C		SCRAMBLE-L	
Attacks	2-stage	UB-SAT *or* BMC	2-stage	UB-SAT *or* BMC
FSM	✓	✓	✓	✓
Sequential Datapath	N/A	✓	N/A	✓*
Scan-chain	N/A	✓	N/A	✓*

∗: Requires large augmentation model incurring area overhead.

SCRAMBLE-L to minimize the area overhead of the memories. Our evaluation in Table 7.1 illustrates the effectiveness of FSMIM when applied to the ISCAS-89 benchmarks, resulting in memory size reduction above 90%.

7.4 LUT-based Remapping of SCRAMBLE

Table 7.2 shows the effectiveness of each variant of SCRAMBLE against 2-stage and UB-SAT or BMC attacks. Although the main aim of SCRAMBLE-C is to protect the design against UB-SAT and BMC, it also breaks 2-stage attacks. Similar to SCRAMBLE-L, if we use a combination of both datapath FFs and FSM FFs as input to SCRAMBLE-C (Similar to Fig. 7.3), topological analysis (stage 1) of 2-stage attack cannot detect the correct set of FSM FFs. Therefore, the functional analysis (stage 2) has to generate the STG using an incorrect set of FFs (extremely larger set), resulting in a significant increase in the attack

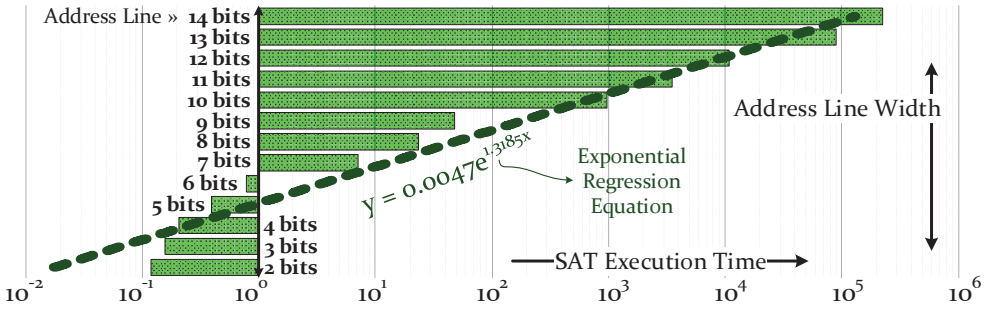


Figure 7.7: SAT-based Memory Modeling for Different Address Width.

time with respect to the number of additional (datapath) FFs included in the set. Also, the extracted STG is constructed using a combination of datapath FFs and FSM FFs, which leads to an incorrect STG, and the adversary is not able to extract the original FSM from the extracted STG.

Although SCRAMBLE-L protects the design against 2-stage attacks by hiding the logic within memory, the adversary can generate the equivalent logic of the memory (X input (address width) and Y outputs (word size)) by replacing it with Y of X-input LUTs, which is a fully configurable logic, and then the SAT attack could be invoked on the LUT-based remapped model. However, increasing the input size of the LUTs exponentially increases the execution time of BDD-based or SAT-based attacks. Fig. 7.7 shows that by increasing the address width (from 2 bits to 14 bits), when we replace the memory with the same size LUTs, SAT execution time increases exponentially. In addition, due to the unrolling structure of UB-SAT or BMC, these LUTs must be replicated per each iteration (per each unrolling), which makes them almost unresolvable by SAT-driven attacks. We demonstrate that UB-SAT or BMC cannot reveal the correct functionality of a design even while SCRAMBLE-L has been used with only one 256 words (address width is 8) memory.

7.5 Robustness Evaluation

We evaluated the strength of SCRAMBLE on two sets of benchmark circuits: (1) sequential ISCAS-89 benchmark circuits and (2) few well-known small-scale ASICs to large-scale

Table 7.3: Attack Execution Time on SCRAMBLE-C.

				Attack Time (second)						Datapath Locking		
				scanSAT			BMC			PPA overhead of		
				KeyRB Size			KeyRB Size			16-input KeyRB		
Circuit	#FF	#Gate	In/Out	8	16	32	8	16	32	Power	Delay	Area
s1196	18	529	14/14	2029	✗	**N/A**	1109	✗	**N/A**	26.3%	36.5%	24.1%
s1423	74	657	17/5	3441	✗	✗	438.6	9356	✗	25.8%	28.1%	23%
s5378	179	2779	35/49	6406	✗	✗	6921	✗	✗	8.9%	18.5%	7.1%
s9234	211	5597	36/39	1801	✗	✗	1548	✗	✗	5.1%	14.8%	3.9%
s15850	534	9772	77/150	5882	✗	✗	7097	✗	✗	3.1%	12.9%	2.4%
s35932	1728	16065	35/320	8604	✗	✗	7110	✗	✗	1.1%	6.5%	0.9%
s38584	1426	19253	38/304	4072	✗	✗	6287	✗	✗	1.2%	5.7%	0.9%

✗ *: Timeout* $= 10^6$ *Seconds*

microprocessors. We have deployed a 2-stage attack according to Algorithm 4 to assess the strength of SCRAMBLE in FSM locking. For sequential datapath locking, we deployed an integrated BMC with SAT [60] accelerated using stages described in KC2 [62]. Finally, to assess the effectiveness of scan chain locking, we implemented the ScanSAT as described in [61]. All attacks are carried on a Dell PowerEdgeR620 with Intel Xeon E5-2670 2.50GHz and 64GB of RAM.

Table 7.3 captures the execution time of scanSAT [61] (for scan locking) and accelerated BMC [60, 62] (for sequential datapath locking) while SCRAMBLE-C is used on ISCAS-89 benchmarks. The maximum runtime of attack is set to 10^6 seconds, and attack will time-out (✗ in tables) if attack time exceeds the limit. Note that in some cases, the number of required FFs is limited. For instance, in *s1196*, with 18 FFs, the maximum possible size of KeyRB is only 16. As illustrated, by utilizing the KeyRB with size 16, for almost all benchmark circuits, both attacks cannot retrieve the keys. Also, Table 7.3 reports the power, performance, and area (PPA) overhead of SCRAMBLE-C with a KeyRB of size 16. While the KeyRB size is fixed, the area overhead is constant and the percentage area overhead reduces when the size of the benchmark circuits increases. As shown, for even mid-size ISCAS-89 benchmark circuits, the area overhead is less than 10%.

Table 7.4: Attack Execution Time on SCRAMBLE-C and SCRAMBLE-L.

			Attack Time (second)											
			BMC						2-stage					
			SCRAMBLE-C KeyRB Size			SCRAMBLE-L Mem Addr			SCRAMBLE-C KeyRB Size			SCRAMBLE-L Mem Addr		
Circuit	#FF	#Gate	8	16	32	7	8	9	8	16	32	7	8	9
RS232	168	59	2.7	2029	✗	35.7	✗	✗	✗	✗	✗	✗	✗	✗
32b RSA	555	2139	1.4	3441	✗	45.8	✗	✗	✗	✗	✗	✗	✗	✗
MC8051	578	6590	47.7	6406	✗	50.1	✗	✗	✗	✗	✗	✗	✗	✗
SPARC	120K	233K	938	✗	✗	288.2	✗	✗	✗	✗	✗	✗	✗	✗

✗ : *Timeout* = 10^6 *Seconds*

Table 7.5: The PPA Overhead of Resilient SCRAMBLE-C and SCRAMBLE-L

	SCRAMBLE-C (Resilient with KeyRB Size = 32)				**SCRAMBLE-L** (Resilient with SRAM Size = $2^8 \times 8$)			
Circuit	RS232	32b RSA	MC8051	SPARC	RS232	32b RSA	MC8051	SPARC
Area (%)	38.5%	4.5%	1.2%	0.05%	173%	17.8%	5.1%	0.1%
Power (%)	44.8%	5.6%	1.7%	0.1%	224%	26.8%	7.2%	0.3%
Delay (%)	48.4%	10.8%	11.4%	9.7%	22.7%	5.5%	6.8%	3.9%

To assess the robustness of SCRAMBLE for FSM locking, both SCRAMBLE-C and SCRAMBLE-L have been used on the second group of circuits. Also, the locked circuits have been evaluated using both BMC and 2-stage attacks. As illustrated in Table 7.4, BMC can break SCRAMBLE-C while the KeyRB size is up to 16. However, for none of the circuits, BMC cannot retrieve the correct key while the KeyRB size is 32. Also, in case of BMC, only utilizing a memory with 256 words (address width = 8) is enough to make the locked circuit resilient against BMC.

Unlike BMC, which can break SCRAMBLE for small-size KeyRBs and memories, 2-stage attacks are far weaker. As shown in Fig. 7.4, since the number of false paths is extremely larger, after re-drawing the FSM using 2-stage, there is no chance for the adversary to extract the original part of the FSMs. Hence, as shown in Table 7.4, 2-stage attacks completely fail against SCRAMBLE.

Table 7.6: The PPA Overhead of SCRAMBLE with Different Sizes.

	SCRAMBLE-C (KeyRB Input Size)				**SCRAMBLE-L** (SRAM Size)			Sample ISCAS-89 Benchmarks	
Overhead	8	16	32	64	(2^7×8)	(2^8×8)	(2^9×8)	s15850	s38584
Area (um^2)	58.1	136.7	330.8	620.4	305.8	612.1	1119	6262	21458
Power (uW)	4.5	6.9	14.5	31.9	31.4	80.6	118.9	325.7	1031
Delay (ns)	0.33	0.37	0.48	0.56	0.17	0.18	0.19	1.24	2.68

Since the minimum size of KeyRB in SCRAMBLE-C (memory in SCRAMBLE-L), which provides a resilient FSM locking against BMC, is 32 (256 words), we reported the PPA overhead of these sizes for second groups of the circuits in Table 7.5. As shown, even for mid-size *32b RSA* circuit, the overhead is less than 5%. Also, the impact of increasing the size of KeyRB or memory on the PPA overhead for different sizes has been illustrated in Table 7.6. As shown, in both SCRAMBLE-C and SCRAMBLE-L, increasing either the size or address width, approximately doubles the overhead. However, compared to ISCAS-89 benchmark circuits, such as *s15850* or *s38584*, the incurred overhead is reasonable.

Chapter 8: kt-DFT: A key-trapped Design-for-Trust Architecture for Logic Locking

Blocking unauthorized access to the scan chain [46, 48, 54–56] limits the access of an adversary only to the primary inputs and primary outputs (PI/PO). However, with expanding the SAT attack, it was later shown, that an adversary can still attack a sequential circuit with no access to the scan chain by using unrolling-based (UB) SAT attack [60] or a bounded-model-checking (BMC) attack [61]. However, these sequential attacks are far weaker than the traditional SAT and are mostly applicable to moderately small sequential circuits. Since the sequential attacks are not scalable, by blocking the scan chain, and applying many of the prior logic obfuscation techniques, a moderately small-size locked circuit could easily resist such attacks.

Prior work on restricting unauthorized access to the scan chain could be divided into (1) scan chain locking [48, 54] and (2) scan chain blocking [55, 56]. In the scan chain locking techniques, such as encrypt flip-flop [48] or dynamically obfuscated scan (DOS) [54], the scan chain is statically or dynamically locked by inserting key gates. However, ScanSAT [61] could break both statically and dynamically scan chain locking techniques by transforming the locked scan chain into a combinational circuit and thereby launching the SAT attack on them (the unrolling-based SAT) [61].

In the scan chain blocking techniques, after loading the logic locking key (from tpNVM), the access to the scan-out(s) (SO) would be blocked [56]. By eliminating the access to the SO, an adversary's ability to monitor the behavior of the circuit will be limited only to the PO. This eliminates the possibility of the SAT attack as well as any attack that requires access to the scan chain, forcing an attacker to use the far weaker and non-scalable sequential attacks.

Table 8.1: Comparison of the State-of-the-art DFS architectures.

Defenses	Test Time	Test Complexity	Resilient against ScanSAT [61]	Shift&leak [57]	Glitch&Leak
EFF + RLL [48]	low	None	✗	✓	✓
R-DFS + SLL [56]	low	None	✓	✗	✓
mR-DFS + SLL [57]	high	key reload per pattern	✓	✓	✗
kt-DFS + SLL	low	None	✓	✓	✓

Scan chain blocking in the presence of logic obfuscation was first introduced in [56]. In the rest of this Chapter we refer to this solution as **r**obust **d**esign-**f**or-**s**ecurity (R-DFS). In addition to blocking the SO, the R-DFS also introduces a new storage element for holding the obfuscation key, denoted by secure cell (SC). However, the security of the R-DFS architecture was later challenged by the *shift-and-leak* attack [57]. To remedy the leakage issue, the authors proposed modification to the scan blocking architecture (we call it **mR-DFS**), equipping the SCs with a mode switch shift disable (MSSD) circuitry [57]. The mR-DFS blocks any shift operation after the obfuscation key is loaded from the tpNVM, removing the ability of an adversary to apply the shift-and-leak attack.

In this Chapter, by showing the architectural drawbacks of mR-DFS, we introduce our proposed DFS scan blocking architecture for protecting the logic obfuscation key. More precisely, the contributions of this work are as follows: (1) We illustrate how a glitch-based shift-and-leak attack allows an adversary to leak the logic obfuscation key even if the shift operation is disabled in mR-DFS, thereby, leaking the actual logic obfuscation key through the PO. (2) As a countermeasure, we propose a new key-trapped design-for-security (**kt-DFS**) architecture, where the scan chain that loads the logic obfuscation key is fully detached from regular scan chain(s). To fulfill this requirement, we propose a new secure cell design content of which cannot be shifted in the scan chain after a key registration event is observed. (3) We assess the security of proposed kt-DFS, and compare the proposed

solution with R-DFS and mR-DFS. As shown in Table 8.1, we will illustrate how the kt-DFS can support both structural and functional testing while resisting all leaky-based and SAT-based attacks on logic locking.

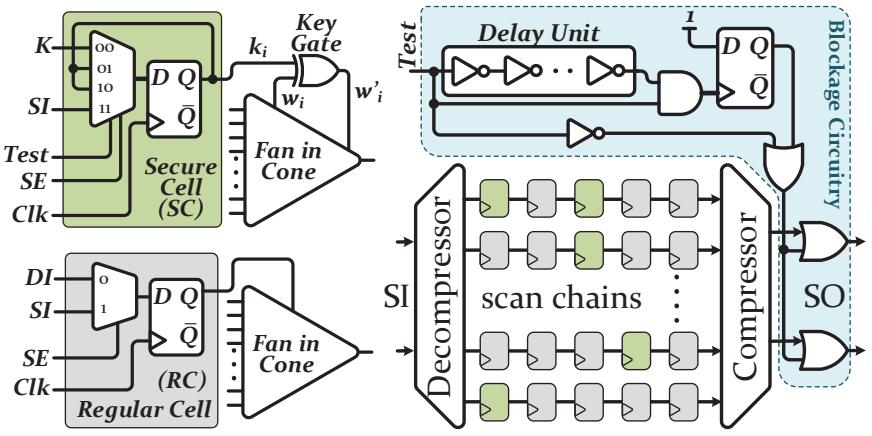
8.1 Background on Scan Blockage Techniques

Both R-DFS [56] and mR-DFS [57] block the SO pins after the obfuscation key is loaded into the design. The mR-DFS is built on top of R-DFS to fix the leakage issue. In the following section, we first describe how R-DFS works. Then, we explain the leakage issue identified in R-DFS, motivating the shift-and-leak attack. Then, we describe how mR-DFS remedies the problem with disabling shift operations after loading the obfuscation key.

8.1.1 R-DFS: Restricting Scan Access

In R-DFS [56], the obfuscation key is stored in a custom-designed scan (storage) cell, denoted as secure cell (SC). As shown in Fig. 8.1(a), in R-DFS, each key value is stored in one SC. The R-DFS architecture, as indicated in Table 8.2, allows four types/modes of operation based on the $Test$ and SE pins. The key values could be loaded into SCs either directly from tpNVM (actual key values in mode M_0) or the scan-in (dummy/actual key in mode M_2). The scan chains, as shown in Fig. 8.1(b), are constructed by stitching the SCs with regular scan Flip-Flops (SFF). The SFFs in this paper are denoted as Regular Cells (RC). The SCs keep their previous values in modes M_{1a} and M_{1b}. The only difference between the M_{1a} and M_{1b} mode is the value of the SE pin that determines the shift/capture mode in RCs. Both of the M_{1a} and M_{1b} modes allow the SCs to be bypassed (keeping their previous values) when the RCs are in shift/capture mode.

For the structural (a.k.a manufacturing fault) test, the $Test$ pin must be 1, allowing the shift and capture operations to be carried in modes M_2 and M_{1b} respectively, giving unrestricted access to the scan. On the other hand, for a functional test, first, the correct key is loaded from tpNVM into SCs using the mode M_0. Then, the initial state is loaded



(a) SC vs. RC **(b)** Blockage Circuitry

Figure 8.1: R-DFS Overall Architecture.

Table 8.2: Modes of Operation in Secure Cell (SC).

Test	SE	Mode	Description
0	0	M_0	The circuit is in functional mode. Actual keys from tpNVM applies to the Logic (Correct Functionality).
0	1	M_{1a}	The SCs hold their previous value. Based on the value of SE, RCs are in capture/shift mode.
1	0	M_{1b}	
1	1	M_2	The SCs become part of the scan chain. Actual/Dummy keys from SI for structural testing.

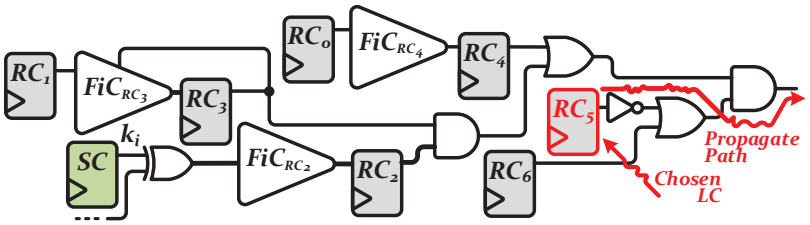
into the RCs in mode M_{1a}, with no change on the key value in SCs. Finally, the response is observed at the PO in mode M_0. To block unauthorized access to the scan chain (when a valid key is loaded), as illustrated in Fig. 8.1(b), the R-DFS architecture utilizes a SO-blockage circuitry. This module blocks/masks the SOs upon a switch from functional mode (mode M_0 that loads the actual key into SCs) to test mode (mode M_2 that supports the shift operation). Hence, after loading the key in mode M_0, SO will no longer be accessible, removing the possibility of SAT attack, and limiting the adversary's attack option to the far weaker and non-scalable unrolling-based or BMC based attacks.

8.1.2 Shift-and-Leak Attack on R-DFS

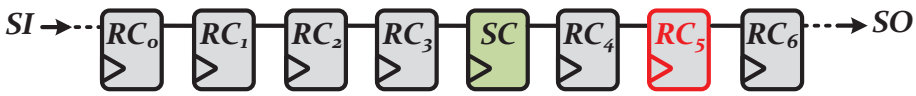
Although R-DFS breaks the SAT attack by blocking the SO, the introduction of shift-and-leak attack [57] shows that there is a valid key leakage possibility in R-DFS that allows the adversary to observe and extract the logic obfuscation key using PO. This attack exploits (1) the availability of the shift-in process through SI, and (2) the capability of reading out the PO through chip pin-outs in the functional mode. Considering Fig. 8.2 as an illustrative example, the steps of a shift-and-leak attack are as follows:

1. Identify leaky cells (LCs) that can leak info onto a PO.
2. Insert a stuck-at-fault at the chosen LC candidate.
3. Propagate the fault onto a PO (SCs set to unknown $X's$). If it fails to propagate, it rules out this LC, and repeats steps 1 and 2.
4. Power up the chip in mode M_0 to load the correct key into SCs.
5. Switch to mode M_{1a} (SCs hold value) and shift in d-bit reverse-shifted of the leak condition into the scan. The value of d is the scan distance between the targeted SC and the chosen LC.
6. Switch to mode M_2 (SCs are in the scan), and perform d-bit shift to have the leak condition in place and the key in chosen LC.
7. Clocklessly switch to mode M_0 and observe the PO, to leak the content of the LC, i.e., the target key bit.

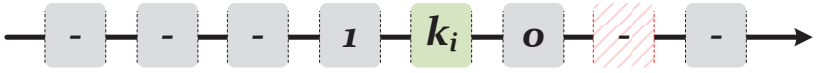
The authors noted that when the number of SCs increases, ATPG may fail to find a leak condition for the chosen LC. To address this challenge, by exploiting the conventional SAT attack [1], a pre-processing step was added to the shift-and-leak attack, in which the logic cone was treated as a locked combinational circuit considering RCs as the primary inputs and SCs as the key inputs. The pre-processing phase (which resembles the steps of the conventional SAT attack) is launched as follows:



(a) Determining the Leaky Cell (LC) in Circuit.



(b) The Scan Chain Configuration when SE=1. (d = 2)



(c) Shift-in the leaky condition (d-bit reverse-shifted) based on d=2.

(d) shift all FFs, including SCs and RCs, in Mode M_2, to put the k_i into the LC.

Figure 8.2: Example of shift-and-leak attack on R-DFS.

1. Extract the combinational fan-in cones of the PO.
2. Obtain a Discriminating Input (DIP) from the SAT tool on the extracted circuit.
3. Power on the IC in mode M_0 (SCs capture the actual key).
4. Switch to M_{1a} (SCs hold their values), and shift in the obtained DIP from the SAT tool to the RCs.
5. Clocklessly switch to mode M_0 and observe the PO (*eval* of the SAT attack). Then, go to step 2 until no more DIP found.

8.1.3 mR-DFS: Resisting Shift-and-Leak

As a countermeasure to the shift-and-leak attack, the work in [57] proposes a modified version of robust design-for-security architecture (denoted as mR-DFS in this paper) with a slight modification to the R-DFS. Since mode M_{1a} is used in the shift-and-leak attack

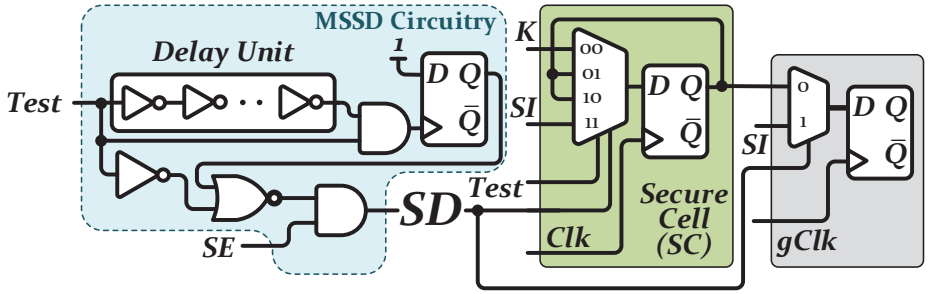


Figure 8.3: Mode Switch Shift Disable (MSSD) in mR-DFS.

to shift-in the known patterns (leak condition or DIP) to *RC*s, in mR-DFS, this mode is blocked. Also, to avoid any other form of leakage, after switching to mode M_0, it is not possible to re-enable any shift mode in the scan chain. To do that, as shown in Fig. 8.3, they build a shift disable (SD) signal, such that when $Test = 1$, SD follows SE. But, after the first capture of the actual key, i.e. when the $Test$ is low or when there is a positive transition on the $Test$, SD becomes *ALWAYS ZERO*, thereby blocking the shift operation. Hence, there is no longer a mode where *SC*s can be bypassed, retaining their values, while *RC*s can be loaded/shifted.

8.2 mR-DFS Architectural Drawbacks

Although mR-DFS addresses the leakage problem in R-DFS using shift disable (SD), the introduction of this shift disable (SD) signal in mR-DFS poses some new challenges for design and implementation flow, as well as test and debug process. These challenges are discussed next:

8.2.1 High Functional Test Time

Since there is no longer mode M_{1a} in mR-DFS architecture, the tester has to rely on mode M_2 to shift in and load the RCs. Also, since the shift is disabled when $Test$=0 or after the first positive transition on the $Test$, $Test$ must be high during power ON. Hence, the tester should use M_2 as the initial mode to shift in and load the initial state into the RCs. After

loading the initial state, the tester switches the mode to M_0 to load the actual key. Since it is not possible to re-enable the shift process after switching to mode M_0, the tester has to rely on the responses on PO. For the next test pattern, the tester needs to switch back to the mode M_2 to shift in and re-load the initial state corresponded to the next pattern. However, due to the blockage of the shift operation after switching to mode M_0 ($Test = 0$), the tester cannot use shift-in anymore for shifting in the next initial state. Hence, the tester has to reset the FF of MSSD circuitry to re-enable shift-in. This reset re-enables the SD to follow SE, thereby, the tester can shift in the next initial state. However this reset (sys_rst) will clear all storage elements, including SCs. So, it forces the tester to re-load the keys for the next test pattern. Hence, the actual key must be loaded again from tpNVM to accomplish the functional test, and this key reloading process (with each test pattern) significantly increases the functional test time. It should be noted that the initial state could be chosen to be used for a group of test patterns; however, choosing a specific initial state to be used for a group of patterns would increase the complexity of the functional test significantly. Besides, the designer cannot separate the reset pins for MSSD. Assuming that this reset pin is separated, the adversary can engage it to re-enable shift operation while the actual key is in place.

8.2.2 Necessity of Duplicating the SCs

In mR-DFS, after shifting in the initial state to the RCs using mode M_2, the tester switches to mode M_0 for only one cycle to load the actual key. However, during this one cycle, the RCs (loaded by initial state) would be updated. To avoid this problem, a clock gating circuitry has been introduced in mR-DFS to disable the clock for one clock cycle after switching from M_2 to M_0.

Without any consideration for this requirement in mR-DFS architecture, there are two possible methods to load the actual key from tpNVM in one clock cycle, however, both of them incurs considerable performance/area overhead: (1) engaging an ultra-wide memory that provides all bits of logic obfuscation key at once using only one read operation, (2)

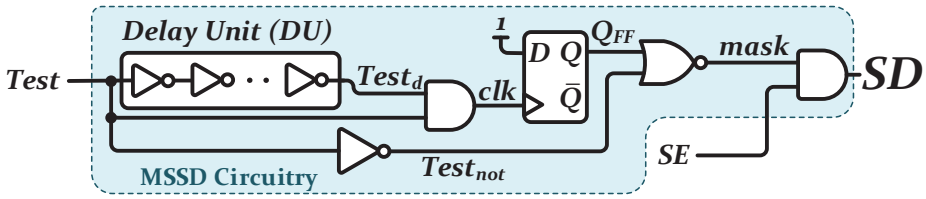
engaging temporary registers (FFs) to load the key into them at power ON, then connecting each SC to its corresponding temporary register to be loaded in one clock cycle.

Regarding the former solution, it is required to have direct wiring from tpNVM to each SC (per each key gate). Hence, the ultra-wide memory must have an extremely high fanout to provide this direct connection. This ultra-high fanout wiring increases the complexity of placement and routing (PnR) process, and it would significantly decrease the performance of the design, and due to optimization constraints in each design, using this scheme is almost impractical.

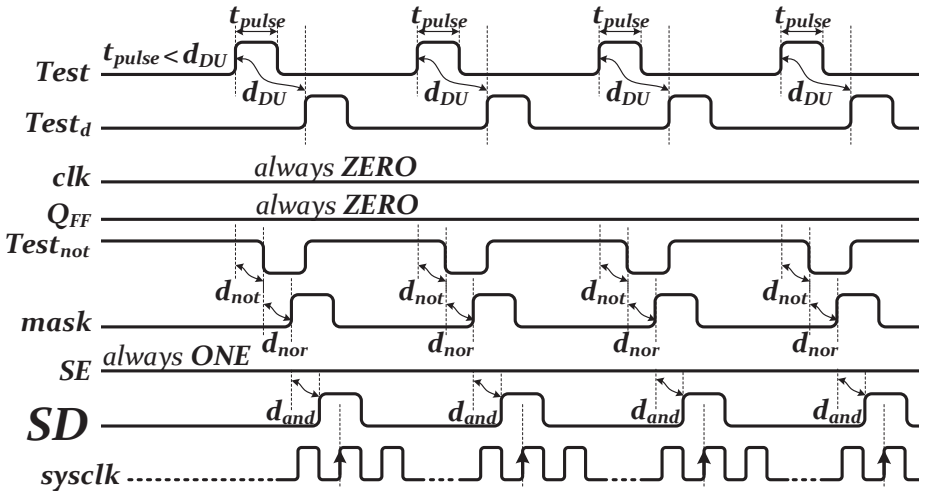
By choosing the latter method, the incurred overhead is more reasonable. However, the required reset (*sys_rst*) for loading the next initial state will clear whole registers in the chip. So, a key re-loading from tpNVM to the temporary register is required for each (group of) test pattern. It raises two big problems in mR-DFS: (1) It significantly increases the required time for functional test, and (2) Since key re-loading takes more than one clock cycle, it violates the assumption of mR-DFS, where clock-gating disables the clock signal only for one clock cycle to preserve the value of the RCs. So, after only one clock cycle, during the key re-loading, the RCs would be updated, and the functional test will fail.

8.2.3 Re-enabling Shift using Leaky Glitches

In mR-DFS, as shown in Fig. 8.3, the selector of $MUX21$ in RCs is controlled by SD, which becomes *ALWAYS ZERO* immediately after the first attempt of switching back from mode M_0 to M_2 (re-enabling shift process). Switching from mode M_0 to M_2 means that there is a positive transition on the $Test$ pin, and this positive transition allows the FF in MSSD circuitry to capture its input (*CONSTANT ONE*). However, there is still a possibility to switch back from mode M_0 to M_2 (positive transition on the $Test$ pin) while the FF does not capture its input (*CONSTANT ONE*) to make SD to be *ALWAYS ZERO*. To show that, we draw a timing diagram of the post-synthesis timing simulation of all internal wires of MSSD circuitry.



(a) MSSD (Blockage) Circuitry in mR-DFS.



(b) Glitches in Post-Synthesis Timing Simulation of MSSD Circuitry.

Figure 8.4: Re-enabling Shift after Actual Key Load.

As shown in Fig. 8.4(a), a delay unit (DU) has been used as a part of the fan-in-cone of the FF in MSSD circuitry, which is built using 10 inverters [57]. Assuming that the adversary is aware of timing information of the circuit, as shown in Fig. 8.4(b), she generates a stimuli for $Test$ pin in which the duration of high pulses is less than the delay of DU ($t_{pulse} < d_{DU}$). Hence, the inputs of the first AND gate, i.e. $Test$ and $Test_d$, have no overlap when both signals are high, and accordingly, DFF's clk would be $ALWAYS\ ZERO$. Since it is assumed that the DFF sets to 0 on reset, Q_{FF} would also be $ALWAYS\ ZERO$. So, the function of NOR gate is similar to NOT gate, whose input is $Test_{not}$. Consequently, $mask$ follows $Test$ with a delay of $d_{not} + d_{nor}$, and similarly, if we suppose that SE is $ALWAYS\ ONE$, SD follows $Test$ with a delay of $d_{not} + d_{nor} + d_{and}$. Since SD controls the shift operation in mR-DFS, using these potential glitches, the SD can re-enable the shift operation after mode M_0.

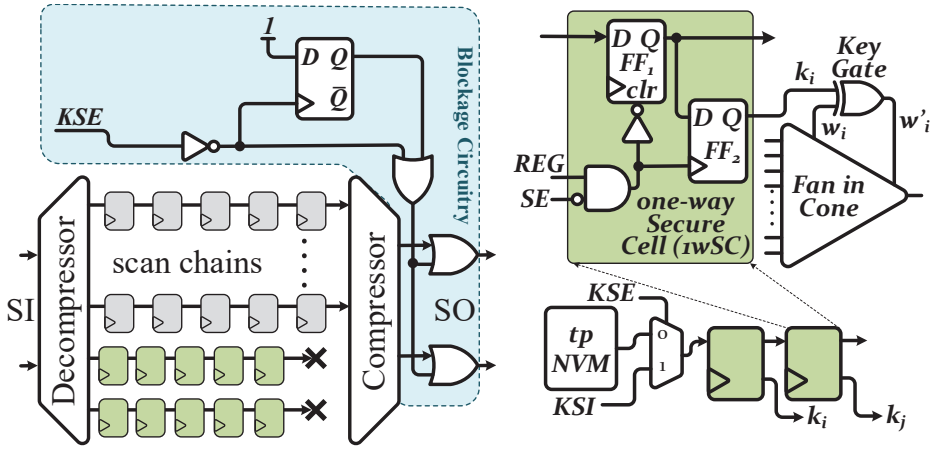
8.3 Proposed kt-DFS Architecture

When the logic obfuscation is in place, to introduce a secure and robust scan chain architecture, three requirements must be met:

1. There must be no possibility of key leakage during the test.
2. Both structural test and functional test must be carried out in a reasonable time (low test time overhead compared to the test time of the original design) without significant loss of coverage.
3. The complexity of test flow (structural and functional) and the overhead of secure scan chain architecture must be minimized.

In our proposed key-trapped DFS (kt-DFS), the scan chain(s) of the SCs are completely decoupled from the scan chain(s) of the RCs. In fact, **there is no reason for stitching the RC and SC cells in one chain, which has been the source of vulnerability in both R-DFS and mR-DFS**. As illustrated in Fig. 8.5(a), there is no common path between RCs and SCs in our proposed kt-DFS architecture. Also, considering that the SCs are only responsible to store the key value, none of the internal operations/computations overwrites the content of the SCs. So, when the scan chain is in place for the SCs, only the shift-in through SI is available for them to load the keys, and the SO is permanently blocked for scan chain(s) of the SCs.

To guarantee the security of SCs against any form of leakage, we re-design and introduce a new secure cell, called 1-way secure cell (1wSC). Fig. 8.5(b) depicts the details of 1wSC. Each 1wSC has two internal storage elements: a scan-connected storage (denoted as FF_1), and a trap storage (denoted as FF_2). The scan-connected storage could be used to shift values in and out of the 1wSC or into the trap storage. However, the value of the trap storage cannot be shifted out, and is only connected to its corresponded key gate. The transfer of key value from FF_1 to FF_2 takes place after setting $REG = 1$ and $SE = 0$, which is called *register mode*. Registration of the key into trap storage takes place on the



(a) Decoupling SCs from RCs **(b)** One-way Secure Cell (1wSC)

Figure 8.5: Proposed kt-DFS Overall Architecture.

rising edge of the clock input of the FF_2, which is a function of REG and SE. Also, this condition is used as the $RESET$ condition of all FF_1s to clear their values. Hence, $AND(Test, \overline{SE})$ is used as the clock source of FF_2s, and its toggled is used as the $RESET$ for FF_1s.

Also, the trap storage does not have a reset, and upon power-up randomly initialized to 0 or 1. So, upon transition of the key from scan-connected storage to the trap storage, since the storage is initialized randomly, the adversary cannot determine the previous value of the trap storage. This prevents the back-side imaging attack based on the captured heat map as described in [111] (e.g. when the activity is observed on heat map for a specific storage element, the adversary cannot determine if the transition is $\{0 \rightarrow 1\}$ or $\{1 \rightarrow 0\}$, and if NO activity is observed, the adversary cannot determine if the transition is $\{0 \rightarrow 1\}$ or $\{1 \rightarrow 1\}$).

In our proposed kt-DFS, the keys could be loaded into 1wSC from either tpNVM or scan-in (SI). Hence, the tester would be able to carry out the structural test by loading the desired key using SI. But, since the scan chain(s) of 1wSCs are decoupled in kt-DFS, two dedicated scan-enable and scan-in are used for the scan chain(s) of the SCs, called KSE

and KSI respectively.

The behavior of 1wSC is controlled using two pins, here called REG and SE. As captured in Table 8.3, based on these two pins, a 1wSC can be operated in three main modes:

1. **Functional Mode (M_0)**: $\{REG, SE\}$=$\{0,0\}$, and the RCs are in capture mode. Trap storage (FF_2s) must have the key. However, scan-connected storage (FF_1s) is able to capture a new key.

2. **Shift Mode ($M_{1,3}$)**: $\{REG, SE\}$=$\{X,1\}$, and the RCs are in shift mode. Scan-connected storage (FF_1s) is able to capture the key simultaneously, and there is no action on trap storage (FF_2s).

3. **Register Mode (M_2)**: $\{REG, SE\}$=$\{1,0\}$, and the pre-loaded key in scan-connected storage (FF_1s) would be written to trap storage (FF_2s), and scan-connected storage (FF_1s) will be cleared.

Similar to R-DFS and mR-DFS, a blockage circuitry is required to block the SO after the first attempt of key loading from the tpNVM. To support our proposed operational modes in the kt-DFS, a new blockage circuitry is designed. In kt-DFS, the SO must be blocked after loading the actual keys into FF_2s. When KSE is low, the FF_1 is fed using tpNVM. Hence, $\overline{KSE}$ is used to mask the SO. Note that the actual key would be loaded into FF_2 when $REG = 1$ and $SE = 0$ (register mode). However, before this condition, the tester has to load the actual key into FF_1s while the KSE is low. Hence, by only considering $KSE = 0$ as the blocking condition, we also cover the register-mode. Accordingly, the SO would be no longer available when KSE becomes low.

8.3.1 No Possibility of Key Leakage in kt-DFS

Considering that the leakage problem in R-DFS and mR-DFS is for unnecessary stitching of the RC and SC in the same scan chain, we fully decouple the SCs and RCs scan chains

Table 8.3: Modes of Operation in kt-DFS.

REG	SE	Mode	Description
0	0	M_0 (Functional Mode)	FF_2 must have the key*. FF_1 could capture the key*.
0	1	M_1 (Shift Mode)	FF_1 could capture the key*.
1	0	M_2 (Register Mode)	FF_2 are fed from FF_1. FF_1 will be reset to *ZERO*. Chain is erased.
1	1	M_3 (Shift Mode)	FF_1 could capture the key*.

* Based on KSE, actual/dummy key could be loaded from tpNVM/KSI

in kt-DFS, and the output of the scan chain(s) of SCs is permanently blocked. The values stored in the scan-connected storage (FF_1s) will be cleared with the transfer of the key to the trap storage (FF_2s). This guarantees that key values are trapped and no either regular or glitch-based shift can leak the key values to the SO.

8.3.2 Functional/Structural Test in kt-DFS

In kt-DFS architecture, the functional test and the structural test could be done without any significant limitation or any substantial overhead. For the structural test, since the SCs are equipped with new KSE and KSI pins, it could be accomplished using the following steps:

1. Set $KSE \rightarrow 1$ and mode to M_0. Shift in a dummy key via KSI.
2. Switch to mode M_2 to write the key into FF_2, and to clear FF_1.
3. Switch to mode M_1 to shift in the initial state into RCs.
4. Switch to mode M_0 for one clock cycle for capturing new state.
5. Switch again to mode M_1 to shift out the RCs to SO.

Unlike the structural test, the functional test requires the actual key. Hence, loading

the key from tpNVM followed by register mode will block the SO. Considering the blockage of the SO, the steps of the functional test is as follows:

1. Set $KSE \rightarrow 0$ and mode to M_0. Shift in the actual key from tpNVM. (When $KSE = 0$, the SO is blocked.)

2. Switch to mode M_2 to write the key into FF_2, and to clear FF_1. (Once $KSE = 0$ and mode is M_2, the SO will no longer available.)

3. Switch to mode M_1 to shift in the initial state into the RCs.

4. Switch to mode M_0 for one clock cycle for capturing new state, and clocklessly observe the PO.

It should be noted that similar to R-DFS and mR-DFS, the tester accomplishes the functional test through observing the PO with negligible loss of coverage.

8.3.3 Test Complexity and Scan Chain Overhead

Decoupling the scan chain(s) of SCs from that of RCs helps to facilitate the test flow for the tester compared to the test flow in mR-DFS. Despite mR-DFS with a mandatory *sys_rst* for each (group of) test pattern, no additional operation is required in kt-DFS for any form of the test. No *sys_rst* is required, and none of the operations is blocked after the first attempt of the actual key loading from tpNVM, and similar to R-DFS, only the SO is blocked to break the SAT attack. However, unlike R-DFS, it is fully secure against any form of leakage-based attacks, such as shift-and-leak.

The 1wSC in our proposed kt-DFS has two storage units and has a larger footprint compared to the SCs used in R-DFS and mR-DFS. However, the R-DFS and mR-DFS also need to transfer the key values from tpNVM to SCs. Using a very wide memory to derive thousands of keys is quite demanding in terms of area, and it imposes higher complexity during PnR. Hence, the R-DFS and mR-DFS also need to resort to a chain of temporary registers to transfer the keys. This means there is also a duplicated register per each SC in

Table 8.4: Specifications of the Benchmark Circuits in kt-DFS.

Circuit	s15850	s35932	s38584	b17	b18	b19
# of Inputs	77	35	38	37	37	24
# of Outputs	150	320	304	97	23	30
# of Gates	∼10K	∼16K	∼20K	∼28K	∼95K	∼190K
# of FFs	∼0.5K	∼1.7K	∼1.5K	∼1.5K	∼3.3K	6.6K
Area (mm^2)	0.025	0.031	0.041	0.055	0.238	0.539
Power (mW)	1.37	1.98	2.91	3.27	9.08	19.4

both R-DFS and mR-DFS. Furthermore, compared to $MUX41$ in both R-DFS and mR-DFS, only one AND gate and one NOT have been used in each 1wSC, which slightly improves the area overhead.

8.4 kt-DFS vs. other DFS Architectures

To analyze the security of the kt-DFS, and to provide better comparative results, with engaging the largest ISCAS-89 and ITC-99 benchmark circuits, as shown in Table 8.4, we re-produce the results for R-DFS+SLL [54], mR-DFS+RLL [57]. Also, for the proposed kt-DFS, we engage strong logic locking (SLL) [15] in all experiments to determine the location of key gates, and the number of key bits is set to 128/256. All the experiments have been accomplished on a Dell PowerEdge R620 equipped with Intel Xeon E5-2670 2.50GHz and 64GB of RAM, using Synopsys Design Compiler 2017.09, Tetramax 2017.09, and VCS 2017.12 tools along with the Synopsys generic 32nm library.

Table 8.5 demonstrates the area/power overhead mitigation of the proposed kt-DFS architecture compared to the state-of-the-art techniques. We achieved this improvement because we could remove multiplexers (MUX21 or MUX41) from our new proposed secure cell structure. Since we fully decoupled the SCs from SFFs, it allows us to remove the MUXes that were required to control the inputs of SCs when SCs and SFFs are in the same scan chain. It is worth mentioning that to avoid facing drastic area/delay overhead in the existing approaches, we use temporary registers used for temporary loading the key from

Table 8.5: PPA Overhead Comparison between Different DFS Architectures.

Circuit	R-DFS+SLL [56]		mR-DFS+RLL [57]		kt-DFS+SLL	
	Area (%)	Power (%)	Area (%)	Power (%)	Area (%)	Power (%)
s15850	17.85%	24.76%	15.02%	21.77%	12.68%	16.27%
s35932	15.99%	19.39%	11.54%	16.54%	8.52%	10.28%
s38584	15.52%	18.95%	12.37%	15.28%	8.76%	9.92%
b17	7.31%	10.21%	6.33%	9.17%	2.76%	6.24%
b18	3.89%	6.71%	2.85%	5.51%	0.97%	3.38%
b19	2.53%	4.55%	1.28%	4.06%	0.51%	1.75%

the tpNVM. Without this mechanism, we cannot have the keys in existing techniques at once as claimed.

Table 8.6 reflects the impact of a varying number of scan chains as well as the key size when the proposed kt-DFS is in place. Since the chains of SCs are fully decoupled, varying the number of scan chains has a minimal impact on the overhead of the proposed architecture. For chains of SCs, we assume that when key size is 128, it is one chain of SCs, and when the key size is 256, it is two chains of SCs. In terms of power overhead, since dividing the scan chain into multiple ones always incur more power overhead, the impact ratio is a bit higher. Also, as demonstrated in Table 8.6, when we increase the key size (128 $\rightarrow$ 256), since the main part of the overhead is the extra FFs added for key storage (This is the same for all existing techniques), for larger circuits, due to decreasing the ratio of SCs to SFFs, the overhead is less, and in general, it is small enough as expected.

Unlike state-of-the-art techniques, in the proposed kt-DFS, we fully decoupled SCs from SFFs. However, this separation will affect the placement and routing of chains to get the most benefit of optimization steps during DFT synthesis (based on the locality of storage cells). Table 8.7 shows the area overhead after placement-and-routing (post-PnR) of our proposed kt-DFS when SCs are decoupled from SFFs versus when SCs are stitched with SFFs. As shown, when the number of scan chains for SFFs is increased, due to breaking SFFs into sub-domains (sub-locality), post-PnR area overhead is much closer to the stitched

Table 8.6: kt-DFS PPA Overhead with Different {Key Sizes, Number of Scan Chains}*.

	Area Overhead							
Circuit	{128, 1}*	{128, 2}	{128, 4}	{128, 8}	{256, 1}	{256, 2}	{256, 4}	{256, 8}
s15850	12.68%	12.71%	12.75%	12.95%	18.82%	18.89%	18.98%	18.08%
s35932	8.52%	8.65%	8.72%	8.95%	11.22%	11.26%	11.32%	11.42%
s38584	8.76%	8.81%	8.86%	8.97%	10.79%	10.86%	10.91%	11.03%
b17	2.76%	2.78%	2.81%	2.83%	5.92%	5.94%	5.97%	5.99%
b18	0.97%	0.98%	0.98%	0.99%	1.43%	1.46%	1.46%	1.46%
b19	0.51%	0.51%	0.51%	0.52%	1.01%	1.02%	1.02%	1.02%
	Power Overhead							
Circuit	{128, 1}*	{128, 2}	{128, 4}	{128, 8}	{256, 1}	{256, 2}	{256, 4}	{256, 8}
s15850	16.27%	17.36%	18.24%	19.52%	23.54%	23.68%	25.15%	26.77%
s35932	10.28%	10.43%	11.63%	12.71%	13.28%	13.13%	14.26%	15.82%
s38584	9.92%	9.86%	10.71%	12.33%	12.88%	12.91%	14.09%	15.67%
b17	6.24%	6.26%	7.15%	7.75%	9.81%	10.23%	12.08%	13.25%
b18	3.38%	3.36%	3.74%	3.91%	5.16%	5.83%	6.17%	6.44%
b19	1.71%	1.74%	1.95%	1.97%	2.09%	2.01%	2.34%	2.69%

version. On the other hand, when the number of scan chains is also few (like 1), the impact of decoupling is also minimal. Hence, to summarize, decoupling SCs from SFFs has minimal impact on post-PnR overhead.

Table 8.8 represents the structural test coverage and the leakage of R-DFS, mR-DFS, and our proposed kt-DFS when the number of scan chains is set to be 1. For almost all techniques, the manufacturing test works perfectly fine, and the test coverage is roughly the same for all cases. However, for R-DFS and mR-DFs with stitching architecture, shift-and-leak and glitch-based shift-and-leak attack can recover the key of the locked circuit. However, our proposed architecture, with decoupled structure, helps keeping the locked circuit secure against leakage possibilities.

Since access to the scan chains is restricted in these techniques, the SAT attack cannot be deployed. This does not prevent an attacker from deploying the unrolling or bounded-model-checking (BMC) [60] attack that only needs PI/PO. However, this group of attacks runs into scalability issues as they rely on two sub-routines which are in PSPACE and NP [57]. Even the accelerated version of this attack (described in [62]) fails to terminate for

Table 8.7: SCs and SFFs Decoupling vs Stitching in kt-DFS.

Circuit	Decoupling SCs and SFFs				Stitching SCs with SFFs			
	{128, 1}*	{128, 2}	{128, 4}	{128, 8}	{128, 1}	{128, 2}	{128, 4}	{128, 8}
s15850	13.53%	13.72%	13.81%	13.97%	12.86%	13.22%	13.28%	13.72%
s35932	9.13%	9.22%	9.41%	9.55%	8.56%	8.65%	9.12%	9.32%
s38584	9.07%	9.54%	9.38%	9.66%	8.67%	9.01%	9.18%	9.23%
b17	3.01%	2.96%	3.22%	3.18%	2.76%	2.64%	2.89%	3.09%
b18	1.05%	1.06%	1.06%	1.06%	0.94%	0.97%	0.98%	1.01%
b19	0.68%	0.67%	0.69%	0.69%	0.65%	0.66%	0.66%	0.66%

Table 8.8: Test Coverage and Key Leakage Comparison between DFS Architectures.

Circuit	Original	R-DFS [56]		mR-DFS [57]		Proposed kt-DFS	
	Test (%)	Test (%)	Key Leak (#)	Test (%)	Key Leak (#)	Test (%)	Key Leak (#)
s15850	100%	100%	127	100%	127	100%	0
s35932	100%	100%	128	100%	128	100%	0
s38584	100%	100%	128	100%	128	100%	0
b17	99.91%	99.72%	127	99.69%	127	99.67%	0
b18	99.77%	99.78%	126	99.73%	126	99.73%	0
b19	99.81%	99.78%	127	99.78%	127	99.78%	0

even small designs. Besides, new techniques such as DFSSD [50] shows how low overhead techniques, like deep faults and shallow state duality, could be used to break the state-of-the-art sequential SAT attacks. To show the lack of scalability of the BMC or unrolling-based SAT attacks, we apply KC2 [62] on kt-DFS+SLL locked circuits, and the results are reflected in Table 8.9. As shown, this attack could only work for the two smallest circuits with the key size equal to 100, and for all other cases, it fails to reach the result before the time-out (10^5 Seconds).

Table 8.9: KC2 Execution Time on kt-DFS+SSL.

Circuit	Key Size = 100		Key Size = 200	
	Iterations	Execution Time (s)	Iterations	Execution Time (s)
s15850	31	2666	*timeout*	*timeout*
s35932	184	15328	*timeout*	*timeout*
s38584	*timeout*	*timeout*	*timeout*	*timeout*
b17	*timeout*	*timeout*	*timeout*	*timeout*
b18	*timeout*	*timeout*	*timeout*	*timeout*
b19	*timeout*	*timeout*	*timeout*	*timeout*

- timeout: 10^5 Seconds $\approx$ one day (Stop the attack when time reaches timeout)

Chapter 9: Discussion and Opportunities

In this thesis, we introduced and evaluated different logic locking countermeasures, including non-XOR-based (LUT-based and MUX-based) techniques, MUX-based and memory-based scan logic locking, and scan blockage technique. We now enumerate some substantial concise outcomes and conclusions that may help us drawing the next steps as the open research studies:

1. LUT-based logic locking could provide high resiliency when the key attributes of locking are determined appropriately.

2. The key attributes of LUT-based logic locking could be (1) size of the LUT, (2) number of LUTs inserted into the circuit, and (3) the replacement strategy.

3. LUT-Lock and other state-of-the-art LUT-based locking techniques suffer from huge overhead, which currently makes this direction unappealing.

4. The replacement strategy could considerably affect the size of the LUTs and the numbers of LUTs required to be inserted to guarantee the SAT resiliency. However, none of the existing approaches could help us to reduce the overhead of LUT to a reasonable threshold.

5. Routing-based locking solutions could provide the desired resiliency at a much lower overhead ratio compared to LUT-based locking.

6. Routing-based locking could be implemented at different levels of abstraction with much more flexibility compared to LUT-based locking solution.

7. Pure routing-based techniques, such as Full-Lock and Cross-lock are vulnerable against satisfiability-based routing optimization techniques, such as numerical bound problems, eventually allowed us to break them using canonical pruning & SAT attack.

8. To still get the benefit of routing-based locking solution, and to avoid the efficiency of satisfiability-based routing optimization techniques on them, intertwining logic and routing modules is the best solution.

9. Embedding actual timing paths as the logic within the routing blocks will also help to minimize the possibility of applying attacks like removal and bypass attacks.

10. The routing-based (MUX-based) locking technique could be engaged for scan, sequential, and FSM locking at much lower overhead.

11. The concept of in-memory computation could be engaged as a means of logic locking, helping the designers to add explicit black boxes within the design, making the de-obfuscation flow much harder.

12. Compared to scan-based logic locking, scan blockage techniques could provide robustness against the SAT and its derivatives at much lower overhead.

13. The scan blockage techniques required to be implemented to meet two important criteria: (a) providing resilience against the existing de-obfuscation attacks with no possibility of leakage, and (b) having no compromising, including test complexity, high test time, or reduced test coverage.

With this in mind, some open challenges in these directions are as follows:

1. More investigation could be inevitable in case of improving LUT-based logic locking towards (a) to design and build a customized LUT structure, (b) to have more exploration on the combination of MUX-based and LUT-based locking, (c) to find better LUT placement strategies required less LUT insertion, and to move from LUT-per-gate replacement which incurs significant overhead to LUT-per-cone replacement.

2. The introduction of newer attacks on logic locking, especially recent machine-learning-based attacks, such as NNgSAT [112], shows complex modules like routing modules might need more improvement. Hence, the security evaluation of InterLock on newer attacks is required.

3. A very recent study has been introduced the new concept *oracle dishonesty* in DisORC [59], which could be considered as a scan blockage technique that is featured by the capability of turning into a dishonest reference whenever a potential attack is detected. By turning the oracle into a dishonest one, the adversary will lose many of the basic assumptions known as the basics of different threat models, which makes the de-obfuscation process much harder. However, the implementation of the dishonesty with an advanced architecture like DisORC could be an open direction for further studies, which provides enhanced security at lower overhead and less test compromising.

4. Having direct access and reading the electrical signals on a chip is a big challenge against any security countermeasures, which recently received a lot of attention [111]. One approach to combat these threats is the exploit of randomness, such as randomizing initialization and using register with random initial values within the test/scan components. Hence, the design and implementation of test/scan components that support such randomization to combat physical accesses require more evaluation. Similarly, hardware Trojan insertion could undermine logic obfuscation techniques. Few recent studies have investigated hardware Trojan attacks on logic obfuscation [113]. Designing a new test/scan structure that could detect unauthorized test access (such as test access by activated hardware Trojan) is mandatory in this case. This is an open research area that requires more attention to combat this breed of threats.

Chapter 10: Conclusion

Due to the high cost of building IC manufacturing fab, with huge recurring maintenance costs, many manufacturing and fabrication are pushed offshore. However, due to lack of reliable monitoring on outsourced stages, results in emerging security vulnerabilities, including but not limited to reverse-engineering, hardware Trojan insertion, counterfeiting, IP piracy, and over-production.

Logic obfuscation, *a.k.a.* logic locking, is a proactive design-for-trust (DfTr) technique that could combat all the aforementioned threats. Logic locking could add ambiguity to the circuit by adding limited post-fabrication programmability into the circuit. This programmability could be achieved by adding/inserting some specific gates, known as key gates. The key gates will be fed using programming value, referred to as the key, and the key would be initiated via a trusted party, and in most cases, it would be stored within a tamper-proof non-volatile memory (tpNVM).

Unlike most logic locking solutions that rely on XOR gates as the key gate, in this thesis we first tried to open a new direction in this topic with the investigation of non-XOR-based logic locking techniques. We introduced one LUT-based logic locking, called LUT-lock, and to the best of our knowledge, it is the first SAT resilient LUT-based logic solution. We also comprehensively evaluated LUT-based locking solutions in terms of LUT size, number of LUTs, and replacement strategy, and our experimental results demonstrate a very high overhead rate in this group of techniques.

We also introduced Full-Lock, as a new routing-based locking solution, which builds SAT-hard instances at lower overhead compared to LUT-based locking solutions. However, our further investigation revealed that the complexity of the pure routing-based locking could be reduced significantly using the satisfiability-based routing optimization techniques. Then, we introduced a more advanced routing-based locking technique, called InterLock, in

which by embedding logic within the routing block, we show none of the satisfiability-based routing optimization techniques could provide enough size/complexity reduction. We also evaluated routing-based locking in different technologies and different levels of abstractions to show the efficiency of this new logic locking category.

After the introduction of robust and efficient MUX-based (routing-based) logic locking technique, we exploited such structures for scan-based, sequential, or FSM logic locking. As a new study, called SCRAMBLE, we showed how routing modules with much smaller sizes could provide robustness when they target the scan chain structure within an IC. We also evaluated the possibility of using simple in-memory computation as a means of logic locking. Using in-memory computation, part(s) of the logic could be modeled and re-mapped using a one-cycle memory, and the content of the memory will be initiated after the fabrication as the secret of the logic locking. We showed that with a very small size of memory, the locked circuit could be resilient against a wide range of the existing threats, including the SAT and its derivatives.

Furthermore, by revealing the architectural drawbacks of the existing scan blockage techniques, we introduce a key-trapped design-for-security (kt-DFS) architecture. In kt-DFS, we propose a new secure scan chain cell that is robust against any form of logical leakage once the key is initiated into the circuit. Also, we introduce a simple and lightweight scan blockage circuitry that can block and limit access to the scan chain architecture while any unauthorized access is recognized. We showed that scan blockage techniques could be known as a requirement for the circuits while the logic locking is in place to guarantee that the locked circuit provides acceptable robustness against the existing de-obfuscation attacks.

Appendix A: List of Publications

[**ISVLSI'18**] **Hadi Mardani Kamali**, Kimia Zamiri Azar, Kris Gaj, Houman Homayoun, Avesta Sasan, *"LUT-Lock: A Novel LUT-based Logic Obfuscation for FPGA-bitstream and ASIC-hardware Protection,"* 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 405-410, 2018.

[**GLSVLSI'18**] Shervin Roshanisefat, **Hadi Mardani Kamali**, Avesta Sasan, *"SRCLock: SAT-resistant cyclic logic locking for protecting the hardware,"* Proceedings of the 2018 on Great Lakes Symposium on VLSI (GLSVLSI), pp. 153-158, 2018.

[**GLSVLSI'18**] **Hadi Mardani Kamali**, Avesta Sasan, *"MUCH-SWIFT: A High-Throughput Multi-Core HW/SW Co-design K-means Clustering Architecture,"* Proceedings of the 2018 on Great Lakes Symposium on VLSI (GLSVLSI), pp. 459-462, 2018.

[**TCHES'19**] Kimia Zamiri Azar, **Hadi Mardani Kamali**, Houman Homayoun, Avesta 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, vol. 2019, no. 1, pp. 97-122, 2019.

[**RAID'19**] Kimia Zamiri Azar, Farnoud Farahmand, **Hadi Mardani Kamali**, Shervin Roshanisefat, Houman Homayoun, William Diehl, Kris Gaj, Avesta Sasan, *"COMA: Communication and Obfuscation Management Architecture," International Symposium on Research in Attacks, Intrusions and Defenses, pp. 181-195, 2019.*

[**GLSVLSI'19**] Kimia Zamiri Azar, **Hadi Mardani Kamali**, Houman Homayoun, Avesta Sasan, *"Threats on logic locking: A decade later,"* Proceedings of the 2019 on Great Lakes Symposium on VLSI, pp. 471-476, 2019.

[**DAC'19**] **Hadi Mardani Kamali**, Kimia Zamiri Azar, Houman Homayoun, Avesta Sasan, *"Full-lock: Hard distributions of sat instances for obfuscating circuits using fully configurable logic and routing blocks,"* Proceedings of the 56th Annual Design Automation Conference, pp. 1-6, 2019.

[**ICCAD'19**] Gaurav Kolhe, **Hadi Mardani Kamali**, Miklesh Naicker, Tyler David Sheaves, Hamid Mahmoodi, PD Sai Manoj, Houman Homayoun, Setareh Rafatirad, Avesta Sasan, *"Security and complexity analysis of lut-based obfuscation: From blueprint to reality,"* 2019 IEEE/ACM International Conference On Computer Aided Design, pp. 1-8, 2019.

[**VTS'20**] Shervin Roshanisefat, **Hadi Mardani Kamali**, Kimia Zamiri Azar, Sai Manoj Pudukotai Dinakarrao, Naghmeh Karimi, Houman Homayoun, Avesta Sasan, *"Dfssd: Deep faults and shallow state duality, a provably strong obfuscation solution for circuits with restricted access to scan chain,"* 2020 IEEE 38th VLSI Test Symposium (VTS), pp. 1-6, 2020.

[**DCAS'20**] **Hadi Mardani Kamali**, Kimia Zamiri Azar, Shervin Roshanisefat, Ashkan Vakil, Houman Homayoun, Avesta Sasan, *"Extru: A lightweight, fast, and secure expirable trust for the internet of things,"* 2020 IEEE 14th Dallas Circuits and Systems Conference (DCAS), pp. 1-8, 2020.

[**GLSVLSI'20**] **Hadi Mardani Kamali**, Kimia Zamiri Azar, Houman Homayoun, Avesta Sasan, *"On designing secure and robust scan chain for protecting obfuscated logic,"* Proceedings of the 2020 on Great Lakes Symposium on VLSI, pp. 1-7, 2020.

[**TVLSI'20**] Shervin Roshanisefat, **Hadi Mardani Kamali**, Houman Homayoun, Avesta Sasan, *"Sat-hard cyclic logic obfuscation for protecting the ip in the manufacturing supply chain,"* IEEE Transactions on Very Large Scale Integration (VLSI) Systems, pp. 954-967, 2020.

[**ISVLSI'20**] **Hadi Mardani Kamali**, Kimia Zamiri Azar, Houman Homayoun, Avesta Sasan, *"SCRAMBLE: The state, connectivity and routing augmentation model for building logic encryption,"* 2020 IEEE Computer Society Annual Symposium on VLSI, pp. 153-159.

[**ICCAD'20n**] Kimia Zamiri Azar, **Hadi Mardani Kamali**, Houman Homayoun, Avesta Sasan, *"NNgSAT: Neural Network guided SAT Attack on Logic Locked Complex Structures,"* 2020 IEEE/ACM International Conference On Computer Aided Design, pp. 1-9, 2020.

[**ICCAD'20i**] **Hadi Mardani Kamali**, Kimia Zamiri Azar, Houman Homayoun, Avesta

Sasan, *"InterLock: An Intercorrelated Logic and Routing Locking,"* 2020 IEEE/ACM International Conference On Computer Aided Design, pp. 1-9, 2020.

[**TVLSI'21**] Kimia Zamiri Azar, **Hadi Mardani Kamali**, Shervin Roshanisefat, Houman Homayoun, Christos P Sotiriou, Avesta Sasan, *"Data Flow Obfuscation: A New Paradigm for Obfuscating Circuits,"* IEEE Transactions on Very Large Scale Integration (VLSI) Systems, pp. 1-14, 2021.

[**ISQED'21**] **Hadi Mardani Kamali**, Kimia Zamiri Azar, Houman Homayoun, Avesta Sasan, *"ChaoLock: Yet Another SAT-hard Logic Locking using Chaos Computing,"* IEEE International Symposium on Quality Electronic Design (ISQED), pp. 1-7, 2021.

[**Frontier'21**] Zhiqian Chen, Lei Zhang, Gaurav Kolhe, **Hadi Mardani Kamali**, Setareh Rafatirad, Sai Manoj Pudukotai Dinakarrao, Houman Homayoun, Chang-Tien Lu, Liang Zhao, *"Deep Graph Learning for Circuit Deobfuscation,"* Frontiers in Big Data, pp. 1-12, 2021.

[**IEEE ACCESS'21**] Kimia Zamiri Azar, **Hadi Mardani Kamali**, Houman Homayoun, Avesta Sasan, *"From Cryptography to Logic Locking: A Survey on The Architecture Evolution of Secure Scan Chains,"* IEEE Access, pp. 1-18, 2021.

[**GLSVLSI'21**] Shervin Roshanisefat, **Hadi Mardani Kamali**, Houman Homayoun, Avesta Sasan, *"RANE: An Open-Source Formal De-obfuscation Attack for Reverse Engineering of Logic Encrypted Circuits,"* Proceedings of the 2021 on Great Lakes Symposium on VLSI, pp. 1-8, 2021.

Bibliography

Bibliography

[1] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the Security of Logic Encryption Algorithms," in *Int'l Symp. on Hardware Oriented Security and Trust (HOST)*, 2015, pp. 137–143.

[2] M. El Massad, S. Garg, M. Tripunitara, "Integrated Circuit (IC) Decamouflaging: Reverse Engineering Camouflaged ICs within Minutes," in *NDSS*, 2015, pp. 1–14.

[3] 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 International Symposium on On-Line Testing And Robust System Design (IOLTS)*, 2018, pp. 275–280.

[4] Y. Xie and A. Srivastava, "Delay Locking: Security Enhancement of Logic Locking against IC Counterfeiting," in *Proceedings of Design Automation Conference (DAC)*, 2017, pp. 1–9.

[5] K. Shamsi, M. Li, D. Z. Pan, Y. Jin, "Cross-lock: Dense layout-level interconnect locking using cross-bar architectures," in *Proceedings of the on Great Lakes Symposium on VLSI (GLSVLSI)*, 2018, pp. 147–152.

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

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

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

[9] M. Tehranipoor, C. Wang, *Introduction to hardware security and trust*. Springer Science & Business Media, 2011.

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

[11] A. B. Kahng, J. Lach, W. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Constraint-based Watermarking Techniques for Design IP protection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 10, pp. 1236–1252, 2001.

[12] Y. Alkabani and F. Koushanfar, "Active Hardware Metering for Intellectual Property Protection and Security," in *USENIX Security Symposium*, 2007, pp. 291–306.

[13] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security Analysis of Integrated Circuit Camouflaging," in *Proceedings of the ACM SIGSAC conference on Computer & communications security*, 2013, pp. 709–720.

[14] J. Roy F. Koushanfar, and I. L. Markov, "EPIC: Ending Piracy of Integrated Circuits," in *Design, Automation & Test in Europe Conf. (DATE)*, 2008, pp. 1069–1074.

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

[16] P. Tuyls, G. Schrijen, B. Škorić, J. Van Geloven, N. Verhaegh, and R. Wolters, "Read-proof Hardware from Protective Coatings," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2006, pp. 369–383.

[17] 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.

[18] H. M. Kamali, K. Z. Azar, K. Gaj, H. Homayoun, A. Sasan, "Full-lock: Hard Distributions of SAT Instances for Obfuscating Circuits using Fully Configurable Logic and Routing Blocks," in *56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[19] H. M. Kamali, K. Z. Azar, K. Gaj, H. Homayoun, and A. Sasan, "Interlock: An Intercorrelated Logic and Routing Locking," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.

[20] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "SCRAMBLE: The State, Connectivity and Routing Augmentation Model for Building Logic Encryption," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2020, pp. 153–159.

[21] Actel Corporation, "Design Security in Nonvolatile Flash and Antifuse FPGAs - Security Backgrounder," *Technical Report on Quick Logic FPGAs*, 2002.

[22] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, Avesta, "Threats on Logic Locking: A Decade Later," in *Proceedings of the Great Lakes Symposium on VLSI*, 2019, pp. 471–476.

[23] 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.

[24] 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.

[25] Y. Xie and A. Srivastava, "Mitigating SAT Attack on Logic Locking," in *International Conference on Cryptographic Hardware and Embedded Systems (CHES)*, 2016, pp. 127–146.

[26] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu, "SARLock: SAT Attack Resistant Logic Locking," in *Hardware Oriented Security and Trust (HOST) Symposium*, 2016, pp. 236–241.

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

[28] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, Y. Jin, "Cyclic Obfuscation for Creating SAT-unresolvable Circuits," in *Proceedings of the on Great Lakes Symposium on VLSI (GLSVLSI)*, 2017, pp. 173–178.

[29] S. Roshanisefat, H. M. Kamali, A. Sasan, "SRCLock: SAT-resistant cyclic logic locking for protecting the hardware," in *Proceedings of the on Great Lakes Symposium on VLSI (GLSVLSI)*, 2018, pp. 153–158.

[30] A. Rezaei, Y. Shen, S. Kong, J. Gu, and H. Zhou, "Cyclic Locking and Memristor-based Obfuscation against CycSAT and inside Foundry Attacks," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2018, pp. 85–90.

[31] A. Rezaei, Y. Li, Y. Shen, S. Kong, and H. Zhou, "CycSAT-unresolvable Cyclic Logic Encryption using Unreachable States," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2019, pp. 358–363.

[32] S. Roshanisefat, H. M. Kamali, H. Homayoun, and A. Sasan, "SAT-Hard Cyclic Logic Obfuscation for Protecting the IP in the Manufacturing Supply Chain," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 4, pp. 954–967, 2020.

[33] K. Z. Azar, H. M. Kamali, S. Roshanisefat, H. Homayoun, C. P. Sotiriou, and A. Sasan, "Data Flow Obfuscation: A New Paradigm for Obfuscating Circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, no. 01, pp. 1–14, 2021.

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

[35] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal Attacks on Logic Locking and Camouflaging Techniques," *IEEE Transactions on Emerging Topics in Computing*, 2017.

[36] X. Xu, B. Shakya, M. Tehranipoor, D. Forte, "Novel Bypass Attack and BDD-based Tradeoff Analysis against all Known Logic Locking Attacks," in *CHES*, 2017, pp. 189–210.

[37] D. Sirone and P. Subramanayan, "Functional Analysis Attacks on Logic Locking," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2514–2527, 2020.

[38] 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.

[39] Y. Shen and H. Zhou, "Double-Dip: Re-evaluating Security of Logic Encryption Algorithms," in *Proceedings of the on Great Lakes Symposium on VLSI (GLSVLSI)*, 2017, pp. 179–184.

[40] H. Chiang, Y. Chen, D. Ji, X. Yang, C. Lin, C. Wang, "LOOPLock: Logic Optimization-Based Cyclic Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2178–2191, 2019.

[41] H. Zhou, R. Jiang, and S. Kong, "CycSAT: SAT-based Attack on Cyclic Logic Encryptions," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 49–56.

[42] Y. Shen, Y. Li, A. Rezaei, S. Kong, D. Dlott, and H. Zhou, "BeSAT: Behavioral SAT-based Attack on Cyclic Logic Encryption," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2019, pp. 657–662.

[43] K. Shamsi, D. Z. Pan, and Y. Jin, "IcySAT: Improved SAT-based Attacks on Cyclic Locked Circuits," in *Int'l Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–7.

[44] 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)*, pp. 97–122, 2019.

[45] A. Chakraborty, Y. Liu, and A. Srivastava, "TimingSAT: timing profile embedded SAT attack," in *ICCAD*, 2018, p. 6.

[46] 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.

[47] H. M. Kamali, K. Z. Azar, S. Roshanisefat, A. Vakil, A. Sasan, "ExTru: A Lightweight, Fast, and Secure Expirable Trust for the Internet of Things," *IEEE 14th Dallas Circuits and Systems Conference (DCAS)*, 2020.

[48] R. Karmakar, S. Chatopadhyay, and R. Kapur, "Encrypt Flip-Flop: A Novel Logic Encryption Technique for Sequential Circuits," *arXiv preprint arXiv:1801.04961*, 2018.

[49] S. Potluri, A. Aysu, A. Kumar, "Seql: Secure Scan-locking for IP Protection," in *International Symposium on Quality Electronic Design (ISQED)*, 2020, pp. 7–13.

[50] S. Roshanisefat, H. M. Kamali, K. Z. Azar, S. M. P. Dinakarrao, N. Karimi, H. Homayoun, and A. Sasan, "Dfssd: Deep Faults and Shallow State Duality, a Provably Strong Obfuscation Solution for Circuits with Restricted Access to Scan Chain," in *IEEE VLSI Test Symposium (VTS)*, 2020, pp. 1–6.

[51] D. Zhang, M. He, X. Wang, and M. Tehranipoor, "Dynamically Obfuscated Scan for Protecting IPs against Scan-based Attacks throughout Supply Chain," in *VLSI Test Symposium (VTS)*, 2017, pp. 1–6.

[52] R. Karmakar, H. Kumar, and S. Chattopadhyay, "Efficient Key-gate Placement And Dynamic Scan Obfuscation Towards Robust Logic Encryption," *IEEE Transactions on Emerging Topics in Computing*, 2019.

[53] R. Karmakar, S. Chattopadhyay, and R. Kapur, Rohit, "A Scan Obfuscation guided Design-for-Security Approach for Sequential Circuits," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2019.

[54] X. Wang *et al.*, "Secure Scan and Test using Obfuscation throughout Supply Chain," *IEEE Trans. on CAD*, vol. 37, no. 9, pp. 1867–1880, 2017.

[55] W. Wang *et al.*, "A Secure DFT Architecture Protecting Crypto Chips Against Scan-Based Attacks," *IEEE Access*, vol. 7, pp. 22 206–22 213, 2019.

[56] U. Guin, Z. Zhou, and A. Singh, "Robust Design-for-Security Architecture for enabling Trust in IC Manufacturing and Test," *IEEE Transactions on VLSI*, vol. 26, no. 5, pp. 818–830, 2018.

[57] N. Limaye, A. Sengupta, M. Nabeel, and O. Sinanoglu, "Is Robust Design-for-Security Robust Enough? Attack on Locked Circuits with Restricted Scan Chain Access," *Int'l Conference on CAD (ICCAD)*, pp. 1–8, 2019.

[58] H. M. Kamali, K. Z. Azar, H. Homayoun, A. Sasan, "On Designing Secure and Robust Scan Chain for Protecting Obfuscated Logic," in *Great Lakes Symposium on VLSI (GLSVLSI)*, 2020, pp. 1–6.

[59] N. Limaye, E. Kalligeros, N. Karousos, I. G. Karybali, and O. Sinanoglu, "Thwarting All Logic Locking Attacks: Dishonest Oracle with Truly Random Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[60] M. El Massad, S. Garg, and M. Tripunitara, "Reverse Engineering Camouflaged Sequential Circuits without Scan Access," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2017, pp. 33–40.

[61] L. Alrahis, M. Yasin, N. Limaye, H. Saleh, B. Mohammad, M. Alqutayri, and O. Sinanoglu, "Scansat: Unlocking Static and Dynamic Scan Obfuscation," *IEEE Transactions on Emerging Topics in Computing*, 2019.

[62] K. Shamsi, M. Li, D. Z. Pan, and Y. Jinr, "KC2: Key-condition Crunching for Sequential Circuit Deobfuscation," in *Design, Automation & Test in Europe Conference (DATE)*, 2019, pp. 534–539.

[63] N. Limaye and O. Sinanoglu, "DynUnlock: Unlocking Scan Chains Obfuscated using Dynamic Keys," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, pp. 270–273.

[64] T. Winograd, H. Salmani, H. Mahmoodi, K. Gaj, and H. Homayoun, "Hybrid STT-CMOS Designs for Reverse-Engineering Prevention," in *Annual Design Automation Conference (DAC)*, 2016, pp. 1–6.

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

[66] R. S. Chakraborty, I. Saha, A. Palchaudhuri, and G. K. Naik, "Hardware Trojan Insertion by Direct Modification of FPGA Configuration Bitstream," *IEEE Design & Test*, vol. 30, no. 2, pp. 45–54, 2013.

[67] N. Benhadjyoussef, H. Mestiri, M. Machhout, and R. Tourki, "Implementation of CPA Analysis against AES Design on FPGA," in *International Conference on Communications and Information Technology (ICCIT)*, 2012, pp. 124–128.

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

[69] W. Zhao, E. Belhaire, C. Chappert, and P. Mazoyer, "Spin Transfer Torque (STT)-MRAM–based Runtime Reconfiguration FPGA Circuit," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 9, no. 2, pp. 1–16, 2009.

[70] W. Zhao, J. Duval, J. Klein, and C. Chappert, "A compact model for magnetic tunnel junction (MTJ) switched by thermally assisted Spin transfer torque (TAS+ STT)," *Nanoscale Research Letters*, vol. 6, no. 1, pp. 1–4, 2011.

[71] P. C. Cheeseman, B. Kanefsky, and W. M. Taylor, "Where the Really Hard Problems Are," in *IJCAI*, vol. 91, 1991, pp. 331–340.

[72] H. M. Kamali, "Using Multi-core HW/SW Co-design Architecture for Accelerating K-means Clustering Algorithm," *arXiv preprint arXiv:1807.09250*, 2018.

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

[74] 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 Great Lakes Symposium on VLSI (GLSVLSI)*, 2018, pp. 459–462.

[75] M. Soos, K. Nohl, and C. Castelluccia, "Extending SAT Solvers to Cryptographic Problems," in *Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, 2009, pp. 244–257.

[76] G-J. Nam, F. Aloul, K. 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.

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

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

[79] G. Kolhe, H. M. Kamali, M. Naicker, T. Sheaves, H. Mahmoodi, Sai Manoj PD, H. Homayoun, Houman, S. Rafatirad, and A. Sasan, "Security and Complexity Analysis of LUT-based Obfuscation: from Blueprint to Reality," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[80] D. Chai and A. Kuehlmann, "A Fast Pseudo-Boolean Constraint Solver," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 3, pp. 305–317, 2005.

[81] M. Velev and P. Gao, "Comparison of Boolean Satisfiability Encodings on FPGA detailed Routing Problems," in *Design, Automation and Test in Europe*, 2008, pp. 1268–1273.

[82] M. Velev and P. Gao, "Efficient SAT Techniques for Absolute Encoding of Permutation Problems: Application to Hamiltonian Cycles," in *Eighth Symposium on Abstraction, Reformulation, and Approximation*, 2009.

[83] N. Manthey, M. Heule, and A. Biere, "Automated Reencoding of Boolean Formulas," in *Haifa Verification Conference*, 2012, pp. 102–117.

[84] M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory," *Journal of the ACM (JACM)*, vol. 7, no. 3, pp. 201–215, 1960.

[85] A. Biere, "Resolve and Expand," in *Int'l Conference on Theory and Applications of Satisfiability Testing*, 2004, pp. 59–70.

[86] N. Eén and A. Biere, "Effective Preprocessing in SAT through Variable and Clause Elimination," in *Int'l conference on theory and applications of satisfiability testing*, 2005, pp. 61–75.

[87] A. Biere, D. Le Berre, E. Lonca, and N. Manthey, "Detecting Cardinality Constraints in CNF," in *International Conference on Theory and Applications of Satisfiability Testing*, 2014, pp. 285–301.

[88] O. Bailleux and Y. Boufkhad, "Efficient CNF Encoding of Boolean Cardinality Constraints," in *International conference on principles and practice of constraint programming*, 2003, pp. 108–122.

[89] C. Sinz, "Towards an Optimal CNF Encoding of Boolean Cardinality Constraints," in *International conference on principles and practice of constraint programming*, 2005, pp. 827–831.

[90] J. Marques-Silva and I. Lynce, "Towards Robust CNF Encodings of Cardinality Constraints," in *International Conference on Principles and Practice of Constraint Programming*, 2007, pp. 483–497.

[91] R. Asín, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, "Cardinality Networks: a Theoretical and Empirical Study," *Constraints*, vol. 16, no. 2, pp. 195–221, 2011.

[92] M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa, "QMaxSAT: A Partial Max-SAT Solver," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 8, no. 1-2, pp. 95–100, 2012.

[93] D. Le Berre and A. Parrain, "The Sat4j Library, Release 2.2," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 7, no. 2-3, pp. 59–64, 2010.

[94] M. Gebser, B. Kaufmann, T. Schaub, "The Conflict-Driven Answer Set Solver Clasp: Progress Report," in *International conference on logic programming and nonmonotonic reasoning*, 2009, pp. 509–514.

[95] N. Eén and N Sörensson, "An Extensible SAT-solver," in *International conference on theory and applications of satisfiability testing*, 2003, pp. 502–518.

[96] E. Lee, G. Lemieux, and S. Mirabbasi, "Interconnect Driver Design for Long Wires in Field-Programmable Gate Arrays," *Journal of Signal Processing Systems*, vol. 51, no. 1, pp. 57–76, 2008.

[97] J. Luu *et al.*, "VTR 7.0: Next Generation Architecture and CAD System for FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 7, no. 2, pp. 1–30, 2014.

[98] J. Zhang, X. Tang, P. Gaillardon, and G. De Micheli, "Configurable Circuits Featuring Dual-Threshold-Voltage Design with Three-Independent-Gate Silicon Nanowire FETs," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 10, pp. 2851–2861, 2014.

[99] E. Giacomin, J. Gonzalez, and P. Gaillardon, "Low-power Multiplexer Designs using Three-Independent-Gate Field Effect Transistors," in *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, 2017, pp. 33–38.

[100] T. Meade *et al.*, "Revisit sequential logic obfuscation: Attacks and defenses," in *IEEE Int'l Symp. on Circuits and Systems (ISCAS)*, 2017, pp. 1–4.

[101] L. Li *et al.*, "Structural Transformation for Best-possible Obfuscation of Sequential Circuits," in *IEEE Int'l Symp. on Hardware Oriented Security and Trust (HOST)*, 2013, pp. 55–60.

[102] R. Chakraborty *et al.*, "HARPOON: An Obfuscation-based SoC Design Methodology for Hardware Protection," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1493–1502, 2009.

[103] F. Koushanfar, "Active Hardware Metering by Finite State Machine Obfuscation," in *Hardware Protection through Obfuscation*, 2017, pp. 161–187.

[104] J. Dofe and Q. Yu, "Novel Dynamic State-Deflection Method for Gate-Level Design Obfuscation," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 273–285, 2018.

[105] M. Fyrbiak *et al.*, "On the Difficulty of FSM-based Hardware Obfuscation," *IACR Trans. on Crypto Hardware and Embedded Systems (TCHES)*, pp. 293–330, 2018.

[106] A. R. Desai *et al.*, "Interlocking Obfuscation for Anti-Tamper Hardware," in *Proc. of the Cyber Security and Information Research Workshop*, 2013, pp. 1–8.

[107] Y. Shi *et al.*, "A Highly Efficient Method for Extracting FSMs from Flattened Gate-level Netlist," in *IEEE Int'l Symp. on Circuits and Systems (ISCAS)*, 2010, pp. 2610–2613.

[108] R. Tarjan, "Depth-First Search and Linear Graph Algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

[109] T. Meade *et al.*, "Netlist Reverse Engineering for High-Level Functionality Reconstruction," in *Asia and South Pacific Design Automation Conf. (ASP-DAC)*, 2016, pp. 655–660.

[110] I. G. Vargas *et al.*, "ROM-based Finite State Machine Implementation in Low Cost FPGAs," in *IEEE Int'l Symp. on Industrial Electronics*, 2007, pp. 2342–2347.

[111] M. T. Rahman *et al.*, "The Key is Left under the Mat: On the Inappropriate Security Assumption of Logic Locking Schemes," *Int'l Symp. on Hardware Oriented Security and Trust (HOST)*, pp. 1–10, 2018.

[112] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "NNgSAT: Neural Network guided SAT Attack on Logic Locked Complex Structures," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.

[113] A. Jain, Z. Zhou, and U. Guin, "TAAL: Tampering Attack on Any Key-based Logic Locked Circuits," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 26, no. 4, pp. 1–22, 2021.

Biography

Hadi Mardani Kamali has been a Ph. D. student in the Department of Electrical and Computer Engineering (ECE) at George Mason University, Fairfax, VA from 2017 to 2021. He was a member of Green, Accelerated, and Trustworthy Engineering (GATE) Lab, advised by Dr. Avesta Sasan. Hadi's main research interest lies in the areas of hardware security and trust, security for supply chain, and VLSI design and test. He received his Master's Degree in Computer Engineering from Sharif University of Technology, Tehran, Iran in 2013. Furthermore, He obtained his Bachelor's Degree in Computer Engineering from K. N. Toosi University of Technology, Tehran, Iran in 2011.