VULNERABILITY ASSESSMENT OF LOGIC LOCKING TECHNIQUES:
TOWARDS NEXT GENERATION ATTACKS ON LOGIC LOCKING

by

Kimia Zamiri Azar
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
B. Jabbari _____	Dr. Bijan Jabbari, Committee Member
Khaled N. Khasawneh _____	Dr. Khaled Khasawneh, Committee Member
Behzad Esmaeili _____	Dr. Behzad Esmaeili, Committee Member
Monson H Hayes _____	Dr. Monson H. Hayes, Department Chair
Kenneth S Ball _____	Dr. Kenneth S. Ball, Dean, The Volgenau School of Engineering

Date: _____ Summer 2021
George Mason University
Fairfax, VA

Vulnerability Assessment of Logic Locking Techniques: Towards Next Generation Attacks on Logic Locking

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

By

Kimia Zamiri Azar
Master of Science
Shahid Beheshti University, 2015
Bachelor of Science
K. N. Toosi University of Technology, 2013

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 husband, Hadi, 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. Bijan Jabbari, Dr. Khaled Khasawneh, and Dr. Behzad Esmaeili 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 husband, Hadi, 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. Hadi, 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

VULNERABILITY ASSESSMENT OF LOGIC LOCKING TECHNIQUES: TOWARDS NEXT GENERATION ATTACKS ON LOGIC LOCKING

Kimia Zamiri Azar, PhD

George Mason University, 2021

Dissertation Director: Dr. Avesta Sasan

To save the ever-increasing costs of maintaining an integrated circuit (IC) supply chain facility, take advantage of cutting-edge technology nodes, and meet the market demand, the manufacturing supply chain of ICs is globally distributed, known as the horizontal model in the supply chain. In the IC supply chain's horizontal model, separate entities fulfill various stages of design, fabrication, testing, packaging, and integration of ICs, forming a globally distributed chain. Outsourcing different stages of the manufacturing supply chain to the third-party facilities with no reliable monitoring on them results in identifying them as untrusted entities. Involving untrusted third-party facilities in supply chain manufacturing has introduced multiple forms of security threats such as IC overproduction, hardware Trojan insertion, reverse engineering (RE), intellectual property (IP) theft, and counterfeiting. To combat these threats, many design-for-trust countermeasure mechanisms have been widely studied in the literature, such as watermarking, IC metering, IC camouflaging, split manufacturing, and logic obfuscation. Amongst them, logic obfuscation *a.k.a.* logic locking, as a proactive scheme, has received significant attention in recent years, in which the designer would be able to add post-manufacturing programming capability into the circuits.

Logic obfuscation is the process of hiding the correct functionality of a circuit, during the stages at untrusted parties, when the programming value, referred to as the key, is unknown. Only once the correct key is provided, the circuit behaves correctly, and the correct key would be initiated and stored in a tamper-proof non-volatile memory after fabrication at a trusted party. However, the introduction of different de-obfuscation attacks, particularly Boolean satisfiability (SAT)-based attacks, have undermined the effectiveness of the vast majority of existing logic locking countermeasures.

The evolution of different de-obfuscation attacks in recent years results in the introduction of numerous logic locking solutions, which makes them resilient and robust against many of the state-of-the-art de-obfuscation attacks, including SAT-based attacks. In this thesis, we first provide a comprehensive overview of the state-of-the-art defense and attack mechanisms in logic locking. Then, we reveal some limitations of the existing attack mechanisms leading us to introduce newer and stronger attack approaches with much more capabilities and performance compared to the existing ones. For this purpose, we introduce the satisfiability modulo theory (SMT) attack, in which the adversary has the capability of modeling non-Boolean logic locking mechanisms using theory solvers. SMT attack is the first of its kind that is able to model non-Boolean characteristics of the circuit. Then, we will introduce the neural network guided SAT (NNgSAT) attack that exploits the benefit of a message passing neural network (MPNN) to reduce the complexity of the de-obfuscation model, especially when complex structures, such as routing modules and multipliers, are parts of the logic locked circuits.

After that, we also go one step further and propose two new countermeasures to combat state-of-the-art attacks. Unlike almost all state-of-the-art logic locking solutions that focus on functional/logic locking, we introduce a new logic locking paradigm, called data flow obfuscation, which targets the flow/timing of the circuit as a new means for logic locking. We exploit the essence of asynchronicity to lock the flow/timing of the circuits, making it almost impossible to be modeled/formulated using state-of-the-art attacks.

We also introduce a communication and obfuscation management architecture (COMA), as an alternative solution, for enhancing the security of logic locking against the existing attack. The main aim of COMA is to protect the main secret of logic locking, which is the logic locking key, from being stolen or revealed. For this purpose, in COMA, we propose an architecture allowing the designer to store the logic locking key outside of the IC that is manufactured in an untrusted foundry.

As a result, the outcome of this thesis aims to provide an assessment of the capabilities and limitations of the existing studies on logic locking, either defense or attack mechanisms. By introducing newer and stronger approaches, this research also opens new directions for the designers to evaluate the security of the designs using more appropriate and well-formulated mechanisms, leading to stronger and more reliable design and implementation with enhanced security.

Chapter 1: Introduction to Hardware Obfuscation

1.1 Hardware Security Vulnerabilities

The cost of building a new semiconductor fab was estimated to be $5.0 billion in 2015, with large recurring maintenance costs [7], and sharply increases as technology migrates to smaller nodes. To reduce the fabrication cost, take advantage of cutting-edge technology nodes, and meet the market demand, the manufacturing supply chain of ICs is globally distributed [7]. Fig. 1.1 demonstrates the main steps in the life cycle of an IC, from the specification at the design house to fabricated/packaged chip at the field. In the IC supply chain with a globalized model, *a.k.a* the horizontal model, separate entities fulfill various stages of design, fabrication, testing, packaging, and integration of ICs, forming a globally distributed chain. As demonstrated in Fig. 1.1, the red parts indicate untrusted entities in the IC supply chain. Outsourcing and the involvement of numerous stakeholders in various stages of the supply chain dramatically reduce the cost and time-to-market of the chip [7]. However, it reduces the control of original manufacturers and IP owners/vendors over the supply chain. The loss of control will results in numerous security vulnerabilities, including but not limited to *IP piracy*, *Overbuilding*, *hardware Trojans*, *counterfeiting*, and *Reverse Engineering (RE)* [8, 9].

The *IP piracy* refers to the illegal use of the intellectual property. For example, an attacker in a design house, can steal valuable IP cores and sell them as genuine. The *Overbuilding* means that a rogue foundry can overproduce extra ICs and sell them illegally. *Hardware Trojans*, are malicious modifications to original circuitry inserted by adversaries to exploit hardware or to use hardware mechanisms to create backdoors in the design. These backdoors can leak sensitive (or private) information as well as enable launching other possible attacks. *Counterfeit* ICs are replicas of the genuine ICs that are fraudulently

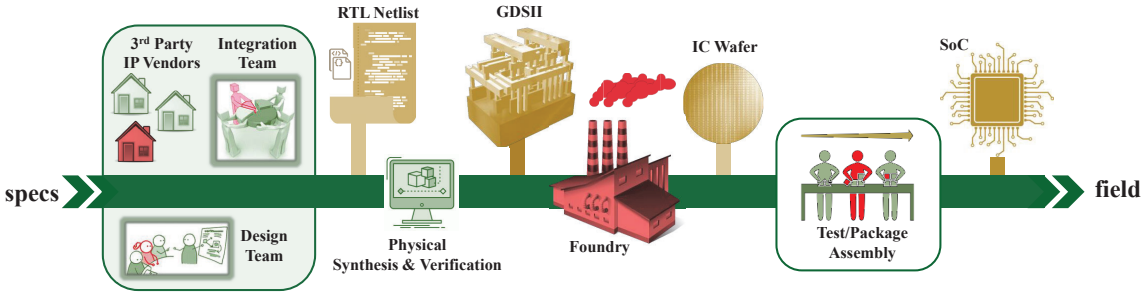


Figure 1.1: Major Steps in IC Supply Chain.

made to appear almost identical to the genuine ICs [10]. *Reverse engineering* is a complex process involving steps, such as attempts to infer the functionality of the design, extraction of the gate-level netlist, and identification of the device technology [11].

1.2 Design-for-Trust Techniques

To counter these threats, various hardware design-for-trust (DfTr) techniques have been widely studied in the literature, including *watermarking*, *IC metering*, *split manufacturing*, *IC camouflaging*, and *logic locking* [12–16]. 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. In *split manufacturing* the design is split into two parts, corresponding to the back end of the line (BEOL) and front end of the line (FEOL) metal layers, that are manufactured in separate foundries and ultimately stacked together. However, it can prevent piracy only by an untrusted foundry, and not by an end-user. The *Camouflaging* techniques use logic gates (or other physical structures such as dummy vias) with high structural similarity, that are indistinguishable from one another to protect against reverse engineering. However, camouflaging is only effective against post-manufacturing attempt(s) of reverse engineering, while it provides no limitations against a foundry's attempt at reverse engineering, as a foundry has access to all masking layers and is not trapped by structural ambiguity for being able to logically extract a netlist. The logic obfuscation, *a.k.a. logic locking*, on the other hand, introduce limited programmability by

inserting key programmable gates to hide or lock the functionality. By using obfuscation, the target chip produces the correct output only when the key inputs are correct. The purpose of obfuscation is to protect against reverse engineering at an untrusted foundry. By using obfuscation, even by having all mask information, the attacker cannot generate the correct functionality of a circuit without the correct key values, and such key values are not shared with the manufacturer.

logic locking techniques, however, did not end the threat against different security vulnerabilities, as these solutions that were proposed over the last decade were broken using various carefully crafted attacks. As demonstrated in Fig. 1.2, a decade of research in this area has resulted in a wide range of defense and attack mechanisms. Amongst attack mechanisms introduced in logic locking, the Boolean satisfiability (SAT) attack could be considered as the turning point in this topic [1]. In this attack model, the attacker has access to (1) a reverse-engineered but obfuscated netlist, (2) a functional (unlocked) chip with open access to the scan chain. Using this attack model, the formulated Boolean Satisfiability Attack (SAT Attack) can effectively break all previously proposed logic locking techniques, including random insertion (RLL), fault-based logic locking (FLL), strong logic locking (SLL), and logic barriers [16–20]. The SAT attack has an iterative structure, and in each iteration, a SAT solver is used to find a specific input, called discriminating input pattern (DIP) that finds two sets of keys producing different outputs. After finding all DIPs, the SAT solver can eliminate all incorrect keys leading to the extraction of the correct functionality of the circuit. In general, the SAT attack could eliminate all incorrect keys within a few iterations, leading to retrieving the correct functionality of the circuit, and unlike the brute force attack that requires attack time exponential with respect to the number of inputs, its execution time grows almost polynomially. The original SAT attack is applicable to combinational circuits. However, the existence of the much-needed scan chain for functional and structural testing, makes the sequential circuits also vulnerable to this attack when an adversary gains access to the scan chain. We will elaborate the details of the SAT attack in Chapter 2.

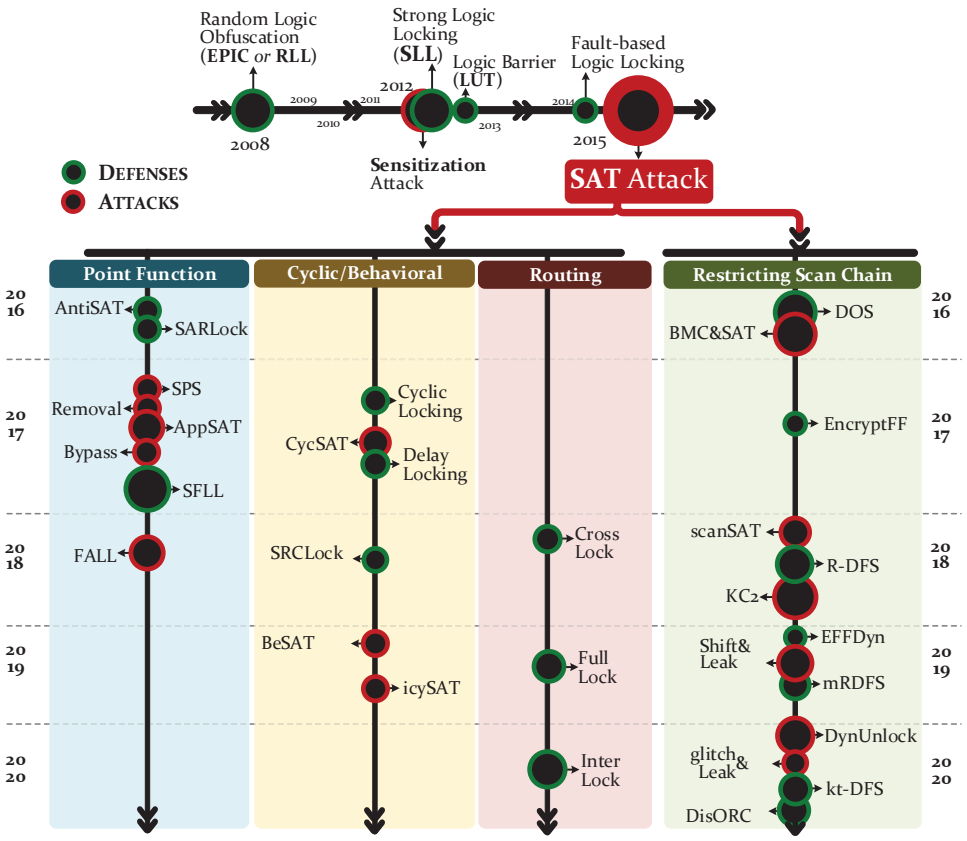


Figure 1.2: Logic Locking: Defenses and Attacks over the Time.

1.3 Post-SAT Logic Locking Era

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 de-obfuscation) 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, as demonstrated in Fig. 1.2, there are four main groups of countermeasures that have been introduced in the literature to show how a logic obfuscation technique could be built to defeat the SAT attack. Three out of four groups try to either

weaken the pruning power of DIPs or introduce a solution that could not be formulated by the SAT attack, which could be enumerated as: (1) point-function structure, (2) cyclic and behavioral obfuscation, and (3) routing obfuscation. However, the main focus of the fourth group of countermeasures, on the other hand, is to (4) restrict any unauthorized access to the scan chain to completely invalidate the possibility of engaging the SAT attack. Although these countermeasures provide robustness against the powerful SAT attack, further studies show how these countermeasures still suffer from fundamental shortcomings leading to the introduction of newer attacks on them. In Chapter 2 we review many of these obfuscation solutions and attack mechanisms in more detail, summarize and compare the effectiveness of obfuscation solutions against these attacks, and describe the strength and weaknesses of various obfuscation and attack solutions.

1.4 Problem Statement

As we discussed previously, the introduction of the SAT attack could be considered as the turning point in logic locking studies. Hence, as demonstrated in Fig. 1.2, all logic locking countermeasures introduced after 2015 (after the introduction of the SAT attack), aim to break this powerful de-obfuscation attack. However, some post-SAT logic locking countermeasures still suffer from fundamental deficiencies. For instance, the first group in Fig. 1.2, i.e. point function-based logic locking techniques, such as Anti-SAT, SARLock, and SFLL [21–23], suffer from various structural and functional vulnerabilities that were eventually exploited to break them [24–27].

1.4.1 Requiring More Powerful Attacks

Apart from those logic locking techniques with different vulnerabilities, there exist some other countermeasures that formulate a resilient and robust logic locking mechanism against the SAT attack as well as other state-of-the-art de-obfuscation attacks. In such mechanisms, different attributes and characteristics have been targeted and employed for obfuscation

purposes to add complexity into the designs, making them much more difficult to be modeled and broken using state-of-the-art attacks. For example, the SAT attack benefits from the directed acyclic graph (DAG) based nature of the input netlist and the ability of SAT attack to logically model the obfuscation into a satisfiability problem. To counter the SAT attack, recently some design obfuscation schemes have been proposed to violate these assumptions. For instance, in some techniques of the second group of Fig. 1.2, i.e. cyclic/behavioral obfuscation, the key-programmable combinational cycles are added into the design, which traps the SAT solver in an infinite loop [28,29]. Some other approaches focus on obfuscating the behavioral and analog parameters of the design [30, 31]. One example is the approach adopted in [31], where the obfuscation, in addition to logical properties of the netlist, targets the setup and hold properties (timing properties) of the circuit as a locking mechanism. Considering that setup and hold time are not logical properties, they cannot be translated into CNF statements for formulating the SAT attack.

Similarly, in the third group of Fig. 1.2, i.e. routing-based obfuscation, to weaken the SAT attack, the main aim of the obfuscation is to increase the complexity of the SAT problem, thereby the run-time of *each iteration* of the SAT solver would be increased substantially [32, 33]. In such techniques, by exploiting the strength of symmetric routing structures, such as permutation networks or crossbars, the complexity of the SAT circuit per each iteration would be increased significantly. In general, when the logic locked circuits contain complex structures, such as large routing networks, big multipliers, or big tree structures, the logic locked circuit is *hard-to-be-solved* for the SAT attack. Usage of these structures for obfuscation may lead to a strong defense, as many SAT solvers fail to handle such complexity.

1.4.2 Requiring More Powerful Defenses

During the last decade, numerous attacks have been introduced in the literature, showing the vulnerability of the prior art logic locking techniques. The basics of logic locking techniques have evolved along with the evolution of de-obfuscation attacks, and over the time both

the attack and defense mechanisms became increasingly sophisticated. The logic locking techniques, however, are not yet as mature as those in the field of Cryptography, and almost none of them is simultaneously impeccable against different de-obfuscation attacks.

1.4.3 Requiring Secure Logic Locking Key Architecture

Apart from making the logic locking techniques resistant against different de-obfuscation attacks, the secret of logic locking, which is the key of logic locking, must also resist passive, active, or destructive attacks that could be deployed to read the logic locking key values. Mostly, the key of logic locking will be stored in secure and tamper-proof memory (TPM) [34] within the circuit, and during the activation, the key value will be initiated in TPMs, and in each power ON, the logic locking key from TPM will be loaded into the design. Hence, neither the activation of such devices nor the storage of key values in them should expose or leak the key information.

1.5 Motivation

With considering the above-mentioned shortcomings and limitations of attacks and defenses in logic locking literature, in this thesis, we are motivated to address these concerns as follows:

(1) The introduction of such complex logic locking countermeasure techniques, particularly in the second and the third groups of Fig. 1.2, with no successful attack on them motivates us to investigate their vulnerabilities and the possibility of breaking such techniques. With the comprehensive investigation on existing de-obfuscation attacks, we found that most of them get the benefit of formal verification methods to model and break the existing logic locking. For instance, the SAT solver is widely used for the verification of RTL design, and in the SAT attack, this solver has been engaged to model almost all logic locking solutions. Now the big question is that "how we can extend and enhance the capability of de-obfuscation attacks using formal verification methods that are not employed so far?". With a concentration on this approach, in this thesis, with the introduction of two

new attacks, we show how we can extend the capability and performance of the SAT attack using more powerful computing and decision-making engines.

(2) Since there exists a direct relationship between the strength of the de-obfuscation attack and the formal method used for de-obfuscation, in this thesis, we also aim to answer this question that "how we can model and formulate a new logic locking solution, which is hard-to-be-modeled using the existing formal verification tools and methods?". Hence, with going one step further, and with the introduction of a new logic locking solution, we show how we can harden the process of formal-based modeling and make any de-obfuscation attack almost impossible on such techniques.

(3) Additionally, since all the de-obfuscation attacks on logic locking rely on the fact that the logic locking key is stored within the design, in this thesis, we will investigate how we can prevent following such assumption. For instance, as an alternative solution, the logic locking key could be stored outside of the chip. However, such a scenario requires constant connectivity to the key management source and secure communication for the key exchange to prevent any leakage of the key. Now the question is that "how we can design and implement a new architecture to support such capability?". Thus, with the introduction of a new communication and obfuscation management architecture, applicable to the specific families of designs, we show how we can prevent storing the key within the design.

1.6 Contribution of this Thesis

To address the research problems described above, we propose some approaches to extend the capability and performance of existing de-obfuscation attacks, which allow us to have a much more appropriate framework for security evaluation of logic locking in the future. We also introduce newer logic locking countermeasures based on the updated framework to extend the reliability and robustness of logic locking techniques. The main contributions of this thesis are as follows:

1. We first provide a comprehensive review of the current status of logic locking in the

literature. After reviewing the current status, we will show how the proposed schemes in this thesis will help us towards a more appropriate security evaluation on logic locking.

2. We introduce a satisfiability modulo theory (SMT) attack on logic locking. SMT attack could be considered as the superset of the SAT attack with much more capabilities and better performance. The SMT attack would be able to engage different theory solvers to model the non-Boolean nature of the circuit. The main aim of the SMT attack is to reveal the vulnerabilities of the second group of Fig. 1.2, i.e. cyclic/behavioral logic locking techniques. In the SMT attack, with engaging a graph theory solver, in one case study, we show how the SMT attack could be used to break the logic locking mechanisms that target the non-Boolean nature of the design.

3. We exploit a specific neural network engine for enhancing the performance of existing SAT-based attacks. The main aim of this new attack, called neural network guided SAT (NNgSAT) attack is to target and investigate the advantage of using neural network for accelerating the de-obfuscation execution time in the third group of Fig. 1.2, i.e. routing-based logic locking techniques. In NNgSAT, we show how complex structures, such as routing blocks, could be solved with a message passing neural network (MPNN)-based SAT solver.

4. We introduce a reliable logic obfuscation technique that meets the main requirements of a well-designed and appropriate logic locking solution: (1) Hard-to-be-modeled using the existing de-obfuscation attacks, such as the SAT attack and any other state-of-the-art attacks; and (2) Have a low impact on the circuit, including low overhead without compromising the test/implementation flow. To fulfill these requirements, we introduce a new logic locking paradigm, called *data flow obfuscation*, which exploits the essence of asynchronicity. In data flow obfuscation, by benefiting from the handshaking mechanism of asynchronous circuits, the system's FFs/latches will operate out of sync. Hence, the adversary has no sufficient knowledge to apply the

sequential SAT attack. Also, due to the inherited asynchronicity, the exact time of writing/capturing data into/from the scan chain becomes hidden. Hence, the SAT attack cannot be applied even while scan chain access is open. Moreover, this newly proposed paradigm creates stateful/oscillating combinational cycles into the design which extensively boosts the difficulty of modeling this technique.

5. To avoid storing the key within the design, we propose a communication and obfuscation management architecture (COMA) for secure activation of obfuscated circuits that are manufactured in untrusted foundries and meet the constant connectivity requirement, namely ICs that belong to a) 2.5D package-stack devices and b) IoT devices with constant connectivity to the cloud/internet. We describe two variants of our proposed solutions: The first one is used for secure activation of IPs within 2.5D package-integrated devices (similar to DARPA SPADE). The second variant is used for the secure activation of connected IoT devices. The proposed COMA allows us to (1) push the obfuscation key and obfuscation unlock mechanism off of an untrusted chip, (2) make the key a moving target by changing it for each unlocks attempt, (3) uniquely identify each IC, (4) remove the need for implementing a secure memory in an untrusted foundry.

1.7 Thesis Organization

The organization of this thesis is as follows: In Chapter 2, the background and the history of logic locking have been described. Chapter 3 describes the SMT attack, and how the theory solvers could be engaged in it to extend the capability and performance beyond the SAT attack. Chapter 4 will explain how we engage message passing neural network (MPNN) as an acceleration engine to guide the SAT attack, which is applicable to logic locked circuits with complex structures. In Chapter 5, the overall structure of data flow obfuscation has been elaborated. We demonstrate how data flow obfuscation could be resistant against all state-of-the-art de-obfuscation attacks. Chapter 6 provides the detail of communication

and obfuscation management architecture (COMA), which helps us to keep the secret of the logic locking outside of the chip that is manufactured in an untrusted foundry to protect the design from being stolen. Finally, Chapter 7 concludes the thesis by summarizing the current status of logic locking as well as suggestions for future research directions in this topic.

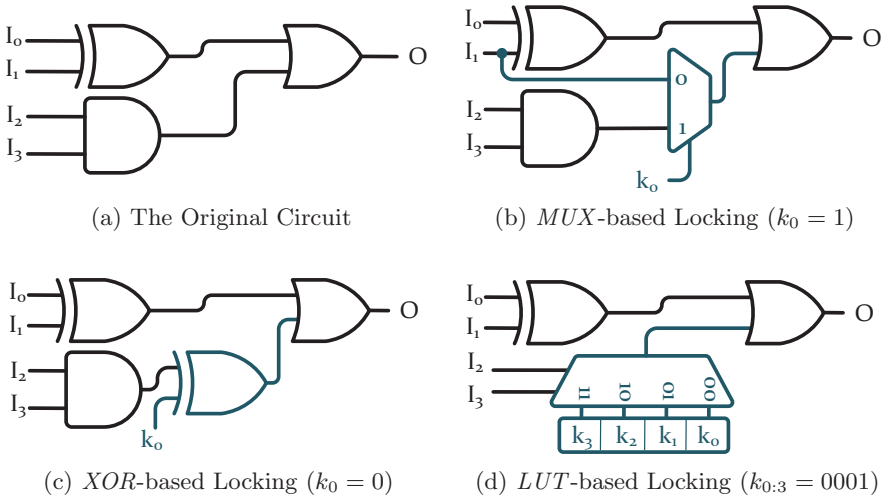
Chapter 2: Background on Logic Locking: Defenses and Attacks

As discussed in Chapter 1, logic locking techniques did not end the threats against IP piracy (or other related concerns), as these solutions that were proposed over the last decade were broken using various carefully crafted attacks. In this Chapter, after introducing the basic definitions of hardware obfuscation, *a.k.a. logic locking*, we review many of these obfuscation solutions and distinguish between weak and robust logic locking solutions. Then by highlighting the weaknesses of existing solutions, we explain and review the most notable attack mechanisms that successfully break the existing countermeasures. Then, we summarize and compare the effectiveness of obfuscation solutions against these attacks, and describe the strength and weaknesses of various obfuscation and attack solutions.

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 obfuscation, respectively. Fig. 2.1 depicts a simple example of each of these models. As an instance, assuming that the original circuit is demonstrated in Fig. 2.1(a), Fig. 2.1(c) shows a simple XOR-based obfuscated (locked) version of this circuit. In this case, if $k_0 = 1$, the second input of the final OR gate would be toggled, thus resulting in the corruption of the output. For this example, when the $k_0 = 0$, the correct functionality would be recovered.

During the last decade, different logic locking solutions commonly have engaged the above-mentioned basic gates with different structures/functions for obfuscation purposes.



(a) The Original Circuit (b) *MUX*-based Locking ($k_0 = 1$)

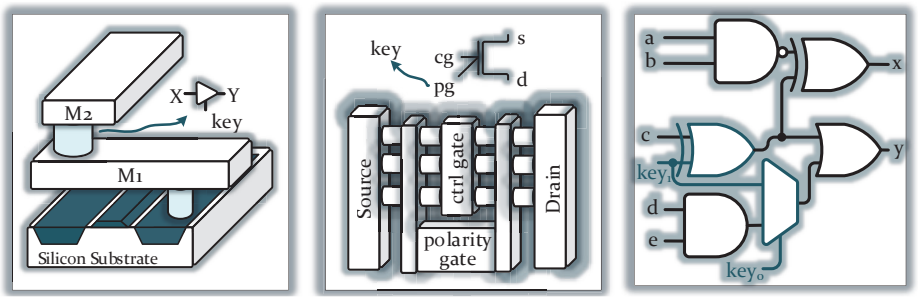
(c) *XOR*-based Locking ($k_0 = 0$) (d) *LUT*-based Locking ($k_{0:3} = 0001$)

Figure 2.1: Basic Gates used for Logic Locking.

Based on the location, structure, count, intercorrelation, etc., of these gates, the countermeasures provide various levels of robustness 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, which connects two adjacent metal layers, has been engaged as key-based programmable unit for routing-based locking [35]. In general, moving from layout-level to HLS- or architecture-level will mitigate implementation effort; however, at a lower level of abstraction, finding a logic locking countermeasure 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, an example of which has been demonstrated in Fig. 2.2(c), that could be done as a post-synthesis stage on the synthesized netlist in the supply chain.

One important property of logic locking techniques is the *output corruptibility*. Corruptibility means that when an incorrect key is applied to the locked circuit, (1) for how many output pins, and (2) for what percentage of the input patterns, the primary output (PO) will be corrupted. Based on the location, structure, count, intercorrelation, etc., of the key-based XORs/MUXs/LUTs that are engaged for locking purposes, the corruptibility



(a) (b) (c)

```
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";
```

(d)

```
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];
}
```

(e)

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

will change. Corruptibility directly affects the resiliency of the countermeasure against the existing attacks. For instance, if the corruptibility is low, it allows the adversary to look for a specific way for only those POs affected or those specific input patterns that produce output corruption. For a well-designed logic locking countermeasure, the corruptibility must be high to avoid such vulnerabilities.

The key of logic locking will be initiated and stored in a tamper-proof non-volatile memory (tpNVM) after the fabrication via a trusted party. At power UP of a locked IC, as a part of the boot process, 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 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

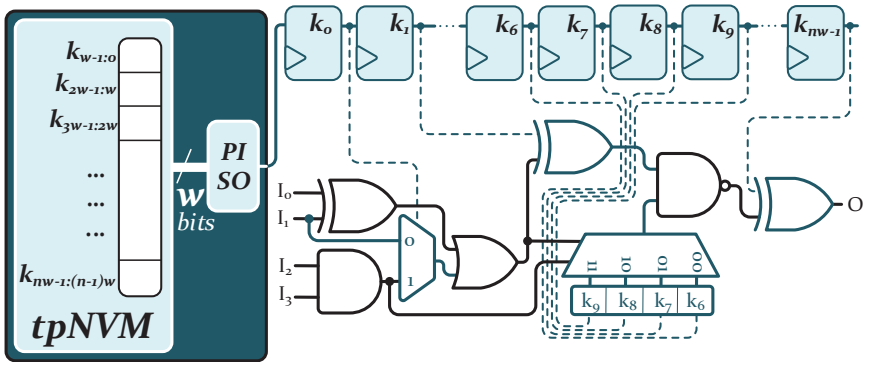


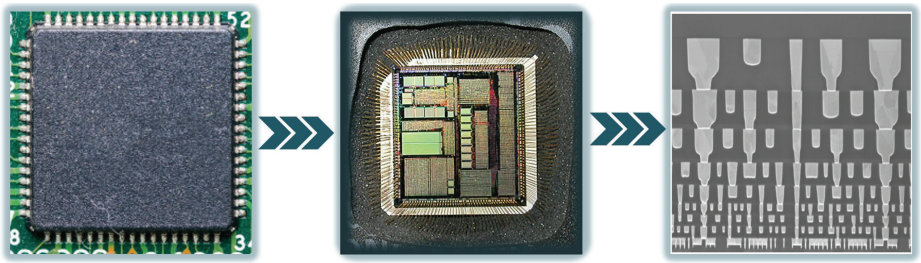
Figure 2.3: Logic Locking Key Initialization from tpNVM.

via parallel-in serial-out (PISO) module, and (3) temporary registers that stores the logic locking key while the IC is power ON.

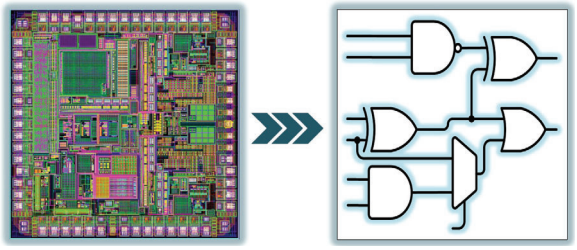
2.2 Models and Assumptions in Attacks on Logic Locking

Based on the threat models and assumptions evaluated in the de-obfuscation attacks on logic locking, the attacks could be categorized into different sub-groups. Some of the attacks require access to one additional activated version of the fabricated circuit, known as *oracle*. This group of 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, most of the attacks are members of *oracle-guided* attacks.

Many of the de-obfuscation attacks require access to the netlist of the chip. Acquiring the netlist of the chip could be accomplished in two common scenarios: (1) the adversary as an end-user can obtain the fabricated 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 this case, during the physical reverse engineering, since the key is stored in tpNVM, it will be wiped out in the de-packaging stage; (2) the adversary might be located at the foundry, and they receive the GDSII of the chip from the design house to be fabricated. The GDSII at the foundry is locked, and the



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



(d) Extracting Layout (e) Extracting the Netlist

Figure 2.4: The Main Steps of Reverse Engineering.

adversary at the foundry has no information about the key. Hence, in both cases, either at the foundry or at the field/market, the netlist acquired by the adversary would be locked.

Unlike the above-mentioned attacks that require access to the locked netlist, there exists a very limited number of de-obfuscation attacks, in which the adversary relies on 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 circuits containing the secrets. So, regardless of the logic locking technique used in the circuit, this group of attacks would be able to reveal the security assets like logic locking key without requiring to have access to the locked netlist.

The availability of design-for-testability (DFT) structure, i.e. 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 OPEN*. Fig. 2.5 shows a simplified scan architecture with two scan 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

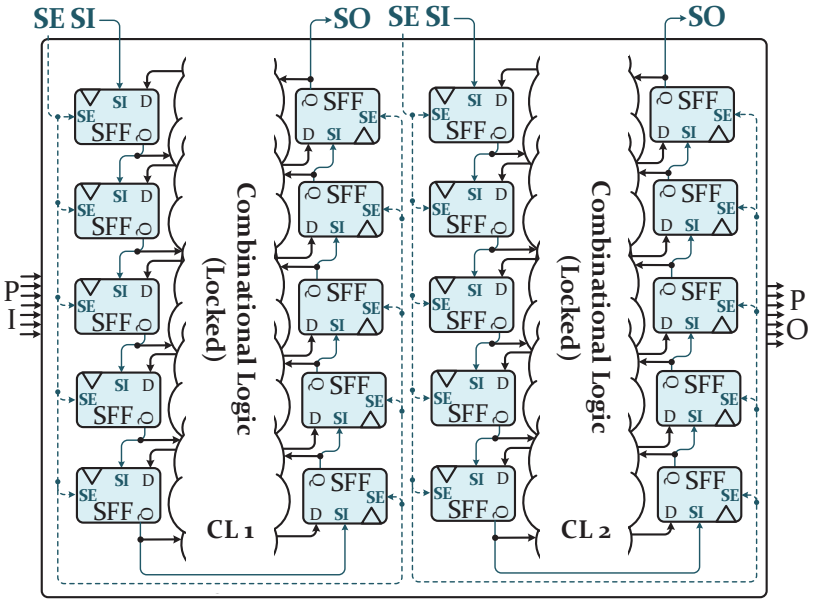


Figure 2.5: Scan Chain Architecture in ICs.

Fig. 2.5, whose FFs are part of the scan chain. The scan chain access allows the adversary to divide the de-obfuscation problem into a bunch of much smaller sub-problems (for 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 de-obfuscation attacks have studied and demonstrated the possibility of retrieving the correct key/functionality of the locked circuit via primary inputs/outputs (PI/PO).

2.3 Most Notable Attacks and Defenses in Logic Locking

Based on the models and assumptions we previously discussed, all attacks on logic locking could be categorized into different groups, each aiming to target and break one or a few logic locking countermeasure techniques. Since many of the existing attacks on logic locking assume that the oracle is available for the adversary, in this Chapter, we will review defenses and attacks in which the availability of the oracle is met [36]. Most of the

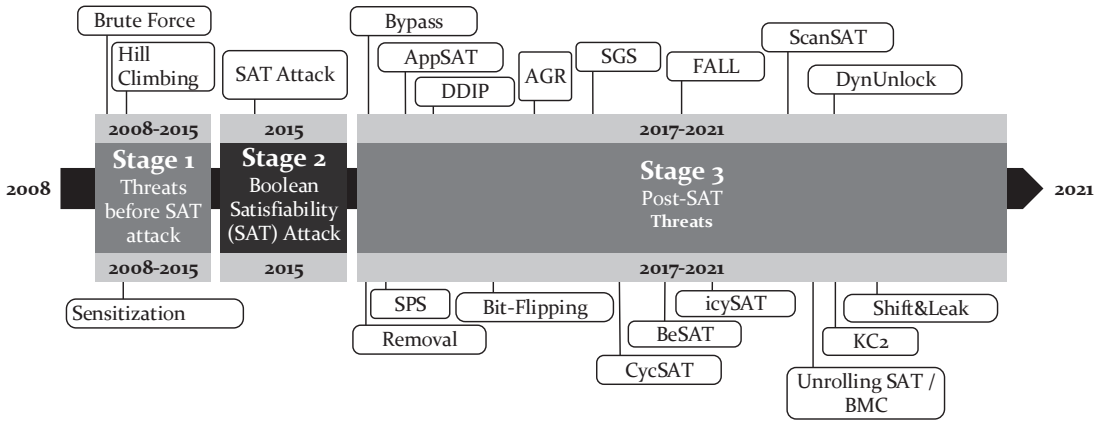


Figure 2.6: Categorization of Attacks against Logic Locking Schemes.

oracle-guided attacks could be also known as algorithmic attacks, in which, a systematic flow has been proposed that results in the exposure of either logic locking key or the correct functionality of the locked circuit. Fig. 2.6 depicts almost all existing oracle-guided attacks, each targeting one or a few logic locking solutions. As illustrated in Fig. 2.6, these attack mechanisms, based on functionality, capability, effectiveness, and time-line are categorized into three categories: (1) *Test-inspired attacks* that were mostly inspired from test concepts and attempted to discover the obfuscation key value based on the location of Key Gates (KGs). (2) SAT attack, formulation of which significantly affected the direction and previous assumptions of the hardware obfuscation research community. (3) Post-SAT Attacks, where the focus of hardware security researchers changed to the design of an attack against obfuscation solutions that resist the SAT attack.

2.4 Stage 1: Test-Based De-obfuscation Attacks

Shortly after the introduction of the very first logic locking solution, i.e. random-based logic locking (RLL or EPIC) [16], in which random placement strategy has been used for inserting key-based XORes, the possibility of using testability and fault analysis attributes was investigated as a means of attack on logic locking. Many of these de-obfuscation attacks exploit almost the same techniques/algorithms that are widely used for automatic

test pattern generation (ATPG). In this section, we review these attacks that rely on the ATPG-based techniques/algorithms.

2.4.1 Brute Force Attack

The *brute force* attack is the most intuitive attack against obfuscated circuits, which is very similar to an exhaustive test. This attack exhaustively searches for the correct key by testing all key and input values. For instance, assuming that adversary has access to the reverse-engineered netlist, and considering that the circuit has four primary input (PI) pins ($i_{0..3}$) and two key inputs (KI)s ($k_{0..1}$), an exhaustive search may result in applying of $2^{2+4} = 64$ test patterns (in the worst case) and checking the output against an activated (functionally correct) chip to verify correctness. Based on the number of primary inputs ($|PI|$) and the number of key bits ($|KI|$), the number of possible test patterns is ($2^{|PI|+|KI|}$). Hence, the search space for a brute force attack is extremely large, making the attack even for small circuits and a small number of keys unfeasible in a reasonable amount of time. For example, a small circuit with 20 input pins, which is obfuscated with 80 key gates poses 2^{100} possible test pattern. An attacker can reduce the number of test patterns using functional test or random test, in which the exponential impact of $|PI|$s will be eliminated, and only $2^{|KI|} \times (func\_test\_patterns)$ is required for brute force attack. But even with this change, the attack time is exponential with respect to the number of key gates.

2.4.2 Sensitization Attack

After introducing the RLL (EPIC) [16], Rajendran *et al.* [18] proposed a *sensitization* attack, which determines individual key values, in a time linear to the $|KI|$, by applying patterns that sensitize key values to the primary outputs (PO)s. As its name implies, sensitization of an internal wire (key bit) **L** to an output **PO** means that the value of **L** can be propagated to **PO** and thus any change on **L** is observable on **PO**. After determining an input pattern that propagates the value of the key-bit to the output, the attacker applies

Table 2.1: Classification of KGs in Sensitization Attack.

Term	Description	Strategy used by attacker
Runs of KGs	Back-to-Back KGs	Replacing by a Single KG
Isolated KGs	No Path between KGs	Finding Unique Pattern per KG (Golden Pattern (GP))
Dominating KGs	$k1$ is on Every Path between $k0$ and POs	Muting $k0$, Sensitizing $k1$
Concurrently Mutable Convergent KGs	Convergent at a Third Gate Both can be Propagated to POs	Muting $k0/k1$, Sensitizing $k1/k0$
Sequentially Mutable Convergent KGs	Convergent at a Third Gate One can be Propagated to POs	Determining $k1$ by GP, Update the Netlist, Target $k0$
Non-Mutable Convergent KGs	Convergent at a Third Gate None can be Propagated to POs	Brute Force Attack

the input pattern to a functional IC (oracle). The correct key value will be propagated to output by applying this pattern to the oracle. The attacker observes and records this output as the value of the sensitized key-bit, and by using such technique all keys could be sensitized and observed at POs.

The propagation of a key-bit to the output is heavily dependent on the location of the key gates (KG)s, hence, they classify KGs based on their location and discuss corresponding attack strategies for each case. The summary of strategies and techniques used in the sensitization attack is reflected in Table 2.1. To combat sensitization attack, they also proposed a countermeasure, called SLL, in which the KGs are inserted in locations with maximum mutual interference. The key value of the KGs located at maximized mutual interference cannot be sensitized and propagated to the POs. Similar to SLL, several prior-art methods in the literature, including fault-analysis-based logic locking (FLL), LUT-based locking, etc. [17–19,37] tried to maximize the complexity of obfuscation using different KGs replacement strategies.

2.4.3 Random-based Hill-Climbing Attack

Plaza *et al.* [38] developed a new algorithmic attack that uses test patterns and observes responses. Unlike sensitization attack [18], their proposed approach does not require netlist

access. They propose a randomized local key-searching algorithm to search the key that can satisfy a subset of correct input/output patterns. The algorithm proposed in [38] is iterative in nature. At first, it selects a random value for key bits, and then at each iteration, the key bits, which are selected randomly, are toggled one by one. The target is to minimize the frequency of differences between the observed and expected responses. Hence, a random key candidate is gradually improved based on observed test responses. If no solution is found in one iteration, the algorithm resets the key to a new random key value. However, the complexity of this attack quickly increases with the increasing number of KGs.

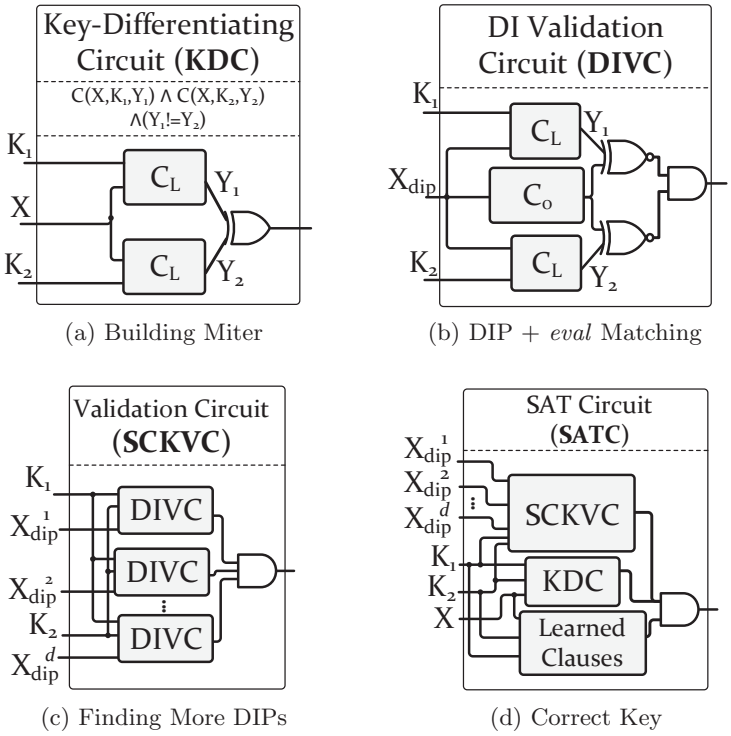
The main purpose of this attack is to break the very first logic locking technique, i.e. RLL [16]. However, in many cases, it faces a very long execution time with no results. This happens for two main reasons which significantly undermine the success rate of this attack: (1) The key will be initiated randomly, and (2) The complexity of the attack will be increased drastically, particularly when the key size is large or the key bits are correlated.

2.5 Stage 2: SAT-based Attacks

Solving a Boolean *satisfiability* problem is the process of satisfying a Boolean expression or equation. In 2015, Subramanyan *et al.* [1] propose a new and powerful attack on logic locking that gets the benefit of the SAT solver, which is used to solve a satisfiability problem. The engagement of the SAT solver as a means for attacking the locked circuits has got the most attention in recent years for some important reasons, such as the strength, the performance of the attack, and the scalability.

2.5.1 Traditional/Pure SAT Attack

The SAT attack was first introduced by Subramanyan *et al.* [1]. At the same time, El Massad *et al.* proposed the same technique, in which a SAT solver is engaged for attacking the combinational logic locked netlist [2]. Getting inspired by the miter (distinguisher) circuit that is widely used for formal verification, the SAT attack uses a specific duplication



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

(c) Finding More DIPs (d) Correct Key

Figure 2.7: The SAT Attack Iterative Flow [1, 2].

mechanism to break the logic locked netlists.

The main steps of the SAT attack have been demonstrated in Fig. 2.7. As shown in Fig. 2.7(a), in the SAT attack, the attacker first duplicates the locked circuit and builds a double circuit known as the key-differentiating circuit (KDC). The KDC is used for finding an input (X_d[i]) that for two different key values, this input generates two different outputs. The key values and the X_d[i], will be found by a SAT solver query. Such input is referred to as the discriminating/distinguishing input pattern (DIP). Each DIP (X_d[i]) is used to create a DIP validation circuit (DIVC). The validation circuit, as shown in Fig. 2.7(b) assures that for a previously found DIP, two different keys generate the same output value (part of the correct key pool). Each iteration of the SAT attack finds a new DIP and adds a new DIP validation circuit (DIVC) to the whole problem. The DIVCs are then *AND*ed together to form a correct key validation circuit (SCKVC). In each iteration, the SAT solver

tries to find a new DIP and two key values that satisfy the double circuit (KDC) and the validation circuit (SCKVC). This iterative 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 previously found X_ds is the correct key. Algorithm 1 provides an algorithmic representation of the SAT attack, which has an iterative structure for finding all DIPs.

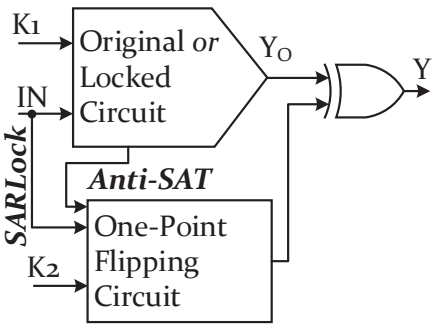
Algorithm 1 SAT-based Attack Algorithm [1]

```
1: function SAT_ATTACK(Circuit C_L, Circuit C_O)
2:     i ← 0;
3:     F_0 ← C_L(X, K_1, Y_1) ∧ C_L(X, K_2, Y_2);
4:     while SAT(F_i ∧ (Y_1 ≠ Y_2)) do
5:         X_d[i] ← sat_assignment (F_i∧(Y_1 ≠Y_2));
6:         Y_d[i] ← C_O(X_d[i]);
7:         F_{i+1} ← F_i ∧ C_L(X_d[i], K_1, Y_d[i]) ∧ C_L(X_d[i], K_2, Y_d[i]);
8:         i ← i+1;
9:     K* ← sat_assignment_{K_1}(F_i);
```

The main purpose of the SAT attack was to break the primitive logic obfuscation techniques, including RLL [16], SLL [18], and FLL [37]. For all these logic locking countermeasure solutions, the SAT attack was able to rule out a significant number of key values at each iteration (by finding each DIP), and it was able to break them within a few iterations/minutes.

2.6 Stage 3: Post-SAT Attacks

In order to thwart the SAT attack, the first attempted approach was to weaken the strength of the DIPs to reduce its pruning power. Reducing the pruning power of DIP means that the found DIP can rule out a less portion of the incorrect keys (the best case is one incorrect key) per each iteration. SARLock [22] and Anti-SAT [21] were the first prior art logic locking countermeasures that focused on the reduction of DIPs' pruning power. Both SARLock and Anti-SAT engaged one-point flipping function, demonstrated in Fig. 2.8. When one of these countermeasures is applied to a circuit, during the SAT invocation, each DIP is able to rule out only one incorrect key. Hence, the SAT attack requires finding all $2^{|KI|} - 1$ incorrect keys one by one. Hence, it makes the logic locking exponentially hard for the SAT



Y_O	IN	k=0	k=1	k=2	k=3	k=4	k=5	k=6	k=7
✓	0	✓	✓	✓	✓	✗	✓	✓	✓
✓	1	✗	✓	✓	✓	✓	✓	✓	✓
✓	2	✓	✓	✗	✓	✓	✓	✓	✓
✓	3	✓	✓	✓	✓	✓	✗	✓	✓
✓	4	✓	✓	✓	✓	✓	✓	✓	✓
✓	5	✓	✓	✓	✓	✓	✓	✗	✓
✓	6	✓	✓	✓	✓	✓	✓	✓	✗
✓	7	✓	✓	✓	✗	✓	✓	✓	✓

Figure 2.8: Flipping Structure of SARLock and Anti-SAT.

attack with respect to the number of key bits $|KI|$.

Similar to the point-function techniques, many other approaches have been proposed after the introduction of the SAT attack to show how we can make robust logic locking techniques against the SAT attack. However, many of the state-of-the-art logic locking countermeasures still suffer from big shortcomings, thus resulting in the introduction of the newer attack leading to break them. In the following, we will evaluate these countermeasures, such as point function techniques, and the attacks on them.

2.6.1 Removal Attack

Point function techniques, such as SARLock [22] and Anti-SAT [21], suffer from big structural and functional shortcomings. One big structural shortcoming of them is that, as shown in Fig 2.8, in the implementation of one-point flipping circuit, the locking circuitry is completely decoupled from the original circuit. A removal attack identifies and removes/bypasses the locking circuitry to retrieve the original circuit and to remove dependence on key values [24]. The removal attack, presented in [24], was used to detect and remove SARLock [22]. However, the distinguishing of locking circuitry in Anti-SAT [21] is not as straightforward as that of SARLock, and removal attack fails to break such countermeasures that are hard to be detected (preventing removal by pure structural analysis).

2.6.2 Signal Probability Skew (SPS) Attack

The Signal Probability Skew (SPS) attack [24] leverages the structural traces in Anti-SAT block to identify and isolate the Anti-SAT block [21]. *Signal probability skew* (SPS) of a signal x is defined as $s = P_r[x = 1] - 0.5$, where $P_r[x = 1]$ indicates the probability that signal x is 1. The range of s is $[-0.5, 0.5]$. If the SPS of signal x is closer to zero, an attacker has a lower chance of guessing the signal value by random. For a 2-input gate, the signal probability skew is the difference between the signal probability of its input wires. The flipping-circuit in the Anti-SAT is constructed using two complementary circuits, g and $\overline{g}$, in which the number of input vectors that make the function g equal to 1 (p) is either close to 1 or $2^n - 1$. These two complementary circuits converge at an AND gate G. Considering this structure, the *absolute difference of the signal probability skew* (ADS) of the inputs of gate G is quite large, noting that the SAT resilience is ensured by this skewed p. Algorithm 2 shows the SPS attack, which identifies the Anti-SAT block's output by computing signal probabilities and searching for the skew(s) of arriving signals to a gate in a given netlist.

Algorithm 2 SPS Attack Algorithm [24]

```
1: function SPS_Attack(Circuit C_L)
2:     ADS_arr ← {};
3:     for each gate ∈ C_L do
4:         ADS_arr(gate_i) ← Compute_ADS(C_L, gate_i);
5:     G ← Find_Maximum(ADS_arr);
6:     Y ← Find_value_from_skew(G);          ▷ Correct value of Anti_SAT output
7:     C_Lock ← remove_TFI(C_L, G, Y);       ▷ Transitive FanIn of the gate G
8:     return C_Lock                         ▷ C_Lock: C_L after removing Anti_SAT block
```

2.6.3 Bypass Attack

Although SARLock and Anti-SAT break the SAT attack, they also suffer from another functional shortcoming in which the output corruption upon application of a wrong key is quite low. As demonstrated in Fig. 2.8, for each incorrect key, only one input pattern corrupts the PO. Hence, the corruptibility of such countermeasures is very low. Observing

and relying on the very low level of output corruption in such SAT-hard obfuscation solutions, the bypass attack [25] was introduced. The bypass attack instantiates two copies of the obfuscated netlist using two randomly selected keys, and builds a miter circuit that evaluates to 1 only when the output of two circuits is different. The miter circuit is then fed to a SAT solver looking for such inputs. The SAT returns with minimum of two inputs for which the outputs are different. These input patterns are tested using an activated IC (golden IC) validating the correct output. Then a bypass circuit is constructed using a comparator that is stitched to the primary output of the netlist which is unlocked using the selected random key, to retrieve the correct functionality if that input pattern is applied. The Bypass attack works well when the SAT-hard solution is not mixed with the traditional logic locking mechanism since its overhead increases very quickly as output corruption of logic locking increases. This observation motivated researchers to look at possibilities of approximate attacks to retrieve the key values associated with non SAT-hard obfuscation solutions that are mixed with SAT-hard solutions.

2.6.4 AppSAT Attack

The corruptibility shortcoming of the SARLock and Anti-SAT could be addressed by engaging and integrating primitive logic locking techniques, such as RLL, SLL, or FLL, with them. Such primitive logic locking solutions exhibit a high level of output corruption. Hence, with integrating point-function techniques with primitive techniques, the resulting approach that is known as compound logic locking techniques, could get the benefit of both of them with high corruptibility acquired from primitive techniques as well as SAT robustness acquired from the point function techniques. However, such compound solutions suffer from a newer attack in which approximation has been used for de-obfuscation. So far, defense solutions to mitigate the SAT attack, are based on the assumption that the attacker needs an exact attack on logic locking, in which the exact correct key is recovered. However, Shamsi *et al.* [27] proposed a new attack, called approximate SAT (AppSAT) which relax this constraint. AppSAT shown in Algorithm 3, is an approximate attack on logic locking based on

the SAT attack and random testing. The authors use *probably-approximate-correct* (PAC) model for formulating approximate learning problems. The exact SAT attack continues to find DIPs until no more DIPs can be found. However, the AppSAT will be terminated in any early step in which the error falls below the certain limit. If this condition happens, the key value will be considered as an approximate key with a specified error rate; otherwise, the random sampling that resulted in a disagreement will be added to a SAT formula as a new constraint. In AppSAT, heuristic methods for estimating the error is used for large functions, to avoid any computation complexity. In the approximate key extracted from the AppSAT on compound logic locking techniques, it is guaranteed that the key corresponded to the primitive technique is correct, and the error rate of the POs related to the key value corresponded to the point function technique will be smaller than the given threshold.

Algorithm 3 AppSAT Attack Algorithm [27]

```
1: function AppSAT_Attack(Circuit C_L, Circuit C_O)
2:     i ← 0; F_0 ← C_L(X, K_1, Y_1) ∧ C_L(X, K_2, Y_2);
3:     while SAT(F_i ∧ (Y_1 ≠ Y_2)) do
4:         X_d[i] ← sat_assignment (F_i∧(Y_1 ≠Y_2)); Y_d[i] ← C_O(X_d[i]);
5:         F_{i+1} ← F_i ∧ C_L(X_d[i], K_1, Y_d[i]) ∧ C_L(X_d[i], K_2, Y_d[i]); i ← i+1;
6:         every n rounds do
7:         for each (x ∈ Random Patterns) do
8:             if C_L(X, K_1, Y) ≠ C_O(X) then
9:                 FailedPatterns ← FailedPatterns + 1;
10:                F_{i+1} ← F_{i+1} ∧ (C_L(X, K_1, Y) = C_O(X)); i ← i+1;
11:        if error ¡ ErrorThreshold then
12:            return K_1 as an approximate key
13:    K* ← sat_assignment_{K_1}(F_i);
```

2.6.5 Double-DIP Attack

Double-DIP [39] is another approximate attack, shown in Algorithm 4. Double-DIP is an extension of SAT attack in which during each iteration, the discriminating input should eliminate at least two wrong keys. To illustrate its effectiveness, Double-DIP targets SARLock+SSL, representing a compound of SAT-hard and high output corruption obfuscation. When the Double-DIP attack terminates, the key of the traditional logic locking (SSL) is guaranteed to be correct. As a result, the compound logic locking will be reduced

to a single SAT attack resilient technique, which could then be attacked using a bypass attack.

Algorithm 4 Double-DIP Attack Algorithm [39]

1: **function** DoubleDIP_Attack(Circuit C_L, Circuit C_O)
2: $i \leftarrow 0$; $F_0 \leftarrow C_L(X, K_1, Y_1) \wedge C_L(X, K_2, Y_2) \wedge C_L(X, K_3, Y_1) \wedge C_L(X, K_4, Y_2)$;
3: **while** $SAT(F_i \wedge (Y_1 \neq Y_2)) \wedge (K_1 \neq K_3)) \wedge (K_2 \neq K_4))$ **do**
4: $X_d[i] \leftarrow$ sat_assignment $(F_i \wedge (Y_1 \neq Y_2)) \wedge (K_1 \neq K_3)) \wedge (K_2 \neq K_4))$;
5: $Y_d[i] \leftarrow C_O(X_d[i])$;
6: $F_{i+1} \leftarrow F_i \bigwedge_{j=1}^{4} C_L(X_d[i], K_j, Y_d[i])$; $i \leftarrow i+1$;
7: $K^* \leftarrow \text{sat\_assignment}_{K_1}(F_i)$;

2.6.6 Bit-Flipping Attack

The Bit-flipping attack [40] is yet another attack against compound logic locking schemes in which a SAT-hard solution such as SARLock is combined with a primitive logic locking to guarantee both high error rates and resilience to the SAT-based attack. In the Bit-flipping attack, the keys are first separated into two groups (k_1 and k_2) by counting DIPs for two keys with a hamming distance equal to one. The attack is motivated by the observation that in primitive logic locking, the wrong key causes a substantial wrong input-output pattern. However, the error rate of the bit-flipping function is usually very small. As shown in Algorithm 5, after separation of keys, this attack fixes SAT-resilient keys, k_2, as a random number, and uses a SAT solver to find the correct key values for k_1. After finding k_1, the bypass attack is applied to retrieve the original circuit.

2.6.7 AppSAT Guided Removal Attack

AppSAT Guided Removal (AGR) attack is another mechanism that targets compound logic locking, particularly Anti-SAT + traditional logic locking [24]. This attack integrates AppSAT with a removal-based simple structural analysis of the locked netlist. Unlike AppSAT, the AGR attack recovers the exact correct key. In this attack, first, the AppSAT is used to find the key of the primitive logic locking techniques. Then, AGR targets the remaining key bits belong to the SAT-resilient (point-function) logic locking, such as the

Algorithm 5 Bit-flipping Attack Algorithm [40]

```
1: function BitFlipping_Attack(Circuit C_L, Circuit C_O)
2:     for each j < Fixed-iteration do
3:         K_A ← a random key;
4:         for each bit b ∈ K_A do
5:             K_B ← K_A while bit b flipped;
6:             i ← 0; F_0 ← C_L(X, K_A, Y_A) ∧ C_L(X, K_B, Y_B);
7:             while SAT(F_i ∧ (Y_A ≠ Y_B)) do
8:                 X_d[i] ← sat_assignment (F_i∧(Y_A ≠Y_B));
9:                 F_{i+1} ← F_i ∧ (X ≠ X_d[i]); i ← i+1;
10:                if i > Threshold then
11:                    b is in K_1,
12:                    break;
           j ← j + 1;
13:    K_2 ← all key bits / K_1;          ▷ Seperation is Done. Then, fix K_2 as a random number.
14:    K_1 ← SAT_ATTACK (C_L, C_O);                          ▷ Find Traditional Keys using SAT.
15:    C*_L ← update_netlist(C_L — K_1)
16:    return (BYPASS_ATTACK(C*_L);
```

Anti-SAT block, through a simple structural analysis. As shown in Algorithm 6, after exploiting the AppSAT attack, in the post-processing steps, AGR finds the gate (G) at which most of the Anti-SAT key bits converge. AGR finds G by tracing the transitive fanout of the Anti-SAT key inputs, where all the Anti-SAT key bits converge. The ratio of key bits converging at each of the inputs of the gate G, are close to 0.5, which is shown as the *selected property* in line 7 of Algorithm 6. AGR identifies the candidates for gate G by checking this property for all gates in the circuit, and then sort these candidate based on the number of key inputs that converge at a gate and pick the gate G from all candidates, which has the most number of key inputs converge to that gate. Then the attacker re-synthesize the design with the constant value for the output of G gate and retrieving the correct functionality.

2.6.8 Sensitization Guided SAT Attack

While the one-point flipping circuit in Anti-SAT and SARLock are completely decoupled from the original netlist, Li *et al.* [41] proposed the AND-tree Insertion (ATI), as a SAT-resilient logic locking, which embeds AND trees inside the original netlist. It not only makes all aforementioned attacks less effective, but it also decreases the implementation

Algorithm 6 AGR Attack Algorithm [24]

```
1: function AGR_Attack(Circuit C_L, Circuit C_O)
2:     #Cand ← num_gates(C_L)
3:     while (#Cand > 1 and !Timeout) do
4:         AppSAT_Attack();                                   ▷ 4 times
5:         Candidates ← {};
6:         for each gate ∈ C_L do
7:             if gate_i has the selected property then
8:                 Candidates ← Candidates + 1;
9:     G ← Find_Max_key_count(Candidates);
10:    C_Lock ← remove_TFI(C_L, G);          ▷ remove Transitive FanIn of the gate G
11:    return C_Lock;                        ▷ C_Lock: C_L after removing Anti_SAT block
```

overhead. Additionally, the inputs of the AND-tree are camouflaged by inserting INV/BUF camouflaged gates, which can be replaced with the XOR/XNOR gates in order to lock the AND-tree. However, this defense was broken by a new attack that was coined as Sensitization Guided SAT (SGS) attack [24]. The SGS attack is carried out in two stages: (1) *sensitization* that exploits bias in input patterns which allows an attacker to apply only a subset of DIPs, i.e., those that bring unique values to the AND-tree inputs. (2) *SAT attack* using the patterns discovered in the first stage.

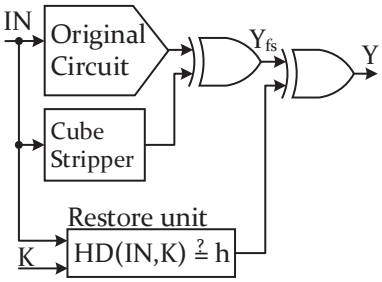
2.6.9 Functional Analysis Attack

Aiming to provide a defense that resists all previously formulated attacks led to the introduction of Stripped-Functionality Logic Locking (SFLL) [23]. In SFLL the original circuit is modified for at-least one input pattern (cube) using a *cube stripping unit*, demonstrated in Fig. 2.9. As shown, Y_{fs} is the output of the stripped circuit, in which the output corresponding to at least one input pattern is flipped. The restore unit not only generates the flip signal for one input pattern per each wrong key, but it also restores the stripped output, (e.g. $IN = 4$ in Fig. 2.9) to recover the correct functionality on Y. Note that applying removal attack on restore unit recovers Y_{fs}, which is not the correct functionality. In addition, SFLL-HD is able to protect $\binom{k}{h}$ input patterns that are of Hamming Distance (HD) h from the k-bit secret key, and accordingly uses Hamming Distance checker as a restore unit (e.g. $h = 0$ in Fig. 2.9 is also called TTLock [42]).

Although SFLL was resilient against all previously formulated attacks, it was exploited using a newly formulated attack, called Functional Analysis on Logic Locking (FALL) attack [26]. In this attack model, the adversary is assumed to be a malicious foundry that knows the locking algorithm and its parameters, e.g. h in SFLL-HD. A FALL attack is carried out in three main stages and relies on structural and functional analysis to determine potential key values of a locked circuit. First, the FALL attack tries to find all nodes which are the results of comparing an input value with a key input. It is done by a comparator identification. Such nodes ($nodes_{RU}$), which contain these particular comparators, are very likely to be part of the functionality restoration unit. The set of all inputs that appear in these comparators, should be in the fan-in cone of the cube stripping unit. Then, it finds a set of all gates whose fan-in-cone is identical to the members of $nodes_{RU}$. This set of gates must contain the output of the cube stripping unit. Second, the attacker applies functional analysis on the candidate nodes suggested by and collected from the first stage to identify suspected key values. Broadly speaking, the attacker uses functional properties of the cube stripping function used in SFLL, to determines the values of the keys. Based on the author's view, this function has three specific properties. So, they have proposed three attack algorithms on SFLL, which exploit the unateness and Hamming distance properties of the cube stripping functions. The input of these algorithms is circuit node c, that computed from the first stage, and the algorithm checks if c behaves as a Hamming distance calculator in the cube stripping unit of SFLL-HD. If the attack is successful, the return value is the protected cube. Third, they have proposed a SAT-based key confirmation algorithm using a list of suspected key values and I/O oracle access, that verifies whether one of the suspected key values computed from the second stage, is correct.

2.6.10 CycSAT Attack

Considering the strength of all previously formulated attacks, some of the researchers started seeking solutions that fundamentally violated the assumptions of these attacks with respect to the nature of locked circuits. One of such attempts was the introduction of cyclic logic



Y_{fs}	IN	k=0	k=1	k=2	k=3	k=4	k=5	k=6	k=7
✓	0	✓	✓	✓	✓	✗	✓	✓	✓
✓	1	✗	✓	✓	✓	✓	✓	✓	✓
✓	2	✓	✓	✗	✓	✓	✓	✓	✓
✓	3	✓	✓	✓	✓	✓	✗	✓	✓
✗	4	✗	✓	✗	✗	✗	✗	✗	✗
✓	5	✓	✓	✓	✓	✓	✓	✗	✓
✓	6	✓	✓	✓	✓	✓	✓	✓	✗
✓	7	✓	✓	✓	✗	✓	✓	✓	✓

Figure 2.9: SFLL-HD while $h = 0$.

locking [28,43], was first proposed in [43]. In this obfuscation technique, as demonstrated in Fig. 2.10, combinational cycles are added and each deliberately established cycle is designed to have more than one way to open. The requirement for having more than one way to open each cycle assures that even if the original netlist has no cycle by itself, the cycles remain irreducible by means of structural analysis. The cyclic obfuscation resulted in obfuscation with a high level of output corruption, while it was able to break the SAT attack either by 1) trapping it in an infinite loop, or 2) forcing it to exit with a wrong key depending on whether the introduced cycles make the circuit stateful or oscillating.

The promise of secure cyclic obfuscation was shortly after broken by CycSAT attack [44], whose algorithm is demonstrated in Algorithm 7. In CycSAT, the key combinations that result in the formation of cycles are found in a pre-processing step. These conditions

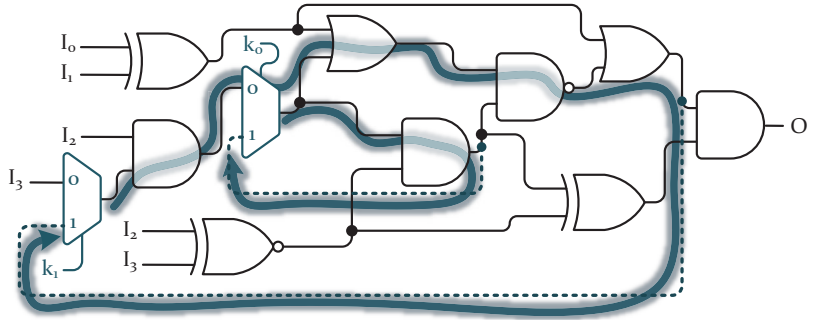


Figure 2.10: An Example of Cyclic Obfuscation using 2-to-1 MUXes.

are then translated into problem augmenting conjunctive normal forms (CNF) formulas, denoted as cycle avoidance clauses, the satisfaction of which guarantees no cycle in the netlist. The cycle avoidance clauses are then added to the original SAT circuit CNF and the SAT attack is executed. The validity of this attack, however, was challenged in [28], as researchers illustrated that the pre-processing time for CycSAT attack is linearly dependent on the number of cycles in the netlist. Hence, by building an exponential relation between the number of feedback and the number of cycles in the design, the pre-processing step of CycSAT will face exponential runtime.

Algorithm 7 CycSAT Attack on Cyclic Locked Circuits [43]

function CycSAT_Attack(Circuit C_L, Circuit C_O)
  $W = (w_0, w_1, ...w_m) \leftarrow$ FindFeedback(C_L);
  for each $(w_i \in W)$ **do**
    $F(w_i, w_i') \leftarrow$ no_ structural_ path(w_i);
  $i \leftarrow 0$; $NC(K) = \wedge_{i=0}^{m} F(w_i, w_i')$
  $C_L^*(X, K, Y) \leftarrow C_L(X, K, Y) \wedge NC(K)$; $F_0 \leftarrow C_L^*(X, K_1, Y_1) \wedge C_L^*(X, K_2, Y_2)$;
  while $SAT(F_i \wedge (Y_1 \neq Y_2))$ **do**
    $X_d[i] \leftarrow$ sat_assignment $(F_i \wedge (Y_1 \neq Y_2))$;
    $Y_d[i] \leftarrow C_O(X_d[i])$;
    $F_{i+1} \leftarrow F_i \wedge C_L(X_d[i], K_1, Y_d[i]) \wedge C_L(X_d[i], K_2, Y_d[i])$;
    $i \leftarrow i+1$;
  $K^* \leftarrow$ sat_assignment$_{K_1}(F_i)$;

2.6.11 Advanced Cyclic SAT Attacks

Inability to analyze all cycles in the prepossessing step of CycSAT 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. BeSAT [45] remedies this shortcoming by augmenting the CycSAT attack with a run-time behavioral analysis. As shown in Algorithm 8, 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.

Although BeSAT resolves some shortcomings of CycSAT, its key restriction rule can face big-size cycle formulas even in moderate-size circuits. Another advanced cyclic-based

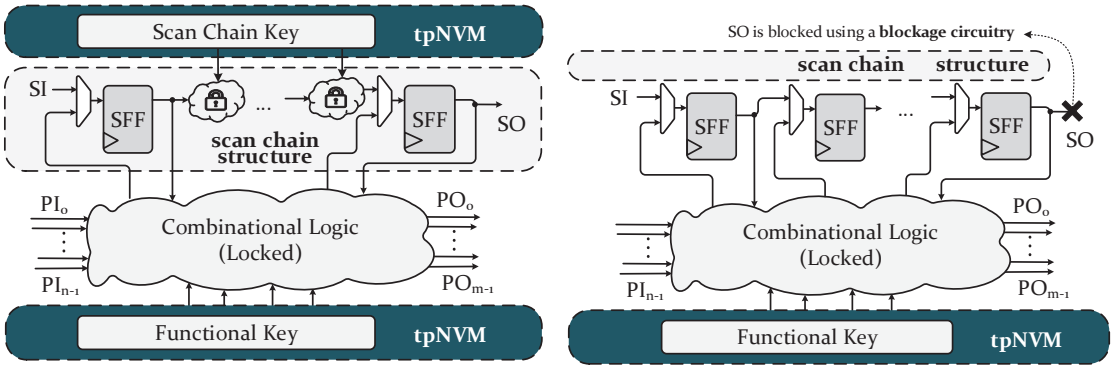
Algorithm 8 BeSAT Attack on Cyclic Locked Circuits [45]

function BeSAT_Attack(Circuit C_L, Circuit C_O)
  $W = (w_0, w_1, ...w_m) \leftarrow$ FindFeedback(C_L);
  for each ($w_i \in W$) **do**
    $F(w_i, w_i') \leftarrow$ no_ structural_ path(w_i);
  $i \leftarrow 0$; $NC(K) = \wedge_{i=0}^{m} F(w_i, w_i')$
  $C_L^*(X, K, Y) \leftarrow C_L(X, K, Y) \wedge NC(K)$; $F_0 \leftarrow C_L^*(X, K_1, Y_1) \wedge C_L^*(X, K_2, Y_2)$;
  while $SAT(F_i \wedge (Y_1 \neq Y_2))$ **do**
    $X_d[i] \leftarrow$ sat_assignment ($F_i \wedge (Y_1 \neq Y_2)$); $Y_d[i] \leftarrow C_O(X_d[i])$;
    $F_{i+1} \leftarrow F_i \wedge C_L(X_d[i], K_1, Y_d[i]) \wedge C_L(X_d[i], K_2, Y_d[i])$;
    if ($X_d[i]$ in DIP) **and** ($C_L(X_d[i], K_1) \neq Y_d[i]$)) **then**
      $F_{i+1} \leftarrow F_{i+1} \wedge (K_1 \neq \hat{K}_1) \wedge (K_2 \neq \hat{K}_1)$;
    else if ($X_d[i]$ in DIP) **and** ($C_L(X_d[i], K_2) \neq Y_d[i]$) **then**
      $F_{i+1} \leftarrow F_{i+1} \wedge (K_1 \neq \hat{K}_2) \wedge (K_2 \neq \hat{K}_2)$;
    $i \leftarrow i+1$;
  while $SAT_{K_1}(F_i)$ **do** ▷ Correct Key: $\hat{K}_c$
    if Ternary_SAT(F_i, K_c) **then**
      $F_i \leftarrow F_i \wedge (K_1 \neq \hat{K}_c)$
    else
      $K^* \leftarrow \hat{K}_c$; **break**;

SAT attack that focuses on the shortcomings of CycSAT and BeSAT is icySAT [46]. They propose an algorithm that can produce non-cyclic conditions in polynomial time w.r.t. the size of the circuit, avoiding the potentially exponential runtime of BeSAT. Also, icySAT improves the attacks on cyclic logic locking techniques for cases whether the original circuit is cyclic, or if the feedback dependencies are re-convergent, or whenever the types of the cycles are oscillating.

2.6.12 Sequential SAT Attack: Unrolling/BMC

Since the availability of the scan chain structure undermines the robustness of many logic locking techniques, there exist other sub-groups in logic locking techniques, such as scan-based logic locking, scan blockage, and sequential logic locking, in which the availability of the scan chain is restricted/blocked [47–52]. In such scenarios, assuming that the oracle is still available, however as demonstrated in Fig. 2.11, the adversary will lose access to the scan chain structure and the access would be limited to PI/PO of the oracle. Therefore, all of the previously discussed de-obfuscation attacks will fail to evaluate and break the locked



(a) Scan Locking (b) Scan Blockage

Figure 2.11: Limiting the Access to the Scan Chain Structure.

circuits with limited scan chain access. However, further studies on logic locking show that restricting access still cannot guarantee robustness against state-of-the-art threats.

The preliminary version of the SAT-based sequential de-obfuscation attack was first introduced in [53]. The sequential SAT attack shows how the SAT solver could still be engaged to break the logic locked circuits with limited scan chain access. The sequential SAT attack uses an iterative method to prune the search space, similar to the SAT attack. Due to the limited access to internal registers, instead of seeking a DIP in each iteration, it instead finds a sequence of input patterns X denoted as discriminating input *sequence* (DIS) that can produce two separate outputs for two different keys. To build the sequence using the SAT attack, the sequential SAT gets the benefit of unrolling/unfolding to create the combinational equivalent circuit. Then, the SAT solver will be used for a specific depth to generate the DIS. For instance, Fig. 2.12, shows a sequential circuit unrolled for τ clock cycles. In this case, the SAT could be invoked to return a DIS with the length of τ. Fig. 2.13 shows the main steps of the combinational SAT attack and the SAT-based sequential de-obfuscation attack, in which unrolling determines the length of DISes.

A query with a bounded depth could also be done using a bounded model checker (BMC). So, in order to accomplish the unrolling step, with determining the boundary, the

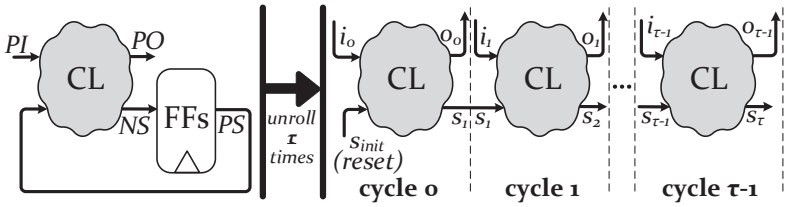
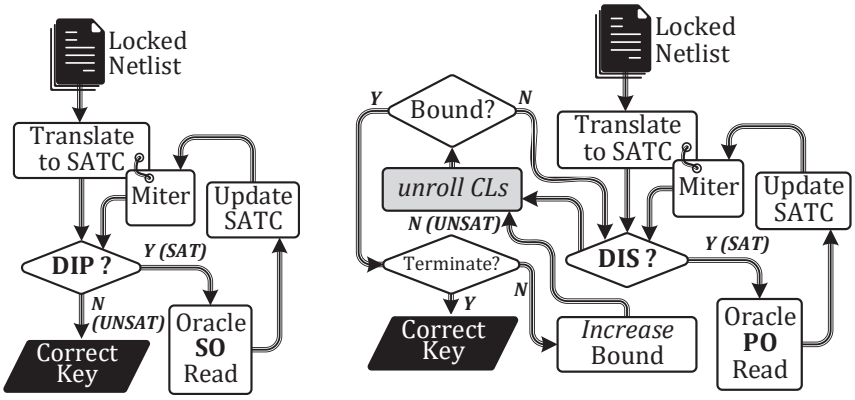


Figure 2.12: Unrolling the Sequential Circuit for τ Cycles.



(a) Combinational SAT (b) Sequential SAT

Figure 2.13: Combinational SAT vs. Sequential SAT Attack.

BMC engine could be invoked to model the locked circuit as an FSM, and the specification could be formalized by temporal logic properties. So, the BMC could be exploited as an alternative approach to do the symbolic model checking before invoking the SAT procedure. Algorithm 9 demonstrates the overall procedure of the primitive sequential SAT attack, once the unrolling is done using the BMC engine. $C(X, K, Y)$ indicates the locked circuit generating output sequence Y using input sequence X and the key value K, and $CBlackBox(X)$ refers to the output sequence of the oracle for the same input sequence. After building the model from the locked circuit, the attack instantiates a BMC to find the X_{DIS}. Then, the model would be updated with a new constraint to guarantee that the next pair of keys, that will be discovered in the subsequent attack iterations, produce the same output for previously discovered X_{DIS}. The iterations continue until no further X_{DIS} is found within the boundary of b. After reaching the boundary, if the algorithm passes three

criteria, the key could be found with one more SAT instantiation. The boundary could be extended if termination conditions are not met. The primitive sequential SAT attack in [53] specified three main termination conditions for this attack:

Unique Completion (UC): This condition verifies that the key generated by the algorithm is unique. The attack is successfully ended, and the key is the correct one, if there is only one key that meets all previous DISes.

Combinational Equivalence (CE): If there is more than one key for all previously found DISes (non-unique key), the attack checks the combinational equivalency of the remaining keys. In this step, the input/output of FFs are considered as pseudo PO/PI allowing the attacker to treat the circuit as combinational. The resulting circuit is subjected to a SAT attack, and if the SAT solver fails to find a different output or next state for two different keys, it concludes that all remaining keys are correct and the attack terminates successfully.

Unbounded Model Check (UMC): If both UC and CE fail, the attack checks the existence of a DIS for the remaining keys using an unbounded model checker. This is an exhaustive search with no limitation on bound (or the number of unrolls). If no DIS is discovered, the existing set of DIS is a complete set, and the attack terminates. Otherwise, the bound is increased and previous steps are repeated.

Algorithm 9 Sequential Attack on Obfuscated Circuits [53]

1: $b = initial\_boundary$, $Terminated = False$;
2: $Model = C(X, K_1, Y_1) \wedge C(X, K_2, Y_2) \wedge (Y_1 \neq Y_2)$;
3: **while** not $Terminated$ **do**
4: **while** $(X_{DIS}, K_1, K_2) \leftarrow BMC(Model, b) = T$ **do**
5: $Y_f \leftarrow C_{BlackBox}(X_{DI})$;
6: $Model = \wedge\, C(X_{DIS}, K_1, Y_f) \wedge C(X_{DIS}, K_2, Y_f)$;
7: **if** $\mathbf{UC}(Model, b) \vee \mathbf{CE}(Model, b) \vee \mathbf{UMC}(Model)$ **then**
8: $Terminated$;
9: $b = b + boundary\_step$;

2.6.13 KC2 Attack

Although the primitive unrolling/BMC-based sequential SAT attack formulates logic locking techniques with no access to scan chain structure, it runs into the scalability issues as it relies on two sub-routines which are in **PSPACE** and **NP**, thereby, failing to terminate for even moderately small circuits, which contain only a few thousand gates. To mitigate the scalability issue of the primitive sequential SAT attack, Shamsi *et al.* propose a fast sequential de-obfuscation attack, called KC2 [54]. As demonstrated in Fig. 2.13, the SAT circuit (SATC) requires an update per each iteration, in which the new learned clauses will be added to the list of previously found clauses. Hence, with more iteration, the size of the SAT problem will be increased drastically. In sequential SAT, it gets worse because the SAT problem will be expanded in two dimensions, i.e. iterations and unrolling. Hence, to avoid the unnecessary piling up of the clauses in the SATC, KC2 implements the following dynamic optimization tweaks in the attack procedure, resulting in the improvement and acceleration of the primitive BMC-based sequential SAT:

(1) **Incremental SAT-Solving**: It determines *when* SAT instance requires to be generated for solving, and by using the BMC engine, the same SAT solver instance can be used for various tasks, e.g. BMC calls, termination check, simplification, and even unbounded checking.

(2) **Key-Condition Sweeping**: It helps detecting equivalent nodes in a circuit and allows merging them. It could be accomplished using different sweeping techniques, such as BDD-sweeping, cut-sweeping, and SAT-sweeping [55].

(3) **Key-Condition to BDD Conversion**: It helps to keep the canonical information corresponded to the key small, even while there exists large conjunction of circuit copies, especially for deep DISes.

(4) **Negative Key-Condition Compression**: It helps keeping track of disqualified keys. By representing each key-disqualifying as a clause, the solver could be reconstructed (simplified) to this condition when the algorithm detects a build up of clauses in the solver.

(5) **Termination Conditions**: Since **CE** is a stronger condition, **UC** could be skipped.

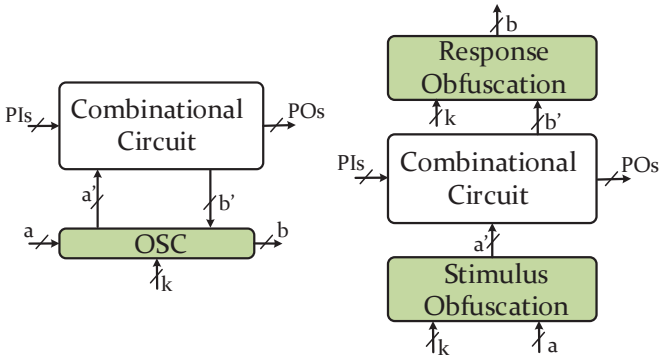
Also, rare invocation of **UMC** could be integrated with interpolation-based model checking (IMC) for faster termination.

Although KC2 integrates these tweaks to overcome the scalability and performance issue of the primitive sequential SAT attack, their experimental results still show that when locking is added in even medium-size circuits, such as large ISCAS-89 circuits with few thousands of gates, KC2 fails to break them.

2.6.14 ScanSAT on Both Static and Dynamic Scan Locking

ScanSAT aims to break the scan-based logic locking techniques. In scan-based logic locking, as demonstrated in Fig. 2.11(a), since the availability of scan is *locked*, the adversary has only access to the PI/PO. Hence, similar to the primitive sequential SAT attack and KC2, ScanSAT [3] is based on the fact that the complex τ-cycle transformation of scan-based locking could be modeled by generating an unfolded/unrolled combinational equivalent counterpart of the scan-locked circuit. In this case, the locking parts added into the scan path become part of the resultant combinational circuit. Hence, the adversary faces a combinational (unrolled) locked circuit with key gates at the pseudo-primary I/Os of the circuit. The combinational equivalent of a locked circuit in Fig. 2.14(a) is provided in Fig. 2.14(b), where the locking on the stimulus and the response are modeled separately as combinational blocks driven by the same scan locking key. In general, ScanSAT models the locked scan chains as a logic-locked combinational circuit, paving the way for the application of the combinational SAT attack to reveal the key (sequence of unrolled), unlocking the scan chains, and thus, restoring access to the oracle.

In addition to de-obfuscating the statically scan-locked circuit, ScanSAT is also able to be applied on dynamically scan-locked circuit that is locked by DOS architecture [56]. In DOS architecture, a LFSR has been engaged to generate runtime dynamic keys. In oracle-guided attacks, since the output of the netlist are continuously checked with the oracle, dynamically changing the key in the oracle will corrupt this oracle-oriented evaluation, and results in failure of such attacks. In ScanSAT, it is assumed that after successfully



(a) Locked Scan Chain (b) Combinational Equivalent Model

Figure 2.14: Converting a Locked Scan Chain to its Combinational Counterpart [3].

reverse engineering, the LFSR structure, and consequently its polynomial are known to the adversary. Hence, finding the seed of LFSR and the update frequency parameter (p as the time interval of updating the key based on the LFSR output), that is the only secret in DOS architecture, would lead to deriving all the keys that are dynamically generated on the chip.

A simple method to identify p is to apply the same stimulus pattern repeatedly from the SI, and observe the response through the SO. The point is that after p capture operations, by repeatedly applying the same stimulus, the response would be different because of the updated key; thus, most likely, there will be a noticeable change in the observed response, helping detect the update operation on the key.

After finding p, the same approach that was used for static scan obfuscation would be used in this case. The difference now is that the SAT attack could be executed for at most p iterations (after p iterations, the key is updated). If more than p DIPs are required to identify a dynamic key, the SAT attack needs to be terminated prematurely upon p DIPs. Another SAT attack must be executed subsequently to identify the next dynamic key in the sequence still within p iteration. Since the updated key is generated by the LFSR whose polynomial is known for the adversary, independent SAT attack runs on each dynamic key reveals partial information of the seed; thus, the information from independent SAT

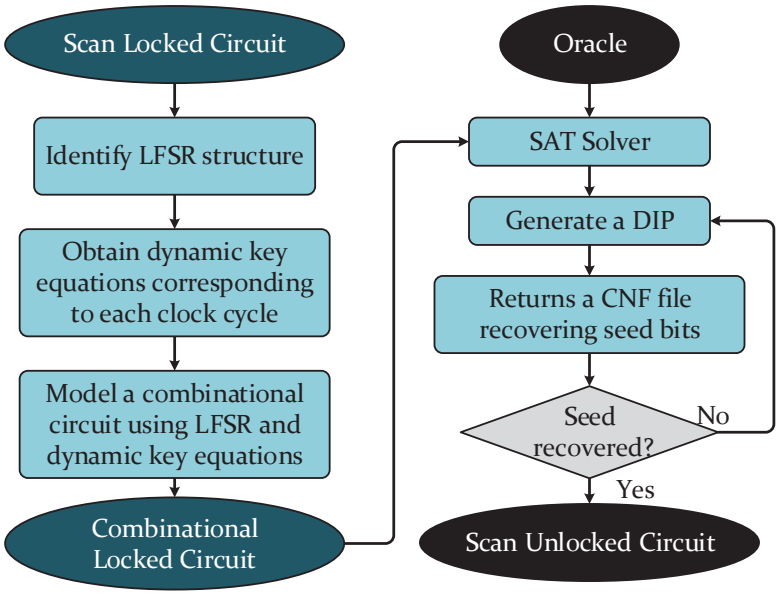


Figure 2.15: Flowchart for the DynUnlock Attack [4].

attack runs by gradually gathering information about the seed in every run, and finally by incorporating into the ScanSAT model, the relationship between the seed and the keys would be revealed.

2.6.15 DynUnlock: Yet Another Attack on Dynamic Scan Locking

As a countermeasure against ScanSAT attack, dynamic encrypt flip-flop (EFF-Dyn) [57] combines scan locking approach from EFF [47] and a PRNG, to introduce dynamicity in the design. In EFF-Dyn, based on the value of scan controlling signal, i.e. scan enable (SE), the source of the key to the circuit would be changed. In the test mode, the test key must be provided externally, and in case of a mismatch with the locking key embedded in the circuit, there exists a PRNG that updates the key in every clock cycle controls the key gates dynamically. However, similar to LFSR, the structure of PRNG and its polynomial would be known for the adversary after successfully reverse engineering. Hence, DynUnlock [4] proposes a similar approach to scanSAT to find the seed of the PRNG in EFF-Dyn.

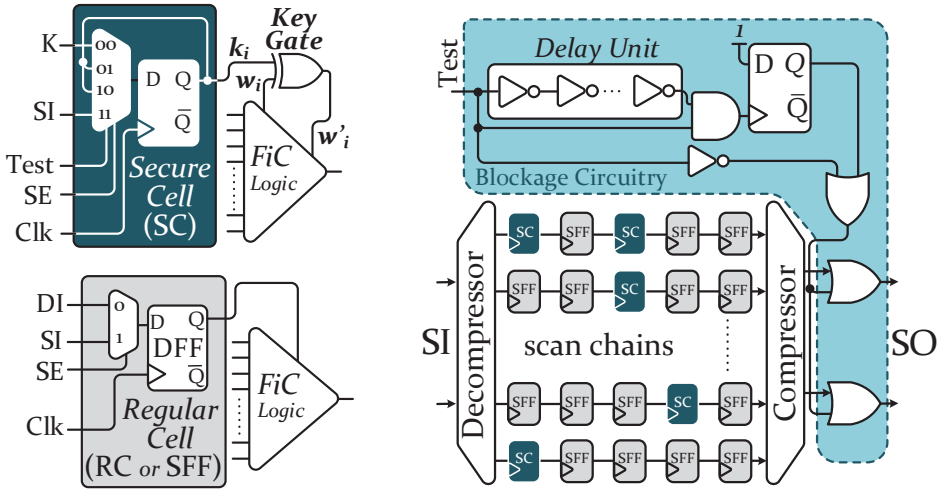
Assuming that the structure of PRNG is similar to an LFSR, in DynUnlock [4], as

demonstrated in Fig. 2.15, it first starts by reverse-engineering the LFSR circuit and obtaining the equations corresponding to each clock cycle. Next, it determines the location of key gates inserted between the SFFs. Then it models this sequential logic circuit into a combinational circuit with SFFs replaced with inputs and outputs. Once modeling is complete the combinational obfuscated counterpart circuit, with seed bits acting as primary key inputs, is fed to a SAT solver, which provides a DIP and its corresponding output pattern. In [4], the authors carry out the attack for just one capture cycle. To recover more bits, they restart the LFSR circuit and obtain a new DIP and its corresponding output pattern from the SAT solver, and recover more seed bits. They repeat the restart step until all the seed bits have been recovered, or the remaining seed bits can be brute-forced.

2.6.16 Shift-and-Leak Attack

Due to the failure of scan obfuscation architectures against sequential SAT and scanSAT attacks, more recent studies evaluate and reveal the effectiveness of the scan chain blockage after activation of the obfuscated circuit. The first scan blockage architecture, called robust design-for-security (R-DFS), was first introduced in [5], in which the obfuscation key is stored in a custom-designed scan (storage) cell, called secure cell as demonstrated in Fig. 2.16. In this new customized cell, based on the value of the SE pin and the new pin called $Test$, the key values could be loaded into SCs either directly from tpNVM or through SI, and the SO will be blocked after activation to avoid any secret leakage. However, the shift-and-leak attack [48] breaks the R-DFS by exploiting the availability of the shift-in process through the SI, and the capability of reading out the PO through chip pin-outs in the functional mode.

To remedy the leakage issue, the authors in [48] proposed modification to the R-DFS (mR-DFS), which 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. However, the work in [49] illustrates how a glitch-based shift-and-leak attack allows an adversary to still leak the logic obfuscation key through the PO even if the shift operation is disabled.



(a) Secure Cell (SC) vs. Regular Cell (SFF)

(b) Restricted Unauthorized Scan Access using Blockage Circuitry

Figure 2.16: Scan Blockage in R-DFS [5].

2.7 Robust Logic Locking Solutions

In pursuit of obfuscation schemes that could not be attacked by SAT-motivated attackers, some researchers tried to extend the locking mechanism to aspects of a circuit's function that cannot be translated to CNF. For example, Xie *et al.* [31] proposed a timing obfuscation scheme, denoted as delay logic locking (DLL). The Goal of the DLL obfuscation scheme is to introduce setup time and hold time violation if the correct key is not applied. In this case, the obfuscation 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 obfuscation.

In addition, few very recent research papers have focused on increasing the execution time of each SAT iteration rather than the total execution time, such as Cross-Lock and Full-Lock [32, 33]. The Full-Lock in [33] is argued that the strength of SAT solvers comes from their *Conflict-Driven Clause Learning* (CDCL) ability, which is resulted by recursively calling *Davis-Putnam-Logemann-Loveland* (DPLL) algorithm. Hence, the Full-Lock creates

Table 2.2: Comparison of State-of-the-art Logic Obfuscation Techniques.

Category	Defense	Attacked by
Pre-SAT	Random-based (RLL) [16]	Sensitization [18], SAT [1]
	Fault-based (FLL) [37]	SAT [1]
	Sensitization-based (SLL) [18]	SAT [1]
Point Function	SARLock [22]	SPS+Removal [24], Bypass [25], Bit-Flipping [40], AGR [24]
	Anti-SAT [21]	SPS+Removal [24], Bypass [25], Bit-Flipping [40], AGR [24]
	SFLL [23]	FALL [26]
	+ Primitive (Compound)	AppSAT [27], Double-DIP [39], Bit-Flipping [40], AGR [24]
Cyclic and Behavioral	Cyclic [43]	CycSAT [44], BeSAT [45], icySAT [46]
	SRCLock [28]	✗ (NO Attack)
	DLL [31]	✗ (NO Attack)
Routing Obfuscation	Cross-lock [32]	✗ (NO Attack)
	Full-Lock [33]	✗ (NO Attack)
Restricting Scan Chain	EFF+RLL [47]	Sequential SAT [53, 54], ScanSAT [3, 4]
	R-DFS+SLL [5]	Sequential SAT [53, 54], Shift&Leak [48, 49]
	MSSD+RLL [48]	Shift&Leak [48, 49]
	DynScan+SLL [57]	DynUnlock [3, 4]

an obfuscation method that results in very deep recursive call trees. They argue that the SAT attack execution time can be expressed by formula 2.1, in which N denotes the number of iterations (DIPs) of the SAT attack, $T_{DPLL}(\Phi)$ is the execution time of recursive calls for DPLL algorithm on CNF Φ, and t is the execution time of the remaining bookkeeping code executed at each iteration.

$$T_{Attack} = \sum_{i=1}^{N} T(i) = \sum_{i=1}^{N} (t + T_{DPLL}(\Phi)) = \sum_{i=1}^{N} \sum_{j=1}^{M} (T_{DPLL}^{Avg}) \simeq MN \times T_{DPLL}^{Avg} \tag{2.1}$$

Authors argue that M in formula 2.1 denotes the number of recursive DPLL calls. In Full-Lock [33], by observing the SAT hardness of CNF formulas produced using a fixed-length CNF constructor [58], it is proved that the CNF related to a routing obfuscation technique is a member of medium-length CNFs that maximizes the number (M) and the computational complexity (T_{DPLL}^{Avg}) of recursive DPLL calls. Hence, as a new routing obfuscation technique, the key-programmable routing block has been built using logarithmic networks in Full-Lock [33]. The strong aspect of this alternative solution is that (1) the

problems posed at each iteration of SAT attack is a SAT-hard problem, (2) the output corruption of this methods is significantly higher than obfuscating solution relying on increasing the N, (3) it is not susceptible to SPS, removal, bypass, and approximate attack.

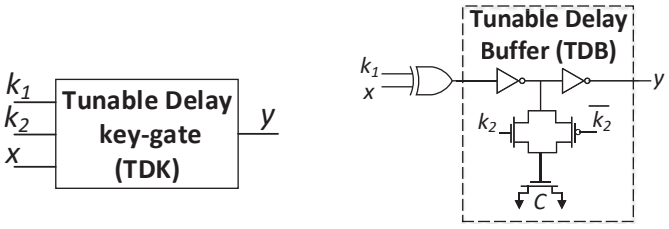
Table 2.2 shows the effectiveness of obfuscation solutions against the attacks discussed in this Chapter. As illustrated, the SRCLock [28], DLL[31], Full-Lock [33], and Cross-lock [32] are resistant to all attacks discussed so far, which motivates us to propose more powerful attacks, which will be discussed in Chapter 3 and Chapter 4 respectively.

Chapter 3: SMT Attack: Next Generation Attack on Obfuscated Circuits

As discussed in Chapter 2 and illustrated in Fig. 1.2 in Chapter 1, after introduction of the SAT attack in 2015 [1, 2], different studies have evaluated various mechanisms for building SAT-hard obfuscation solutions. However, the logic-based obfuscation schemes that rely on extending the Boolean behavior of a circuit can be broken by at least one of the state-of-the-art attacks, including SAT, SPS, removal, bypass, and AppSAT [1, 2, 24, 25, 27, 39]. Hence, recent studies have focused on obfuscation schemes that fundamentally violate the assumptions of these attacks with respect to the nature of the obfuscated circuit, or use non-logical properties of a netlist to obfuscate its behavior [28, 31, 43].

3.1 SAT Attack Failure in Representing Behavioral Attributes

A SAT attack works perfectly fine if the logic obfuscation 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. But in the case of Behavioral logic obfuscation, 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 [31], cyclic-based obfuscation presented in [43], and SRCLock [28] are good instances of such locking mechanism. For the purpose of locking, DLL uses a tunable delay key-gate (TDK) which is illustrated in Fig. 3.1. TDK consists of a conventional key-gate (XOR/XNOR) with a tunable delay buffer (TDB). The capacitive load of the buffer is controlled by a transmission gate, where activating the transmission gate increases the wire load capacitance of the internal wire, resulting in a larger TDK propagation delay. Hence, the functionality and propagation delay of a TDK, both, depends on



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) **(b)** **(c)**

Figure 3.1: Overall Structure of Tunable Delay Key-Gate (TDK).

the value of its key-inputs.

In DLL, the TDK cells are used to control the *setup* and *hold* time violations such that only one sequence of activation keys guarantees that the circuit operates with no violation. To apply the DLL, a design is first altered such that most timing paths are balanced to be sensitive with respect to small changes in the path delay, such that a small variation in delay causes setup or hold violations. This is achieved by means of carefully engineering the clock skew, cell sizing, and V_{th} swapping. Then the TDK cells are inserted in the common portions of setup and hold critical paths, such that attempting to only fix setup causes hold violation, and attempting to fix hold causes setup violation with the exception of one sequence of correctly configured TDK keys that assures all timing paths meet both setup and hold check timing constraints. Considering that the delay is not a logical behavior, the TDK cell behavior could not be completely captured by CNF, hence the delay locking is not directly attackable by a SAT attack. In [31] it was illustrated that even a mixed integer linear programming (MILP) based attack has up to 39% timing violation ratio (TVR). However, as we will illustrate in this Chapter, by employing an SMT attack we could find the keys to this obfuscation problem in few minutes.

3.2 Proposed Attack: SMT Attack

In this Chapter, we construct an attack model based on Satisfiability Modulo Theory (SMT) solver, and illustrate that the capability of this attack goes far beyond that of SAT attacks. More precisely, with specific formulation, we illustrate that the SMT attack on obfuscated circuits is significantly faster and more efficient compared to the SAT attack on Boolean logic obfuscation. Additionally, it could be used to attack behavioral logic obfuscation schemes, which is not possible by a pure SAT-based attack. To illustrate the second point, we attack and break the timing-logic obfuscation scheme in [31], based on which we generalize and illustrate how other similar SMT attacks could be formulated.

3.3 SMT Attack Threat Model

The SMT attack is an oracle-guided attack, in which we assume that the attacker has the reverse-engineered but obfuscated netlist. In addition, the attacker is able to buy/steal the correctly unlocked (activated) IC from the open market/field. Consequently, the attacker can apply arbitrary input to activated IC and observe its corresponding output. Hence, the attacker can query the oracle with any stimuli *pi*, and observe its output *po*. The purpose of the attack is to find the key inputs, that make the obfuscated netlist equivalent to that of the unlocked netlist.

3.4 Basics of the SMT Solver

In this section, we first review the usage and capabilities of an SMT solver, and then we illustrate how the SMT solver could be used to form an SMT attack on obfuscated circuits regardless of obfuscation's reliance on logical or non-logical properties of a circuit.

3.4.1 SMT Solver Usage and Capabilities

A Satisfiability Modulo Theory (SMT) is used to solve a decision problem while honoring constraints that could be expressed using first-order theories such as equality, reasoning, arithmetic, graph-based deduction, etc. Hence, it could be considered as a solver for a broad set of problems that could be categorized as Constraint Satisfaction Problems (CSP), which is a superset of Boolean Satisfiability Problems (BSP) that are solvable by SAT solvers. Additionally, the ability to express theories such as inequality (e.g. $3x + y < z$) provides a much richer Application Programming Interface (API) to the end user to define a problem compared to that of a SAT solver.

In general, there are two different approaches for solving an SMT problem. The first approach is based on translating the problem into a Boolean SAT instances denoted by *Eager* approach; In this approach the existing Boolean SAT solvers are used as is. However, the SMT solver has to work a lot harder for solving some problems that are otherwise very obvious (e.g. for checking the equivalence of two 32-bit values). However, by deploying a theory solver, this could be achieved in no time. For this reason, many SMT solvers follow another approach which referred to as the *Lazy* Approach. The Lazy approach integrates the Boolean satisfiability solvers, which are based on the Davis-Putnam-Logemann-Loveland (DPLL) in modern SAT, and theory solvers that decide the satisfiability of formulas over specific theories. Each theory solver provides two capabilities: (1) theory propagation among various theory solvers for checking possible conflicts on partial assignments, and (2) clause learning result of which is shared by the SAT solver to speed-up pruning the decision tree. Additionally, since several applications of SMT deal with formulas involving two or more theories at ones, modern SMT solvers provide the capability of combining theory solvers using Nelson-Oppen [59] or Shostak [60] method to support a more expressive language. In combining theory solvers, if two theories Γ_1 and Γ_2 are both defined axiomatically, their combination can simply be defined as the theory axiomatized by a union of the axioms of the two theories, Γ_1, and Γ_2. For example, Consider Γ_1 and Γ_2 are two different theories, it is possible to define $\Gamma_1 \oplus \Gamma_2$ as a combined theory of Γ_1 and Γ_2, where $\Gamma_1 \oplus \Gamma_2$

is the set of all models that satisfy $\Gamma_1 \cup \Gamma_2$. This is adequate if the signatures of the two theories are disjoint. Otherwise, if Γ_1 and Γ_2 have symbols in common, one has to consider whether a shared function symbol is meant to stand for the same function in each theory or not. In the latter case, a proper signature renaming must be applied to the theories before taking the union of their axioms. In [61] they have described general conditions for the combination of theories that may have symbols in common. The ability to combine theory solvers proves extremely useful when dealing with applications such as model checking and predicate abstraction-based model check in which we need to check the satisfiability of formulas over several data types.

Theories are defined as classes of models with the same signature. More precisely, a Σ-theory Γ is a pair of (Σ, A) where Σ is a signature and A is a class of Σ-models. In general a theory solver for a theory Γ is a procedure which takes as input a collection of Γ-literals μ and decides whether μ is Γ-satisfiable. A theory (Γ-solver) to be effectively used within an SMT solver should have the following properties [62]: *(1) Model Generation*: theory solver should be able to produce a Γ-model of the problem description μ. *(2) Conflict Set Generation*: when the theory solver reaches inconsistency, it should be able to produce a subset η of μ which has caused the inconsistency. The subset η is referred to as *theory conflict*. *(3) Incrementality*: The Γ-solver should be able to save and keep its status across invocation calls to avoid recomputation. *(4) Backtrackability*: it is important for theory solver to has the ability to undo the step if it is needed. Equality with Uninterpreted Functions (EUF), linear real arithmetic (LRA), linear integer arithmetic (LIA), Mixed Integer and Real Arithmetic, Difference Logic, Bit Vectors, Arrays, etc. are the examples of theories commonly used in SMT.

In this work, we use an SMT solver and formulate some attacks against specific obfuscated circuits, illustrating the power of adapting various theory solvers for extending the capabilities of attack by constraining and monitoring non-logical properties of a netlist. For this purpose, and to illustrate that the SMT attack is a super-set to the SAT attack, we first illustrate that the original SAT attack against obfuscated circuits could be effectively

formulated using an SMT solver, resulting in a similar performance. Then we illustrate how the SMT solver could be used to attack logic obfuscation problems out of the reach of pure SAT attacks, and for that purpose, we break the logic and timing obfuscation in [31] which is not possible by a pure SAT attack. We illustrate that this attack could be achieved using both the Eager and Lazy approach of the SMT attack. Then we illustrate how the SMT attack could become significantly more efficient than a SAT attack by adopting the capabilities of theory solvers like *BitVector*, and formulate an accelerated SMT attack, that requires substantially smaller iterations and runtime compared to a SAT attack against specific obfuscation schemes. In addition, we formulate the accelerated SMT attack to be capable of approximate attacks.

3.5 SMT Attack Overall Flow

When building an SMT attack on obfuscated circuits, as illustrated in Fig. 3.2, the SMT attack could be invoked with any number and combination of theory solvers and a SAT solver. In order to use the SMT solver to formulate an attack, few preliminary steps should be taken. The first step is to make a minor modification to an extracted netlist after reverse engineering, providing the capability of testing various behaviors of the obfuscated circuit to the SAT or SMT solver. The transformation is simply replacing the obfuscated cells with their equivalent Key Programmable Gates (KPG). A KPG performs the same function as the obfuscated cell, however, it allows building a key-controlled representation of the logical behavior of the obfuscated cell for the purpose of the logical-model building. Fig. 3.3 captures the KPG translation gates for each type of the gates that have previously used in recent literature for the purpose of obfuscation. For example, when attacking a camouflaged cell that could be either an AND gate or an XOR gate, it is replaced with its KPG which is simply a MUX with each of its input tied to one of the camouflaged cell possibilities. The function performing the KPG replacement in the algorithms described in this work is ***ReplaceKPG(*** $\boldsymbol{N_{obf}}$ ***)*** that replaces all obfuscated cells in an obfuscated module with their KPGs equivalent based on the translation table in Fig. 3.3.

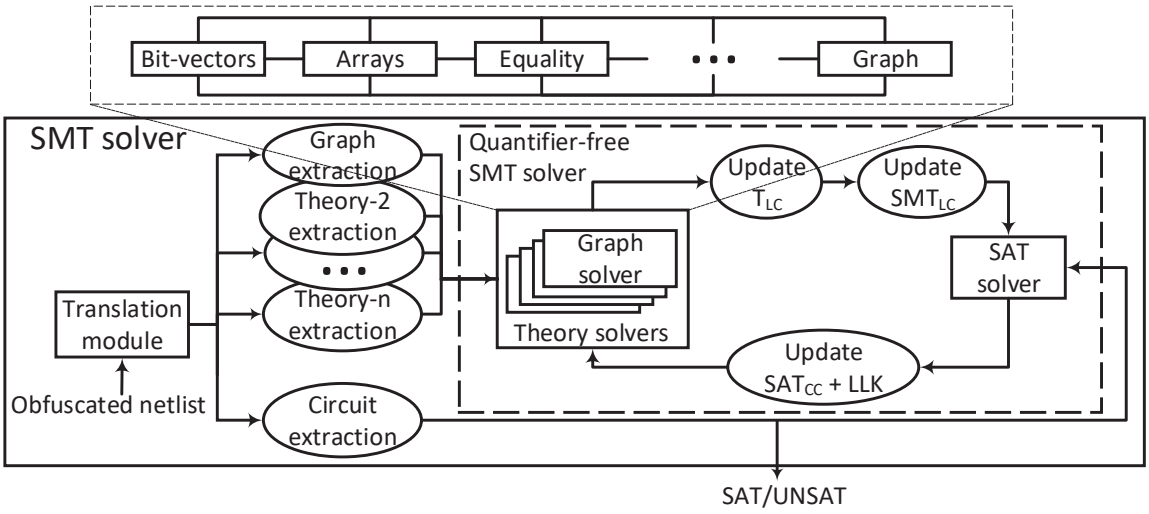


Figure 3.2: Overall Architecture of SMT Attack for Circuit Deobfuscation.

When using an SMT solver, before invoking a theory solver, the input model or input behavior should be translated to a model μ which is understood by that theory solver. As illustrated in Fig. 3.2, the translation step may be different for each theory solver used. As an example, to break the Delay Logic Locking in [31], we use a graph theory solver and translate the obfuscated netlist to a graph model that is understood by the graph-theory solver. The required translation step ($\mu \leftarrow$ Netlist) is simply the inversion of the netlist under attack to its graph representation, where each gate is a *node* in the graph, and each net an *edge*. We have additionally included the functionality to compute the logical effort in our graph translation routine, that annotates each edge with the logical effort needed to drive that edge as a measure of its delay. We could alternatively use a second theory solver to capture the static timing of the netlist and exchange information with the graph theory solver for more accurate results. The final step before invoking the SMT/SAT attack is the translation of the netlist under attack into its CNF form as described in [1].

After building model μ for each Theory and SAT solver, the SMT attack is formulated based on the flow of information exchange between theory and SAT solver. In General, the formulation of the SAT portion of SMT solver is similar to that of pure SAT attack as described in [1]. However, in addition to the SAT solver, each theory solver is then

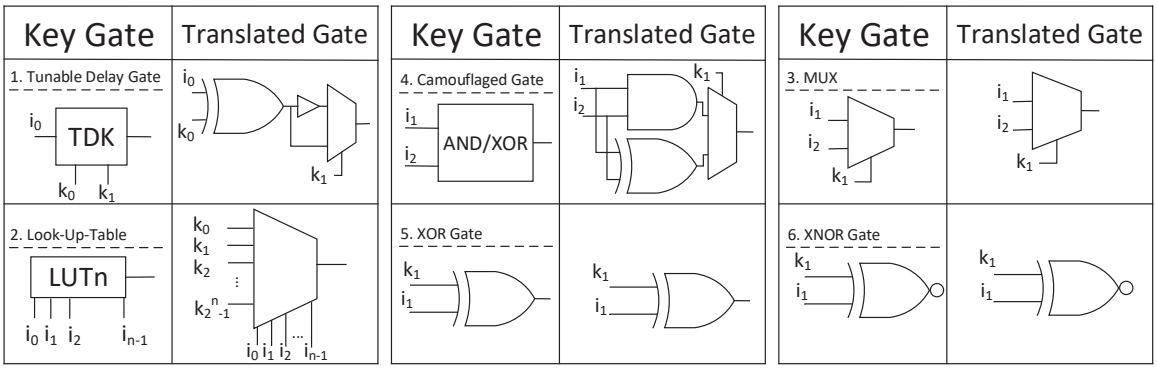


Figure 3.3: Translation Table to Key Programmable Gates (KPG).

tuned by a declaration of theory constraint. At this stage, invoking the SMT solver returns a satisfiable assignment and a list of learned theory and conflict clauses for theory solver and SAT solver respectively. The SMT attack is then achieved by composing the correct control flow for invocation of theory and SAT solver(s), and by managing the intermediate sequence of CNF-based information exchange. The general flow of information in an SMT formulated problem, including that of SMT attack, is illustrated in Fig. 3.2.

3.6 SMT Attack Mode 1: SMT reduced to SAT Attack

As was mentioned previously, the SAT attack finds a functionally correct key for an obfuscated circuit by checking a small subset of all input patterns, hence removing the need for brute-force testing of all input patterns. Considering that SMT solver is a superset of SAT solver and contains a SAT solver, any attack formulated for SAT could be similarly formulated for an SMT solver.

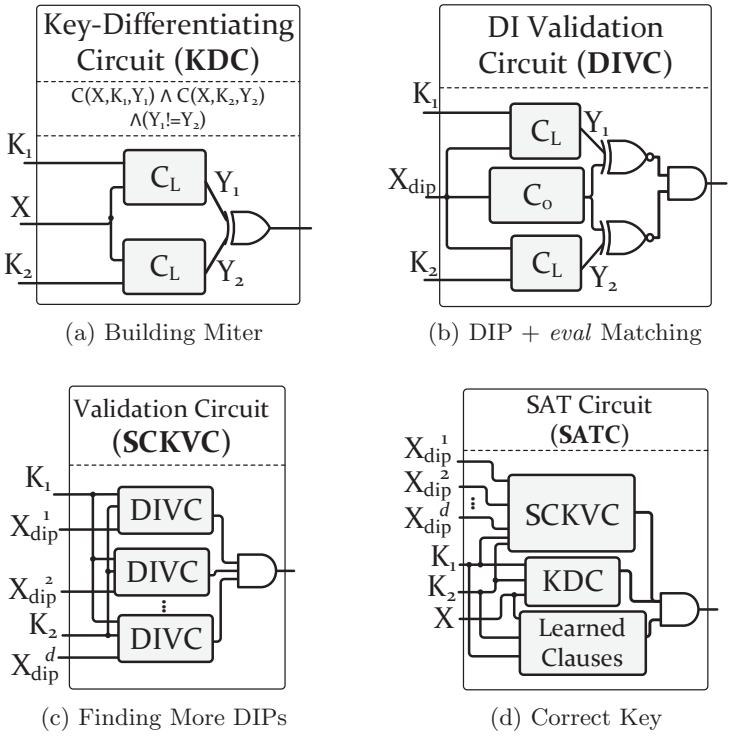
Alg. 10 illustrates the SAT attack that could be similarly implemented in a SMT solver. The formulation of attack remains similar to that of the original attack proposed in [1, 2].

The SAT attack in Alg. 10 follows the steps illustrated in Fig. 3.4. In this algorithm, the obfuscated gates are first replaced with key programmable gates (KPG) to create the Key Programmable Circuit (KPC). Then the CNF representation of the circuit is generated. Two KPCs are then used to generate a Key Differentiating Circuit (KDC). The KDC

Algorithm 10 SMT Reduced to SAT Attack in [1, 2]

```
1: function SAT_Attack(Obfuscated_Netlist N_obf, Functional_Circuit C_org)
2:     KPC ← ReplaceKPG(N_obf);
3:     C(X, K, Y) ← Circuit_Translation_to_CNF(KPC);
4:     KDC = C(X, K_1, Y_1) ∧ C(X, K_2, Y_2) ∧ (Y_1 ≠ Y_2);
5:     SCKVC = TRUE;
6:     SATC = KDC ∧ SCKVC;
7:     LC = TRUE;                                          ▷ Learned Clauses
8:     SMT_LC ← SATC;
9:     while (((X_DI, K_1, K_2, CC) ← SMT.Solve(SMT_LC))= TRUE) do
10:        Y_f ← C_org(X_DI);
11:        DIVC = C(X_DI, K_1, Y_f) ∧ C(X_DI, K_2, Y_f);
12:        SCKVC = SCKVC ∧ DIVC;
13:        LC = LC ∧ CC
14:        SMT_LC = KDC ∧ SCKVC ∧ LC;
15:    Key ← SMT.Solve(SMT_LC);
```

receives an input and two different keys and determines whether they generate the same output or not. The KDC is then used as the first SMT satisfiability problem represented by SMT_{LC} for the first invocation of SMT solver. Calling the SMT *solve* function on the posed formula then return an assignment for keys K_1, K_2, and the discriminating input X_{DI} such that the formulated SMT_{LC} is satisfied. In addition, the SMT solver returns a list of learned Conflict Clauses (CC). In line 10, the correct output (Y_f) for the discriminating input X_{DI} is found. In the next step, the SMT formula needs to be updated to use the discriminating input and learned clauses to further constrain the satisfiability problem. This is done in multiple steps. In line 11, the discriminating input found in the current iteration is used to create a Discriminating Input Validation Circuit (DIVC) which is illustrated in Fig. 3.4(d). The DIVC circuits formed at each iteration are ANDed together to create a circuit that checks the correctness of a key for all previously found discriminating inputs. This circuit is referred to as Set of Correct Key Validation Circuit (SKCVC). In line 13, the currently found Conflict Clauses are added to the set of previously found Learned Clauses (LC). Note that this step is done implicitly for SMT, which is a stateful solver. Finally, in line 14 the SMT satisfiability problem is constrained by ANDing together the KDC, SCKVC and LC clauses. The SAT attack formulated using SMT solver continues until the SMT solver returns UNSAT. A final call to the SMT solver returns the correct key. Note that this SMT



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

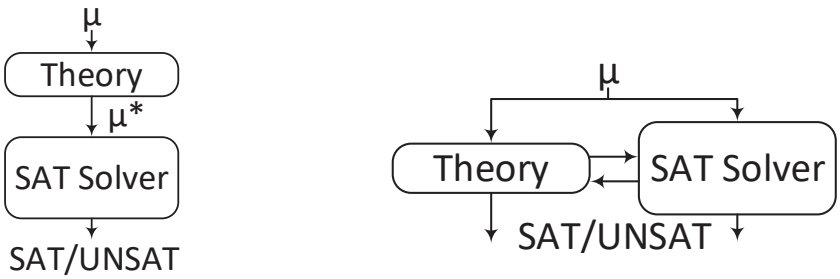
(c) Finding More DIPs (d) Correct Key

Figure 3.4: From Obfuscated Circuit to Recovering the Key in the SAT Circuit.

attack is a one-to-one translation of the original SAT attack in [1, 2]. In section 3.10.1, we illustrate that the formulation of SAT attack using SMT solver results in a very similar performance to that of pure SAT attack. However, the SMT attack could further benefit from the usage of SMT solvers to extend its capabilities to attack obfuscation schemes that could not be logically modeled.

3.7 SMT Attack Mode 2: Eager SMT Attack

Theory solvers could be used to extend the capabilities and performance of SMT solver compared to that of a SAT solver. This, as illustrated in Fig. 3.5 could be done either by (1) using the theory solver to extract all required clauses that complete the CNF description with respect to the obfuscation scheme and then perform a SAT attack, referred to as the SMT *Eager* approach, or (2) by invoking the theory and SAT solver in parallel to



(a) Eager SMT (b) Lazy SMT

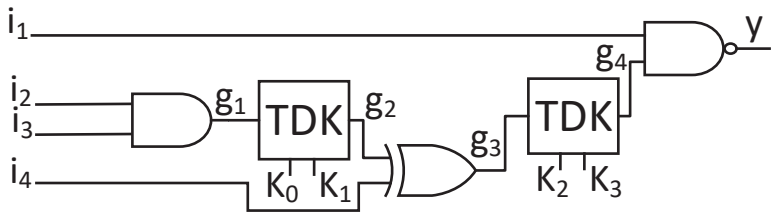
Figure 3.5: SMT Execution Flow.

simultaneously model and solve the problem, referred to as *Lazy* approach.

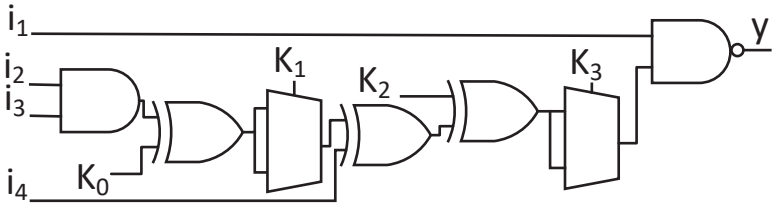
In this section, we illustrate how the Eager approach of SMT attack could be used to attack the obfuscation schemes that could not be broken or understood by a pure SAT attack. For this purpose, we formulated an SMT attack on the delay-locking (DLL) scheme proposed in [31]. Notice that the proposed approach could be used in formulating attacks on other obfuscation techniques that rely on non-logical properties of circuit obfuscation such as timing, power, delay, etc. by using the appropriate theory solvers.

Fig. 3.6 illustrates the translation steps for converting a DLL[31] obfuscated circuit (using translation table in Fig. 3.3) to its key programmable circuit and its graph representation. As illustrated in Fig. 3.6(b), K_1 effectively has no impact on the logical behavior of the circuit and only changes its delay properties. Hence, subjecting this obfuscated circuit to a SAT attack results in a random assignment to K_1. Therefore, by having k TDK cells, which have $2k$ keys in total, a SAT solver returns one logically correct key sequence among 2^k different set of such logically correct keys that control the TDK cells, however, only one of such keys doesn't result in setup and hold violations. Hence, a correct attack should consider the delay and timing properties of the netlist in addition to its logical correctness.

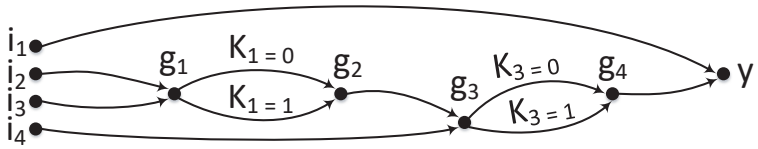
The shortcoming of SAT attack to capture the delay and timing properties of the netlist, when attacking DLL obfuscation, is remedied in the proposed SMT attack using a graph theory solver. To illustrate this, we formulate an Eager and a Lazy SMT attack on DLL obfuscation. In the Eager approach, we use the theory solver as a pre-processing step, in



(a) netlist obfuscated with TDK cells



(b) Key Programmable Circuit (KPC)



(c) representative graph of the netlist

Figure 3.6: Conversion Flow in SMT using Graph Theory Solver.

which we deduct the complete set of Valid-Path Constraint Clauses (VPCC) between all primary inputs and outputs of the obfuscated netlist. This VPCC is a CNF representation of all valid assignments of the keys, such that no setup or hold violation is created. Note that among many such possibilities, only one possibility has both the correct timing and the correct logical behavior.

To build the VPCC clauses, we should compute the setup and hold constraints on every timing path. The setup and hold timing checks for a timing path is expressed using the following inequalities:

$$t_{cs-lr} + t_{clk-q} + t_p + t_{setup} + U \leq t_{cs-cr} + T_{clk} \tag{3.1}$$

$$t_{cs-lr} + t_{clk-q} + t_{cd} \geq t_{hold} + t_{cs-cr} + U \tag{3.2}$$

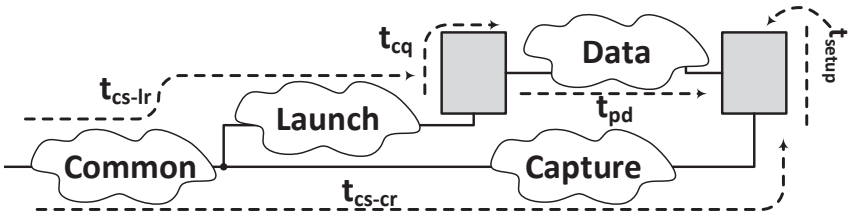


Figure 3.7: Various Delay Components of a Timing Path.

In this equation which uses the notation in Fig. 3.7, the t_{cs-lr} is the clock source to launch register delay, t_{cs-cr} is the clock source to capture register delay, U is the clock jitter/uncertainty, t_{clk-q} is the clock to q delay of the launch register, t_{setup} is the capture-register setup time, t_{hold} is the hold time requirement for the capture register, t_p is the propagation delay through the longest path in the timing path, and finally the t_{cd} is the propagation delay through the shortest path in the logic. Considering that the DLL logic is only inserted on *Data* sections of timing path, it can only affect the t_p and t_{cd}. Note that it is also possible to enhance the DLL obfuscation beyond that described in [31] and use the TDK cells for building clock skew in the clock network, however, a similar attack still could be formulated. For now, let's consider that DLL, as described in [31], only affects the Data section of the timing path. The equations 3.1 and 3.2 could be re-written as:

$$\mathbf{t}_p \leq T_{clk} + (t_{cs-cr} - t_{cs-lr}) - t_{clk-q} - t_{setup} - U = Upper \tag{3.3}$$

$$\mathbf{t}_{cd} \geq t_{hold} + (t_{cs-cr} - t_{cs-lr}) - t_{clk-q} + U = Lower \tag{3.4}$$

Before performing any reverse engineering, we know the T_{CLK} from the functional chip purchased on market. Note that a functional chip (the oracle) is needed to perform the SAT or SMT attack as explained in section 3.3. Now let's consider a netlist obtained after reverse engineering. The end-point and start-point registers for each timing path are known. Hence, by means of spice simulation, the register could be characterized and the t_{clk-q}, t_{setup} and

t_{hold} are extracted. Note that there is a limited type of registers used in a physical design, and at this step, only a handful of registers need to be characterized. Extracting a measure for uncertainty could be also achieved by means of spice simulation.

At this point, considering that a TDK cell can change the delay of a timing path, the delay of each timing path (D_j) could be divided into a constant delay (C_j) and a variable delay ($V_j(K)$), where the variable delay is a function of the number of TDK cells in that timing path, and the key assumed for each TDK. Hence, Delay of Timing path j from start point s to endpoint p ($D_j^{s\to p}$) that passes through N TDK cells each with delay $D_{TDK}^{s\to p}(i)$, depending on the value of key K_i is obtained from:

$$D_j^{s\to p} = C_j^{s\to p} + V_j^{s\to p}(k) \tag{3.5}$$

$$D_j^{s\to p} = C_j^{s\to p} + \sum_{i=1}^{N} K_i \times D_{TDK}^{s\to p}(i) \tag{3.6}$$

For a given timing path, and by using the equation 3.6, we could rewrite the delay constraints in equations 3.3 and 3.4 as:

$$\forall j| \quad D_{j_{max}}^{s\to p} = C_{j_{max}}^{s\to p} + \sum_{i=1}^{N} K_i \times D_{TDK}^{s\to p}(i) \leq Upper \tag{3.7}$$

$$\forall j| \quad D_{j_{min}}^{s\to p} = C_{j_{min}}^{s\to p} + \sum_{i=1}^{N} K_i \times D_{TDK}^{s\to p}(i) \geq Lower \tag{3.8}$$

These inequalities capture the lower and upper bound delay constraint for every pair of input-output pins in a design and collectively capture the model μ of the graph theory solver. Based on this formulation, the number of added inequalities is M $\times$ N, in which M is the number of primary inputs, and N is the number of primary outputs. However, one

inequality bounds all timing paths between the selected input-output pin pair, removing the need to express the inequality for every timing path in the design as needed in MILP-based attack that was suggested in [31].

Algorithm 11 Eager SMT Attack on DLL [31]

```
1: function SMT_Eager_Att(Obfuscated_Netlist N_obf, Functional_Circuit C_org)
2:     KPC ← ReplaceKPG(N_obf);
3:     C(X,K,Y) ← Circuit_Translation_to_CNF(KPC);
4:     KDC = C(X, K1, Y1) ∧ C(X, K2, Y2) ∧ (Y1 ≠ Y2);
5:     SCKVC = TRUE;
6:     SATC = KDC ∧ SCKVC;
7:     LC = TRUE;                                          ▷ Learned Clauses
8:     G(X,K) ← Graph_Translation(N_obf);
9:     T_LC ← GenTLC(G(X,K));                              ▷ Theory Learned Clauses
10:    SMT_LC ← SATC ∧ T_LC;                               ▷ SMT Clauses
11:    while (((X_DI,K1,K2,CC) ← SMT.Solve(SMT_LC))= TRUE) do
12:        Y_f ← C_org(X_DI);
13:        DIVC = C(X_DI,K1,Y_f) ∧ C(X_DI,K2,Y_f);
14:        SCKVC = SCKVC ∧ DIVC;
15:        LC = LC ∧ CC
16:        SMT_LC = KDC ∧ SCKVC ∧ LC;
17:    Key ← SMT.Solve(SMT_LC);
```

Pre-Processing step by using a graph theory solver for SMT attack (*Eager*)

```
1: function GenTLC(Graph G)
2:     Inputs ← G.find_start_points();
3:     Outputs ← G.find_end_points();
4:     T_LC ← []
5:     for each (Sp in Inputs) do
6:         for each (Ep in Outputs) do
7:             Upper(Sp,Ep)(K) ← !(distance_leq(Sp, Ep, t_cd));
8:             Lower(Sp,Ep)(K) ← distance_leq(Sp, Ep, t_p);
9:             T_LC ← SMT.solve(Upper(Sp,Ep)(K) ∧ Lower(Sp,Ep)(K) ∧ T_LC);
10:    return T_LC
```

After writing these inequalities for each input-output pair, a call to the SMT *solve* function returns all key combinations for which all paths constraints/inequalities are satisfied. In the other word, by assuming any of the returned key combinations, the circuit will not violate its setup and hold timing checks. However, note that only one (or few) of these key values is logically correct. The correct key value then could be extracted by invoking a SAT solver, and by providing the set of key combinations (in CNF format) as a constraint to

the logical circuit satisfiability problem. This process is illustrated in Alg. 11. As it can be seen in Alg. 11, function *GenTLC* is responsible for generating all inequalities. Line 7-8 of *GenTLC* function generates inequality for each input (Sp) to each output (Ep).

This algorithm is similar to Alg. 10, with the additional step of using a theory solver for pre-processing the netlist in line 8, extraction of all key combination resulting in a correct timing behavior in line 9, and providing these constraints to the SAT solver in the next step in line 10. Note that the *solve* function in the Eager approach is called in two places; first for generating the timing valid key combination clauses (inside GenTLC function), and then iteratively inside the SAT attack while loop.

For some obfuscation methods, the pre-processing step of the Eager approach may become extremely time-consuming or computationally impossible. An example of such obfuscation problem is the SRCLock [28]. The authors have shown that the obfuscation is SAT hard, since without pre-processing the cycles, the SAT solver will be trapped or produce an incorrect key. Additionally, they have suggested two mechanisms by which the number of cycles in a netlist could exponentially grow with respect to the number of inserted feedbacks. For attacking cyclic logic, as suggested by CycSAT attack [44] we need to pre-process the netlist and extract the No Cycle Conditions to prevent the SAT solver from being trapped. However, in SRCLock[28] the number of cycles grows exponentially, and therefore the runtime of pre-processing step also grows exponentially, preventing us to ever reach the SAT attack. For such problems, the Eager approach that relies on the reduction of the problem to a SAT problem does not work. However, the Lazy approach of the SMT attack provides a solution.

3.8 SMT Attack Mode 3: Lazy SMT Attack

Using the Lazy approach of SMT attack relaxes the requirement of Eager approach to complete the pre-processing step before invoking the SAT attack.

In the Lazy approach, the SAT solver and theory solver(s) simultaneously check different models of a unified satisfiability problem, exchange clauses, and check each other's literal

assignment. This could significantly prune the decision tree of a SAT solver search space for finding a satisfying assignment and remove the need for complete and unbounded execution of theory solver as it only has to check the validity of constraints for SAT assigned literals.

Algorithm 12 Overall SMT Attack (*Lazy* Approach)

1: **function** SMT_Lazy_Att(Obfuscated_Netlist N_{obf}, Functional_Circuit C_{org})
2: $KPC \leftarrow$ ReplaceKPG(N_{obf});
3: $C(X,K,Y) \leftarrow$ Circuit_Translation_to_CNF(KPC);
4: $KDC = C(X, K_1, Y_1) \wedge C(X, K_2, Y_2) \wedge (Y_1 \neq Y_2)$;
5: $SCKVC = TRUE$;
6: $SATC = KDC \wedge SCKVC$;
7: $LC = TRUE$; ▷ Learned Clauses
8: $G(X,K) \leftarrow$ Graph_Translation(N_{obf});
9: $T_{CE}(K) \leftarrow GenTCE(G(X, K))$; ▷ Theory Constraint Expressions (Not Solved)
10: $T_{CE}(K1, K2) \leftarrow T_{CE}(K1) \cup T_{CE}(K2)$;
11: **while** ((X_{DI},K_1,K_2,CC) $\leftarrow$ $SMT.Solve(SATC, T_{CE}(K1, K2))$= $TRUE$) **do**
12: $Y_f \leftarrow C_{org}(X_{DI})$;
13: $DIVC = C(X_{DI},K_1,Y_f) \wedge C(X_{DI},K_2,Y_f)$;
14: $SCKVC = SCKVC \wedge DIVC$;
15: $LC = LC \wedge CC$
16: $SMT_{LC} = KDC \wedge SCKVC \wedge LC$;
17: $Key \leftarrow SMT.Solve(SMT_{LC}, T_{CE}(K))$;

Initialization of constraints for SMT attack (*Lazy* Approach)

1: **function** GenTCE(Graph G)
2: $Inputs \leftarrow G$.find_start_points();
3: $Outputs \leftarrow G$.find_end_points();
4: $T_{CE}(K) \leftarrow []$
5: **for each** (Sp **in** $Inputs$) **do**
6: **for each** (Ep **in** $Outputs$) **do**
7: Upper(Sp,Ep)(K) $\leftarrow$!(**distance_leq**(Sp, Ep, t_{cd}));
8: Lower(Sp,Ep)(K) $\leftarrow$ **distance_leq**(Sp, Ep, t_p);
9: $Range(Sp,Ep)(K) \leftarrow$ Lower(Sp,Ep)(K) $\wedge$ Upper(Sp,Ep)(K);
10: $T_{CE}(K) \leftarrow T_{CE}(K) \cup Range(Sp,Ep)(K)$;
11: **return** $T_{CE}(K)$

In order to illustrate the Lazy approach of SMT attack, in this section, we formulate an SMT attack to again break the DLL [31] obfuscation. The Lazy approach of SMT attack on DLL [31] is illustrated in Alg. 12. The big difference in the Lazy and Eager approach is that after model generation for theory solver, the SMT *solve* function is not called. This is illustrated in line 9 of this algorithm, where the constraining expressions are only defined for the theory solver by making a call to $GenTCE$ function. The returned constraining

expressions are then duplicated for K_1 and K_2. The SMT *solve* function is then called to find an assignment for a discriminating input X_{DI}, and two different keys $K1$ and $K2$ such that generated outputs are different at least in one bit, however, both keys generate a valid timing scenario. Since the SAT model (SATC) and Theory models ($T_{CE}(K1, K2)$) share literals and are subjected to a unified set of constraints, the decision tree and search space for the SMT solvers is significantly reduced.

3.9 SMT Attack Mode 4: Accelerated Lazy SMT Attack

In this section, we argue that re-formulating the Lazy SMT which benefits from capabilities of *BitVector* theory solver allows us to build a more efficient attack.

Our modification to the SAT attack is inspired by the observation that higher output corruption, reduces the SAT hardness of an obfuscation scheme. A discriminating input X_{DI}, is an input capable of sensitizing the logic paths of the netlist under study, such that (1) some of the differences in the values of internal nodes in the result of application of two different keys K_1 and K_2 are propagated to at least one output. (2) none of the previously found DIPs (that were used in building a DIVC) were able to propagate the generated inconsistency to a primary output. This mechanism is continued until the number of sensitized paths, reaches a point where any inconsistency is propagated to the primary outputs using the constructed set of DIVC circuits. At this point, the set of previously found X_{DI}s form a complete set of discriminating inputs, such that if a key generates the correct output for all inputs in this set, it will generate the correct output for all other inputs.

Different DIPs have different pruning power. A DIPs strength could be assessed based on the number of inconsistencies that it could sensitize to the primary outputs conditioned that previous DIPs were incapable of doing so. Hence, depending on the pruning power of DIPs, the size of the complete set of DIPs could be different. A minimal complete set of DIPs is the smallest set of DIPs that could de-obfuscate the circuit. In our Lazy approach for SMT attack, we propose a mechanism to reduce the size of the complete set of DIPs

pushing it towards the minimal set. Since in each SAT or SMT iteration one DIP is found, having a smaller number of DIPs result in a smaller number of iterations.

In the SAT attack, it requires only a single bit difference in the output for the generation of a DIP. In SMT attack, we could make a stronger requirement for the generation of DIPs. This could be achieved by forcing the SMT solver to find DIPs with the largest possible Hamming distance of primary outputs when for the same input, two different keys are applied. Such a DIP has a much higher pruning capability, and is able to sensitize a larger number of key-related inconsistencies to the output. The discovery of such powerful DIPs reduces the number of required DIPs that is needed to form a complete set of DIPs that could de-obfuscate the circuit, reducing the attack time by almost an order of magnitude.

3.9.1 The Usage of BitVector Theory Solver

Assessing DIPs based on the hamming distance of the primary output is easily implementable in SMT solver by using a *BitVector* theory solver. The BitVector theory solver allows us to perform integer-oriented arithmetic operations such as addition, subtraction, and multiplication. The Hamming Distance (HD) of output Y_1 and Y_2 is obtained using:

$$HD(C(X_{DI}, K_1), C(X_{DI}, K2)) = HD(Y_1, Y_2) = \sum_{i=1}^{N} Y_1(i) \oplus Y_2(i) \tag{3.9}$$

The HD is then used to write the constraining expressions that are posed on the BitVector theory solver using the formulation:

$$Th_{Lower} \leq HD(Y_1, Y_2) \leq Th_{Upper} = Size(Output) \tag{3.10}$$

The upper threshold Th_{Upper} is kept constant equal to the size of output pins, but the lower threshold Th_{Lower} is defined as a variable, allowing us to sweep the hamming distance constraint posed on BitVector theory solver from a maximum value of the number of output bits to a minimum value of 1. The lower bound could be reduced every time the SMT solver

returns UNSAT, indicating there is no other DIP that satisfies the HD requirement of the theory solver. The process terminates when the SMT cannot even find a DIP that causes HD of 1. Adaption of this constraint forces the SMT solver to find DIPs with higher pruning power, reducing the size of a complete set of DIPs.

3.9.2 The Usage of Timeout

For an SMT or a SAT attack, the execution time is determined based on the formula, $\sum_{i=1}^{N} t(i)$, where $t(i)$ is the execution time of the i^{th} iteration of an SMT attack. Hence, by just reducing the number of SAT iterations N, we cannot guarantee a shorter execution time, because finding a DIP with tighter constraint may pose a more difficult problem to the SMT solver and increase $t(i)$. For this purpose, we can limit the time allowance for finding a DIP in each iteration. The timeout limit TO prevents the SMT solver from spending a long time for finding a DIP with a large HD when finding such DIP has become excessively difficult. By adapting the timeout feature, during an SMT attack, the HD requirement is reduced when either (1) the SMT solver returns UNSAT, indicating there exists no such input, or when (2) we encounter time-out interrupt. In this case, the HD constraint posed on the BitVector theory solver is reduced by one and the SMT solver is called. Note that the time interrupt is supported by MonoSAT [63] used in this work, and many other freely available SMT solvers. Also, note that the use of time interrupt pushes the final solution away from a minimal complete set of DIPs. However, our experiments illustrate that this usually results in considerably smaller execution time.

3.9.3 Enabling Approximate Attack

Our objective is to enable the SMT attack to be carried against a netlist similar to that of Fig. 3.8, which is obfuscated by both SAT Hard (SH) and high Corruption (HC) obfuscation schemes, to find all keys for the HC obfuscation, and to detect the trap of SH obfuscation and exit while generating an approximate key.

The SAT hard obfuscation mechanisms suggested in recent literature, such as *SARLock,*

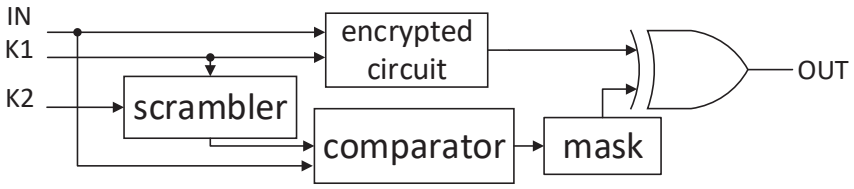


Figure 3.8: A Hybrid Obfuscation Scheme.

Anti-SAT, and SFLL [21–23], have a very small output corruption, and the SAT hardness is maximized when there is only a single input for a given key that results in an incorrect output. The pruning power of DIPs found in each iteration of the SAT solver for SH obfuscation solutions is very small, and each DIP eliminates a single key value. Hence, the number of SAT or SMT iterations increases exponentially with respect to the key size. This is used as a mechanism to trap the SAT solver. To increase the corruption, the SH obfuscation is combined with a HC obfuscation. The purpose of approximate attacks is to find the correct key for the HC obfuscation without being trapped by SH obfuscation.

The accelerated SMT attack could significantly improve the performance of approximate attacks. Since HC obfuscation schemes result in high output corruption, finding DIPs that lead to larger HD at the output biases the SMT attack to find the HC related obfuscation keys in the earlier iterations. The remaining problem is the design of a termination strategy for the accelerated and approximate SMT attack to detect the trap of SH obfuscation, exit, and report the approximate key. For this purpose, we use a constraint on the number of allowed repetitions R when HD is very small (e.g. 1). If the remaining and un-found keys are only the SH keys, the SMT keeps finding weak DIPS (HD of 1) and iterations are completed very quickly. By setting the repetition limit R to an appropriately large value, we can detect the trap and terminate the attack.

The unique feature of accelerated approximate attack is that if we remove the timeout (TO) requirement, then the approximate attack guarantees that the HD of the approximately unlocked circuit and that of the functional circuit is at most HD_{Low} bits different, with HD_{Low} being the hamming distance requirement in which the R repetition is taken

place. This could be proven as follows: Suppose that there exist an undiscovered discriminating input and two keys that cause larger than HD_{Low} bit difference (HD_{Low}+D) in the primary outputs. Hence, the SMT solver when constrained by BitVector theory solver for finding HD = HD_{Low}+D should return SAT. This contradicts the SMT previous execution control state where the SMT attack for that HD has returned UNSAT, otherwise, the HD constraint was not reduced.

3.9.4 Accelerated SMT Attack Formulation

Alg. 13 demonstrates the reformulated Lazy approach of SMT attack on obfuscated circuits. In this algorithm, the HD_{High}, and HD_{Low} are the high and low threshold requirement for hamming distance on primary outputs, TO is the timeout limit per iteration, R is the repetition allowance before exiting and generating an approximate key, and R_{HD} is the hamming distance after which the repetition condition is checked.

The BitVector theory solver input model is defined in lines 14 and 15, and converted to theory constraint expressions in lines 16 and 17. The T_{CE} poses an upper and lower bound on the hamming weight difference of the outputs of two instances of the same circuits with the same input, but two different keys. The SMT attack sweeps the hamming distance in the first while loop, while the second while loop formulates the modulo satisfiability theory attack. The SMT solver receives the SMT_{LC} model, the BitVector theory solver constraint T_{CE} and the timeout allowance TO and check whether there is a valid assignment for SMT_{LC} conditioned that T_{CE} is valid withing TO time allowance. If it exists, the while loop is satisfied. Additionally, it returns the discriminating input X_{DI}, the two keys found (K_1, K_2) and a list of learned conflict clauses CC. Then the X_{DI}, similar to the original SAT attack is used to construct additional DIVC and update the satisfiability model SMT_{LC}. At the end of each iteration, the algorithm checks whether the hamming distance is reduced to the limit, where the repetition condition for SH problems is checked. In this case, if the repetition count reaches the specified threshold value R, the SMT attack is terminated.

Algorithm 13 Accelerated SMT Attack

```
1: function AccSMT_Attack(Obfuscated_Netlist N_obf, Functional_Circuit C_org)
2:     HD_High = Number of output bits;                    ▷ Upper hamming distance limit;
3:     HD_Low = HD_High - 1;                               ▷ Lower hamming distance limit;
4:     TO = 50s;                                           ▷ Timeout constraint;
5:     R = 20;                                             ▷ Repetition limit;
6:     R_HD = 1;                                           ▷ Repetition condition;
7:     R_count = 0;                                        ▷ Repetition count variable;
8:     KPC ← Replace_KPG(N_obf);
9:     C(X,K,Y) ← Circuit_Translation_to_CNF(KPC);
10:    KDC = C(X, K1, Y1) ∧ C(X, K2, Y2) ∧ (Y1 ≠ Y2);
11:    SCKVC = TRUE;
12:    SATC = KDC ∧ SCKVC;
13:    LC = TRUE;                                          ▷ Learned Clauses
14:    BV(X,K) ← Circuit_Output_to_BitVector(N_obf);
15:    BVS(X, K1, K2) = SUM_of_1s(BV(X,K1) ⊕ BV(X,K2))
16:    T_CE ← BVS(X,K1,K2) ≥ HD_Low;                       ▷ Theory constraint expression;
17:    T_CE ← T_CE ∪ (BVS(X, K1, K2) ≤ HD_High);
18:    while HD_Low ≥ 1 do
19:        while (((X_DI,K1,K2,CC) ← SMT.Solve(SMT_LC, T_CE, TO)) = T) do
20:            Y_f ← C_org(X_DI);
21:            DIVC = C(X_DI,K1,Y_f) ∧ C(X_DI,K2,Y_f);
22:            SCKVC = SCKVC ∧ DIVC;
23:            LC = LC ∧ CC
24:            SMT_LC = KDC ∧ SCKVC ∧ LC;
25:            if (HD_Low ≤ HD_R) then
26:                if (R_count == R) then
27:                    Break;
28:                R_count ++;
29:        HD_Low--;
30:    Key ← SMT.Solve(SMT_LC);
```

3.10 SMT Attack Performance Evaluation

For evaluating different modes of SMT Attack, we used a farm of desktops with a 4-core Intel Core-i5 CPU, running at 1.8GHz, with 8 GB RAM. The operating system on desktops was Ubuntu Server 16.04.3 LTS. For a fair comparison, and to reduce the impact of the operating system background processes, we dedicated one desktop to each SMT solver at a time. For benchmarking, we used most of the ISCAS-85 benchmarks, characteristics of which is listed in Table 3.1. Since MiniSAT has been used in the SMT Solver as its built-in SAT solver, we use the default values of resource limits in MiniSAT as resource limits of the SMT attack (68 years for the CPU time limit and $\approx$ 2147 TB for the memory usage

Table 3.1: ISCAS-85 Benchmarks and their Characteristics.

Circuit	c432	c499	c880	c1355	c1908	c2670	c3540	c5315	c7552
# of Inputs	36	41	60	41	33	233	50	178	207
# of Outputs	7	32	26	32	25	140	22	123	108
# of Gates	120	162	320	506	603	872	1179	1726	2636

Table 3.2: Execution Time of SAT vs SMT (Attack Mode 1).

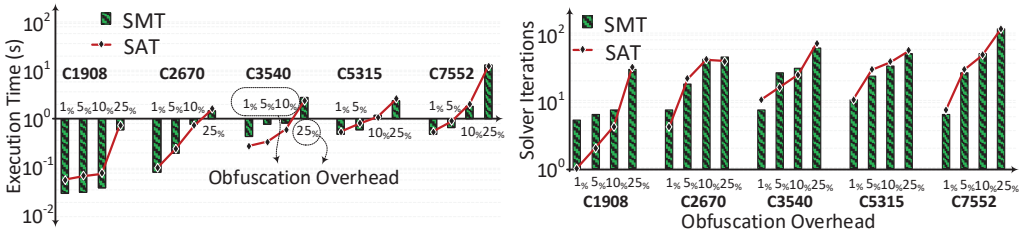
Circuit	c2670				c3540				c5315				c7552			
	SAT		SMT		SAT		SMT		SAT		SMT		SAT		SMT	
	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time
1%	3	0.102	5	0.474	10	0.513	8	1.31	9	0.405	10	0.441	11	0.577	19	0.806
5%	45	1.514	57	3.589	19	1.502	25	1.249	32	1.354	24	2.433	67	5.271	42	4.261
10%	312	14.08	342	15.752	36	1.782	36	2.973	59	3.798	57	4.881	97	15.82	94	15.67
25%	781	114.5	692	108.6	77	9.796	65	8.462	95	19.63	107	22.48	215	225.6	228	270.8

limit). As the baseline for comparing SMT attack performance against a pure SAT attack, we employed the Lingeling-based SAT attack by [1]. In addition, for each attack, we ran the solvers *Five* times on SMT and SAT solvers and reported the average runtime.

3.10.1 Evaluation of SMT reduced to SAT Attack

As explained in section 3.6, and explained by Alg. 10 the SMT solver could be used for a SAT attack using the same formulation as the original SAT attack as proposed in [1, 2]. In this section, we evaluate the performance of SMT attack when used in this mode. The purpose of this section is to illustrate that attack formulate using the SMT solver is a superset of SAT attacks, and with the same formulation provides similar performance. For this comparison, we employed two obfuscation methods: (1) random XOR/XNOR insertion (RLL) [16], and (2) obfuscation using nets with unbalanced probabilities (*IOLTS'14*) [64]. ISCAS-85 benchmarks are obfuscated using these schemes with obfuscation overhead ranging from 1% to 25%.

Table 3.2 compares the execution time of SMT attack and the SAT attack proposed in [1,2] when RLL obfuscation is deployed. As captured in this table, the execution time of the



(a) (b)

Figure 3.9: Performance Comparison between SMT Attack and SAT Attack.

SMT attack when reduced to SAT Attack is approximately equivalent, in terms of a number of iteration and execution time, with that of an original SAT attack across all benchmarks and all ranges of obfuscation overhead. Fig. 3.9 illustrates the same comparison when the *IOLTS'14* obfuscation method is deployed. As illustrated, the SMT reduced to SAT, in terms of performance, behaves similarly to the SAT attack.

3.10.2 Evaluation of Eager SMT Attack

We used the Delay Logic Locking scheme [31] in our case study to show the extended capabilities of the SMT attack in solving obfuscation problems that cannot be modeled in a SAT attack. The Eager approach of SMT attack is evaluated in this section, and the Lazy approach is evaluated in the following section. Additionally, to increase the obfuscation difficulty and demonstrate the strength of the SMT attack, in addition to obfuscation using DLL, we obfuscated the circuit with additional MUX and XOR gates using gate insertion policy in *IOLTS'14* [64], such that 50% of the keys are used for *DLL*, and 50% for *IOLTS'14* obfuscation. Finally, we used some of the keys for both logic and delay obfuscation to create dependencies such that the solvers could not divide and conquer the attack.

The Eager attack against DLL was formulated in Alg. 11. As the algorithm suggests, the Eager approach attacks the obfuscation in two separate phases. In the first phase, the theory solver models and constrains the problem and calls the SMT solver to extract all valid key combinations. The key combinations are converted into a CNF statement, which is passed

Table 3.3: Execution Time of SMT Attack in the Eager Mode (Attack Mode 2).

Circuit	c1908	c2670	c3540	c5315	c7552
1%	0.077 + 1.663	0.068 + 170.0	0.053 + 4.054	1.291 + 114.6	0.580 + 138.6
2%	0.016 + 1.919	0.221 + 175.6	0.200 + 5.001	1.535 + 144.6	1.808 + 185.5
3%	0.054 + 2.161	0.337 + 212.7	1.359 + 6.328	3.057 + 160.4	2.247 + 245.9
5%	0.075 + 2.810	0.495 + 248.4	1.553 + 8.325	3.891 + 256.9	7.812 + 353.3
10%	0.499 + 3.812	38.78 + 407.1	1.524 + 14.35	16.19 + 550.3	33.92 + 782.7
25%	8.951 + 21.71	112.4 + 972.5	9.459 + 92.42	60.30 + 1567	2920 + 5244

SMT execution time $= x + y$, x : The execution time of the SAT engine of the SMT Solver,
y : The execution time of the theory engine of the SMT Solver

to the SAT solver. In the second phase, the SAT solver attacks the circuit satisfiability problems augmented with these additional CNF clauses on valid key combinations, and make a new round of calls to the SMT solvers. As illustrated in Fig. 3.5(a), the invocation of theory and SAT solver, and the overall SMT attack is serialized. Accordingly, in order to reflect our experimental results for evaluating of Eager approach, we separate the execution time of the theory solver and that of the SAT solver.

Table 3.3 captures the results of the Eager SMT attack for different ISCAS-85 benchmarks with different obfuscation overhead. The theory execution time indicates the time required by graph theory to find all possible and valid key combinations (where only one of them is valid). Similarly, SAT execution time demonstrates the time taken by SAT solver to find a valid key, given the additional theory solver generated constraining clauses. As illustrated in this table, the SMT attack, in all cases is concluded and reported the correct key. The result of the pure SAT attack is not reported, as it always produces the wrong key for being oblivious to the DLL key values. Hence, the SMT solver in this respect extends the attack capability by means of including various theory solvers.

Note that the execution time of the SAT solver (the x value in each column of reported data in Table 3.3) depends on the (1) size of the circuit, and (2) the percentage of obfuscated cells. Hence the circuit c7552, for being larger than c1908 has a longer SAT attack time across all percentage obfuscation points. In addition, the increase in the SAT attack time is only slightly super-linear (close to polynomial) with respect to an increase in the degree

of obfuscation. On the other hand, the execution time of the theory solver (the y value in each column of reported data in Table 3.3) depends on (1) the number of input, (2) the number of outputs, and (3) the degree of obfuscation. Hence, a circuit with a larger number of IOs has a longer execution time for its theory solver, but the execution time is bounded by O(NM), with M and N being the number of inputs and outputs respectively. This indicates that the run-time of theory solver (unlike the MILP-based attack that was suggested in [31]) does not exponentially increase with respect to a number of timing paths in a netlist, as it only depends on the number of IOs and not the total number of timing paths.

3.10.3 Evaluation of Lazy SMT Attack

The Lazy approach of SMT attack, as illustrated in Fig. 3.5(b), uses the SMT solve function to simultaneously solve the theory and circuit SAT problem. In this approach, the theory model is defined but is not solved. In many applications, the Lazy approach outperforms the Eager solution. In addition, there are situations, where the Eager solution faces exponential runtime if solved separately. As an instance, SRCLock [28] focus on posing exponential runtime on pre-processor needed for detection of cycles, Hence, the Eager approach is not even applicable. However, the parallel invocation of the theory and SAT solver, and the resulting literal exchange, and the additional constraints posed on the solver could result in a significant reduction in the time needed to explore the problem's decision tree, and removes the need to complete the pre-processing before starting the SAT attack. Hence, if the execution time of the theory solver poses a runtime beyond acceptable, the problem could only be attacked by the Lazy SMT approach.

Table 3.4 shows the Lazy SMT attack execution time on ISCAS-85 benchmarks that were obfuscated using the process that was explained in the previous section (mixing 50% DLL+ 50% IOLTS). Considering the SAT and theory solver are invoked simultaneously, we have a single execution for the entire SMT problem, and unlike the Eager approach, we cannot separate the execution time of the theory solver and the SAT solver. As illustrated,

Table 3.4: Execution Time of SMT Attack in the Lazy Mode (Attack Mode 3).

Circuit	c432	c499	c880	c1355	c1908	c2670	c3540	c5315	c7552
1%	0.033	0.177	0.263	0.567	0.466	20.44	0.983	11.53	13.07
2%	0.049	0.262	0.325	0.676	0.596	21.86	3.443	11.76	17.83
3%	0.065	0.329	0.350	0.877	0.723	23.39	2.436	15.27	19.04
5%	0.049	0.340	0.517	1.085	1.456	28.87	2.587	38.87	45.96
10%	0.204	0.503	1.195	5.622	3.334	83.06	6.712	94.80	319.6
25%	0.599	1.481	2.036	297.2	95.67	2706	126.3	552.8	8045

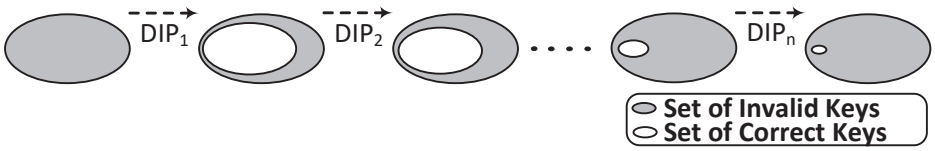
Table 3.5: Comparing the AccSMT (Attack Mode 4) with the Original SAT Attack.

Circuit	c2670				c3540				c5315				c7552			
	SAT		AccSMT		SAT		AccSMT		SAT		AccSMT		SAT		AccSMT	
	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time
1%	3	0.102	2	0.316	10	0.513	3	0.185	9	0.405	2	0.163	11	0.577	3	0.374
5%	45	1.514	11	3.589	19	1.502	6	0.761	32	1.354	6	0.408	67	5.271	17	2.607
10%	312	14.08	26	5.817	36	1.782	11	1.236	59	3.798	12	1.753	97	15.82	19	4.721
25%	781	114.5	107	24.05	77	9.796	16	1.606	95	19.63	27	7.916	215	225.6	24	23.52

in comparison with the Eager approach, in most cases the Lazy approach finds the key obfuscation key in a shorter time.

In the Lazy approach, the number of iterations decreases drastically compared to the Eager approach. However, the execution time of each iteration increases. This is because each DIP needs to satisfy both the theory constraints and the circuit SAT formulation. However, when a DIP is found, it is a stronger DIP with higher pruning power.

By comparing the results of Eager and Lazy approach of SMT attack in Table 3.3 and Table 3.4 we observed that in a majority of cases, the Lazy approach outperforms the Eager approach. However, in some cases (e.g. for Benchmark C1908 with 50% overhead), the Lazy approach may become slower than the Eager approach, indicating that the Lazy approach doesn't always result in a stronger attack. However, note that there exist a set of problems (such as SRCLock [28]), that the Eager approach is not even applicable, since the pre-processing step cannot conclude in a reasonable amount of time, leaving the Lazy approach as the only solution forward.



Complete Set of DIPs = {DIP_1, DIP_2, DIP_3, ..., DIP_n}

Figure 3.10: Potentially Valid Keys Reduced in Each Iteration of SMT/SAT Attack.

3.10.4 Evaluation of Lazy AccSMT Attack

Before invoking the SMT or SAT attack, any key could be considered as a *potentially valid key*. The strength of a DIP comes from its ability in reducing the size of this set in each iteration. After finding each DIP, as illustrated in Fig. 3.10, the size of *potentially valid key* set reduces. When reaching a complete set of DIPs, any key left in this set is a correct key. As discussed in section 3.9, a stronger DIP could sensitize a larger number of inconsistencies (due to the application of a discriminating input and two different keys) to the primary outputs. Hence, it is natural for such a DIP to have a higher pruning power in reducing the number of *potentially valid keys*. To evaluate this claim, we profiled the number of *potentially valid key* after each iteration of SMT and SAT attack, when working on the same obfuscation problem. Fig. 3.11 illustrates the key reduction rate in three ISCAS-85 benchmarks obfuscated by RLL [16]. In all scenarios, the DIPs found by the AccSMT solver are stronger, as the number of remaining keys is reduced at a significantly higher rate. As illustrated, the number of iterations is also significantly reduced because the complete set of DIPs, when the pruning power of DIPs is higher, is of smaller size.

The stronger DIPs found by the AccSMT attack, result in a significant reduction of the number of DIs needed for a complete discriminating input set. Each DI is found in one iteration, Hence, a smaller number of DIs indicates a smaller number of iterations. Table 3.5 compares the execution time and the number of iterations between the SAT solver and the AccSMT solver. The ISCAS-85 benchmarks for this simulation are obfuscated using RLL [16] obfuscation scheme with the overhead of 1% to 5%. As reported in this table, across

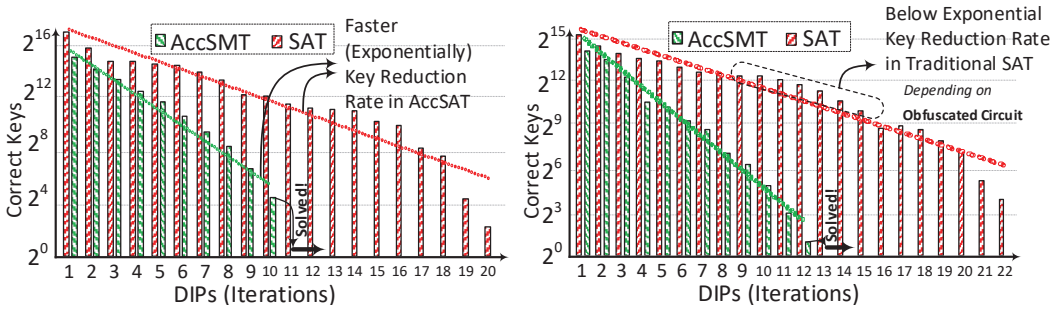


Figure 3.11: Key Reduction Rate in SAT Attack and AccSMT Attack.

all attacks, the AccSMT attack is carried in a smaller number of iterations and requires order(s) of magnitude smaller execution time.

As described in section 3.9.3, the AccSMT attack is able to distinguish between SAT-hard (SH) and high-corruption (HC) obfuscation. It quickly finds the correct keys for HC obfuscation, detects the SH trap, exits, and reports the approximate key.

To evaluate the approximate mode of the AccSMT attack, we have obfuscated the ISCAS-85 benchmarks using *SARLock + IOLTS14* as suggested in [22]. The overall structure of the obfuscated circuit is illustrated in Fig. 3.8. In this hybrid obfuscation scheme, the SARLock is the SH obfuscation, and the RLL is the HC obfuscation protocol. The invocation of the original SAT attack in [1,2] results in a timeout, due to the SARLock trap. However, the AccSMT can very quickly find all the keys for HC obfuscation, detect the SH trap, and report the approximate key. Table 3.6 depicts the number of iterations and execution time of the AccSMT attack for finding the approximate keys for each instance of the obfuscated circuit under attack. Note that repetition count (R=20 in our case study) is excluded from this table.

Table 3.6: Execution Time and the Number of Iterations of AccSMT (Attack Mode 4).

Circuit	c1908		c2670		c3540		c5315		c7552	
	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time
1%	7	0.512	16	3.075	8	1.304	3	0.384	7	2.905
5%	18	0.701	25	11.91	15	1.681	11	1.707	33	17.56
10%	31	4.085	51	26.47	21	3.779	35	7.402	61	44.07
25%	71	8.605	105	76.8	66	22.91	56	16.64	88	58.32

3.11 What we Learnt in this Chapter

In this Chapter, we introduced the SMT attack that could break behavioral locking techniques by means of invocation of theory solvers apart from the SAT solver, to model the non-logical and behavioral aspects of a circuit operation. We illustrated that even using these non-logical properties for obfuscation, does not increase the security of an obfuscated netlist, indicating the need for further study and exploration in this domain to generate obfuscation schemes with provable security.

Chapter 4: NNgSAT: Neural Network guided SAT Attack on Logic Locking

In this Chapter, we propose a **neural-network-guided SAT attack** (*NNgSAT*), in which we examine the capability and effectiveness of a message-passing neural network (MPNN) for solving the complex structures (SAT-hard instances), such as big multipliers, large routing networks, or big tree structures. In NNgSAT, after being trained as a classifier to predict SAT/UNSAT on a SAT problem (NN serves as a SAT solver), the neural network is used to guide/help the actual SAT solver for finding the SAT assignment(s).

4.1 Hard SAT Instances in Logic Obfuscation

The strength of the SAT solver comes from their Conflict-Driven Clause Learning (CDCL) ability. In each iteration of the SAT attack, a new SAT problem will be created, and the goal of the SAT solver is to find a satisfying (SAT) assignment for each SAT problem (per each iteration). The SAT problem is represented in conjunctive normal form (CNF) consisting of clauses, and each clause consists of one (a few) literal(s). The SAT solver tries to either assign or derive the value of each literal, and each assignment of value to a literal pushes the solver down into one of the branches of its decision tree. The decision tree of the SAT solver is built based on Davis–Putnam–Logemann–Loveland (DPLL) tree that is a complete backtracking-based search tree used for deciding the satisfiability of the propositional logic formula. The traversal on the DPLL tree is based on a recursive DPLL algorithm leading to the finding of a SAT assignment (The DIP in each SAT attack iteration).

The SAT attack (or one of its derivatives) faces a challenging problem when a design contains any form of *hard-to-be-solved* instances. As a case of hard SAT instances, which brings difficulties for the SAT solvers, we could list deep or symmetric or tree-based structures,

particularly while these structures are only built using the same basic gate, such as large multipliers, combinational systolic array modules, hierarchical routing blocks, big *AND*-tree structures, etc. [1, 32, 33, 41]. Having such structures sends the corresponded CNF far away being *under/over constrained*, and when the SAT problem is a medium-length CNF, it brings difficulties for the SAT solver. Fig. 4.1 shows the number of recursive DPLL calls[1] for fixed-length 3-SAT CNFs, where the ratio of clauses to variables is varied from 2 to 8 [33, 58]. As demonstrated, when the SAT problem is a medium-length CNF (clauses to variables ratio from 4 to 6), it requires more DPLL calls than *under/over constrained* CNFs (> 6 or < 4).

As discussed in Chapter 2, this observation has been the motivation for techniques such as Cross-Lock and Full-Lock [32, 33]. In these techniques, the main block used for obfuscation is a key-programmable routing block, which helps to build an extremely large medium-length CNF with thousands of variables. Since it faces millions of DPLL calls for each iteration of the SAT attack, it exponentially increases the runtime of each iteration of the SAT attack. It is surprisingly clear that when a design has such *hard-to-be-solved* instances, there is no chance for any of the existing attacks to break them, which motivates us to propose this new NN-based attack.

The question we aim to answer in this Chapter is whether we can train and use a neural network to predict a satisfying assignment to the literals of the SAT-circuit CNF and all constraints corresponded to each iteration of the SAT attack, particularly when the search tree is extremely deep. The goal is to save time and speed up the SAT attack. In cases the SAT solver cannot find the DIP for an iteration, we examine the effectiveness of a message-passing NN (MPNN) for predicting the DIP. This question is the motivation for the formulation of our proposed NNgSAT attack.

[1]DPLL algorithm is a recursive-based function that is the main part of the SAT solver for finding the satisfying assignments.

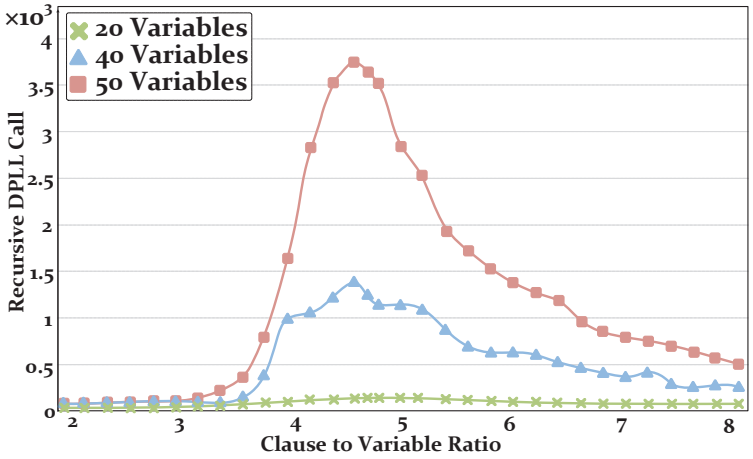
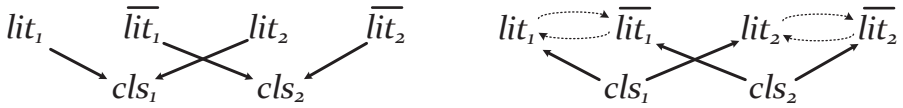


Figure 4.1: The Impact of Clause-to-Variable Ratio on DPLL Calls.

4.2 Neural Network Learns the SAT Solving

Glimpsing the progress of the SAT community starting around 1992 shows that before 2015 there were the most important conceptual advances resulting in the modern SAT algorithm based on CDCL. However, since 2015, the performance improvements has declined significantly [65]. On the other hand, the substantial ever-increasing the usage and application of the neural network (NN) [66–70] on important problems raises a big question that *"can a NN learn and improve the performance of the SAT solving?"*. More recently, a study proposes the first NN-based architecture that is designed for satisfiability problems, called *NeuroSAT* [71]. NeuroSAT is a message-passing neural network (MPNN) that learns to solve SAT problems after only being trained as a classifier to predict satisfiability. In NeuroSAT, it is shown that a trained NN on only toy problems could solve a bigger problem on its own. Also, based on the iterative structure of MPNN, more iterations at test time leads to solving bigger and even completely different domains than the problems it was trained on. For each problem, NeuroSAT starts guessing *UNSAT* at the early stages with low confidence until it finds a solution, at which point it converges and return the satisfying assignment with very high confidence. The training phase of the NeuroSAT relies on a single bit of supervision, in which the only difference between a pair of SAT/UNSAT problem



(a) (b)

Figure 4.2: NeuroSAT Operations on $(lit_1 \vee lit_2) \wedge (\neg lit_1 \vee \neg lit_2)$.

is one bit.

Since the SAT solver accepts the problems in CNF format, and since the SAT problem has a syntactic structure that could be encoded into a vector space, the best way to encode a SAT problem using a NN is to model it using an undirected graph. In this graph, there is one set of nodes each represents a literal, and one more set of nodes each represents a clause. For edges, there is a set of edges between each literal and each clause it appears in, and also there is another set of edges between each literal and its complement. Fig. 4.2, shows a simple example how a SAT problem $((lit_1 \vee lit_2) \wedge (\neg lit_1 \vee \neg lit_2))$ could be represented with an undirected graph with aforementioned rules. Nodes on the top represent each of the four literals, and nodes on the bottom represent each of the two clauses.

Assuming that this graph-based representation will be used for building the architecture of the NN, it could be parameterized by simply two vectors related to literals/clauses $(\mathbf{L}_{init}, \mathbf{C}_{init})$. Also, considering that a message-passing approach is used for the NN, the multi-layer perceptrons (MLP) could be $(\mathbf{L}_{msg}, \mathbf{C}_{msg}, \mathbf{L}_{vote})$, and as a special form of recurrent NN (RNN), there exists two layer-norm long short-term memory (LSTM) networks for literals and clauses $(\mathbf{L}_u, \mathbf{C}_u)$ [72]. Considering n_v as the number of variables and n_c as the number of clauses in a SAT problem, to model it to be solved using the iterative structure of the MPNN, at every time step t, the NN has a matrix $L^{(t)} \in \mathbb{R}^{2n_v \times d}$ whose i^{th} row contains the embedding[2] for the literal l_i and a matrix $C^{(t)} \in \mathbb{R}^{n_c \times d}$ whose j^{th} row contains the embedding for the clause c_j, which are initialized with $\mathbf{L}_{init}$ and $\mathbf{C}_{init}$ respectively.

Using this formal definition, a single operation of voting (one iteration of message passing) consists of applying the following two updates, in which M is the (bipartite) adjacency

[2]The row corresponding to a clause/literal referred to as the embedding of that clause/literal.

matrix defined by $M(i,j) = 1$ if $\{l_i \in c_j\}$, $Flip$ is the operator that takes a matrix L and swaps each row of L with the row corresponding to the literal's negation, and both $L_h^t \in \mathbb{R}^{2n_v \times d}$ and $C_h^t \in \mathbb{R}^{n_c \times d}$ as the hidden states for $\mathbf{L}_u$ and $\mathbf{C}_u$ respectively, which initialized to zero matrices:

$$(C^{t+1}, C_h^{t+1}) \leftarrow \mathbf{C}_u\big([C_h^t, M^\top \mathbf{L}_{msg}(L^{(t)})]\big) \tag{4.1}$$

$$(L^{t+1}, L_h^{t+1}) \leftarrow \mathbf{L}_u\big([L_h^t, Flip(L^{(t)}, M\mathbf{C}_{msg}(C^{(t+1)})]\big) \tag{4.2}$$

In this model, each iteration consists of two stages. First, each clause receives messages from its neighboring literals and updates its embedding based on the current embeddings (Fig.4.2a). Next, each literal receives messages from its neighboring clause as well as from its complement and updates its embedding based on the current embeddings (Fig.4.2b). By using this scheme, after T iterations, it could compute $L_*^{(T)} \leftarrow \mathbf{L}_{vote}(L^{(t)}) \in \mathbb{R}^{2n_v}$, which is a single scalar representation of the vote for each literal, and then computes the average of the literal votes $y^{(T)} \leftarrow mean(L_*^{(T)}) \in \mathbb{R}$. This vote could be used as a parameter to show the confidence rate (CR) of the prediction provided by the NN.

Fig. 4.3 provides a better representation to understand how iterative-based MPNN could be used to predict and guess the satisfying assignment for a SAT problem with a scalar representation of the vote (CR). It illustrates the sequence of literal votes for 24 iterations $L_*^{(1)}$ to $L_*^{(24)}$, $\big(L_*^{(T)} \leftarrow \mathbf{L}_{vote}(L^{(t)})\big)$, as the NN runs on a SAT problem. As shown in Fig. 4.3, to clarify the voting of each literal, $L_*^{(T)}$ is reshaped to be an $\mathbb{R}^{n_v \times 2}$ matrix so that each literal is paired with its complement (the i^{th} row contains the scalar votes for x_i and $\bar{x_i}$). As shown in Fig. 4.3, for iterations at the early stages, almost every literal is voting *UNSAT* with low confidence (light blue). Then, a few scattered literals start voting *SAT* for the next few iterations. However, it is not dominant to affect the mean vote. In most cases with a sudden change, there is a phase transition, and all the literals (and hence the network as a whole) start to vote *SAT* with very high confidence (dark red). After

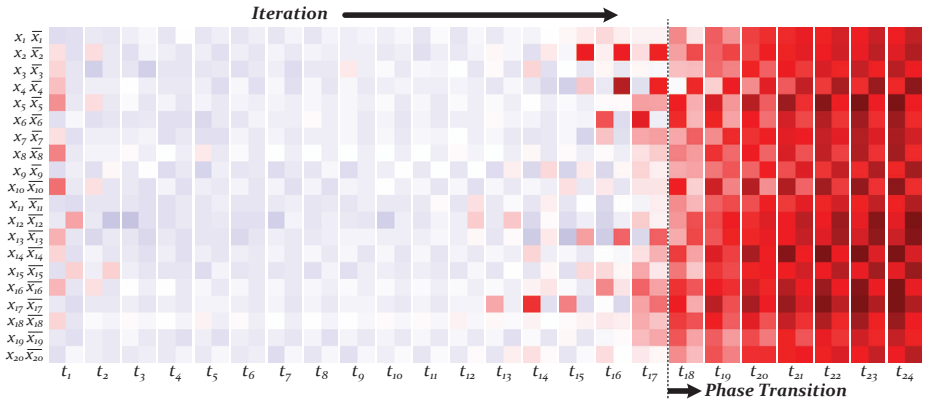


Figure 4.3: The Sequence of Literal Votes in the MPNN
Adopted from [71].

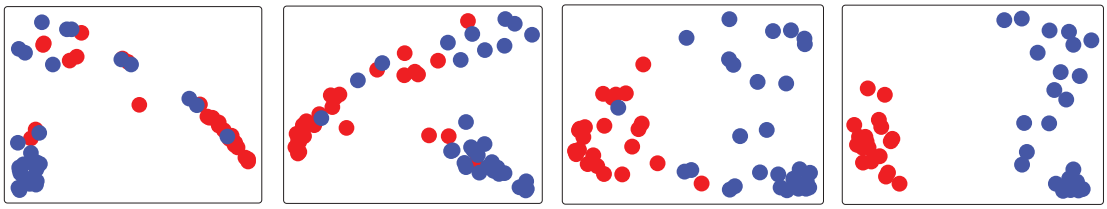


Figure 4.4: Extracting the Satisfying Assignment using 2-Clustering K-means.

this phase transition, the vote for each literal converges, and the network does not need to continue evolving.

As shown in Fig. 4.3, most of the variables have one literal vote distinctly darker (higher confidence rate) than the other. Also, the dark votes have all approximately the same color tone, and the same color tone could be seen for light votes. Based on this distinguishable coloring, it turns out that there exists a meaningful relationship between the satisfying assignment and these patterns (Darker votes are *1*s and Lighter ones are *0*s). However, to have a more reliable decoding solution, it could be translated using a 2-clustering mechanism calculated by the k-means algorithm [73]. As shown in Fig. 4.4, applying 2-clustering on the literal votes in each iteration (L^t) helps to distinguish between 0s and 1s to correctly extract the satisfying assignment (blue and red dots denote literals set to 0 and 1, respectively).

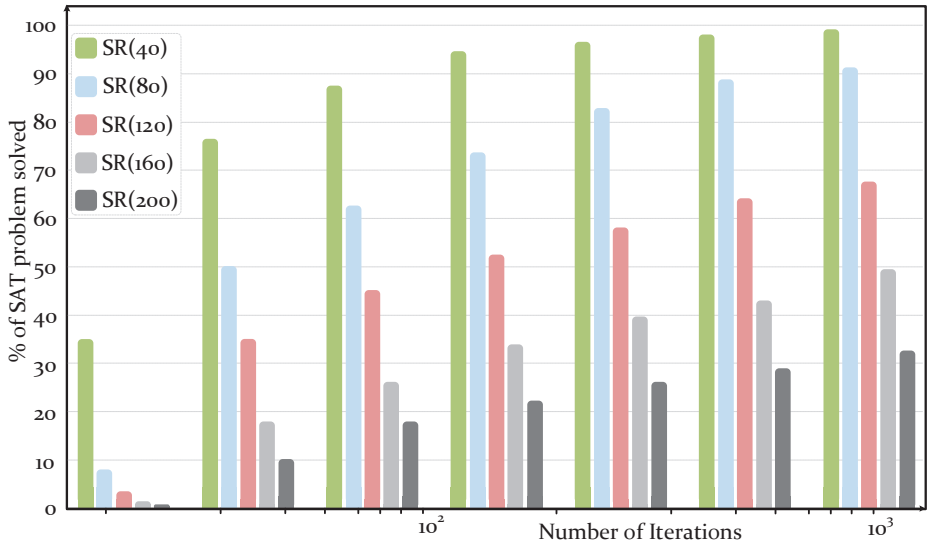


Figure 4.5: NeuroSAT's success rate on $SR(n)$.

A key observation of the NN usage for solving SAT problems is that it can solve SAT problems that are far larger than the models used during training. This is when the NN runs more iterations of message passing leading to find the satisfying assignment. As an example, Fig. 4.5 shows the success rate of the NN on $\mathbf{SR}(n)$[3] for a range of n as a function of the number of iterations T. The NN is only trained on $SR(U(10, 40))$ (the number of variables n is uniformly sampled from between 10 and 40 during training), however, as shown in Fig. 4.5, even though it is trained on $SR(40)$ and below, it solves SAT problems sampled from $SR(n)$ for n much larger than 40 by simply running for more iterations.

The observation that NeuroSAT can solve problems that are substantially larger and more difficult than it ever saw during training (by simply running for more iterations), motivates us to engage an MPNN to examine the effectiveness of this form of the solver for de-obfuscation purpose, particularly while there exists *hard-to-be-solved* structures in the locked circuit.

[3] $\mathbf{SR}(n)$ is a distribution over pairs of random SAT problems on n variables, in which one element of the pair is satisfiable, the other is unsatisfiable, and the two differ by negating only a single literal occurrence in a single clause.

4.3 NNgSAT:A NN guides The SAT Attack

As its name implies, in NNgSAT, getting inspired from NeuroSAT, a *message passing neural network (MPNN)* has been engaged and trained on the specific SAT problems obtained from logic locked circuit to be used as a guide for the SAT solver within the SAT attack. Given the SAT attack algorithm illustrated in Algorithm 1 in Chapter 2, to get the benefit of the MPNN, after being trained using the generated data set(s), the MPNN-based SAT solver is called in parallel with the actual SAT solver per each SAT iteration. Fig. 4.6 provides an overview of the major steps in the NNgSAT attack. As can be seen, most steps are similar to the traditional SAT attack. However, per each iteration, after updating the CNF of *miter*+constraints, and adding the new double-circuit for finding the new DIP, both the SAT solver and MPNN-based SAT solver will be called in parallel to solve the updated CNF. Based on a pre-defined threshold time ($SATtime_{th}$), if the actual SAT solver could find the satisfying assignment before $SATtime_{th}$, the MPNN-based SAT solver will be skipped, and for the next step both solvers will be called again. However, in those cases that the SAT solver could not find the satisfying assignment within $SATtime_{th}$, a part of the predicted satisfying assignment by the MPNN-based SAT solver, which have the highest literal votes (CR), will be extracted as a new (guiding) learned constraint, to help the actual SAT solver for finding the precise satisfying assignment.

Since the MPNN-based SAT solver is called in parallel with the actual SAT solver, after $SATtime_{th}$, we assume that T iterations of the MPNN-based SAT solver is executed (T times of message passing). As shown in Fig. 4.6, in MPNN-based SAT solver, for a CNF with n_c clauses and n_v variables, set of clauses (C) and set of literals (L) will be initialized with n_c clauses and $2n_v$ variables (n_v variables + n_v negated variables). Then, similar to the NeuroSAT, for T iterations of message passing, C and L would be updated in two stages: (1) clause updating: based on the current embeddings of the literals it contains ($\forall c, c \leftarrow C_u(c, \sum_{l \in c} L_{msg(l)})$, (2) literal updating: based on the current embeddings of the clauses it occurs in, plus the current embedding of its negation ($\forall l, l \leftarrow L_u(l, \sum_{l \in c} C_{msg}(c), \bar{l})$. Also,

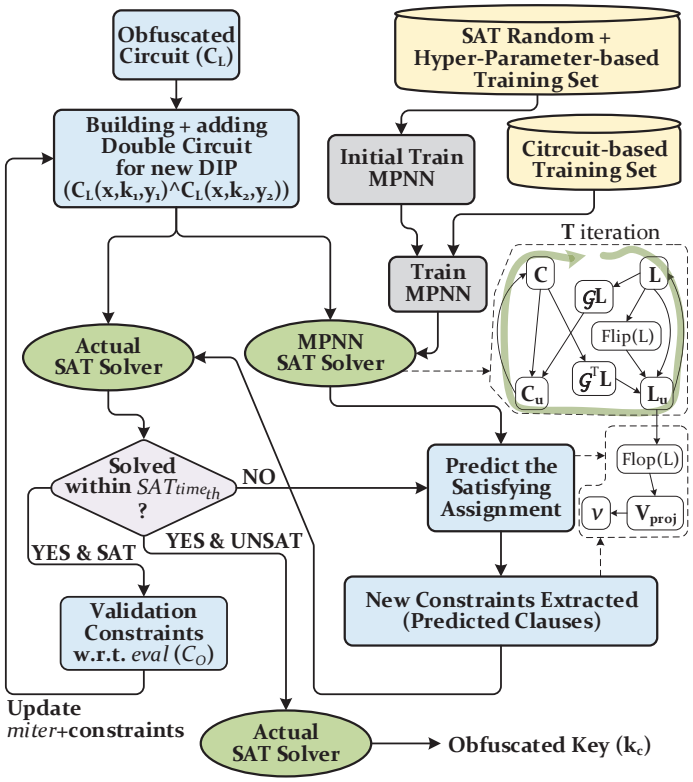


Figure 4.6: The Major Steps of NNgSAT Attack.

to have better convergence, a $n_c \times 2n_v$ sparse matrix $\mathcal{G}$ has been used, in which $\mathcal{G}_{i,j}$=1 if and only if the i^{th} clause contains the j^{th} literal. Similar to NeuroSAT, the $Flip(L)$ function swaps the first half of the rows of a matrix (L) with the second half. Also, to acquire better projection, after T iterations, the NN flops L to proceed the prediction. The $Flop(L)$ function concatenates the first half of the rows of a matrix with the second half along the second axis. Then, the flopped L followed by a projection (V_{proj} as an MLP, to project into an n_v-dimensional vector for completing the prediction (scalar vote)).

In NNgSAT, the number of message passing operations (the MPNN iterations) depends on the value of $SATtime_{th}$ and scalar vote (CR). We assumed that when we reach at time=$SATtime_{th}$, T iterations have been done in the MPNN. However, it might be possible that after T iterations of the MPNN, the prediction is $UNSAT$ or even SAT with low CR.

Hence, to avoid misguiding the SAT attack, more iterations are also required in MPNN-based SAT solver. Hence, to not lose the chance of solving by the actual SAT attack and to continue using the MPNN, we do not stop any of the solvers (either actual or MPNN) after $SATtime_{th}$. Instead, we define a few thresholds for the scalar vote (CR) in the MPNN. When we reach each threshold, we extract the prediction provided by the MPNN, and use those variables that have the highest scalar vote (CR) as a guided constraint. Then, we run a new actual SAT solver learned by the *miter*+constraints extracted from all previous iterations (successfully done), as well as guided by the MPNN. Hence, based on the pre-defined confidence thresholds, a few instances of the actual SAT solver would be run in parallel after $SATtime_{th}$, each is guided by different constraints predicted by the MPNN with different CR. Amongst all actual SAT solvers executing in parallel, the first actual SAT instance that returns *SAT*, will be used as the solution for the current SAT iteration, all others will be skipped.

Figs. 4.7, 4.8, and 4.9 show three different scenarios in NNgSAT w.r.t. the parallel SATs. In Fig. 4.7, after $SATtime_{th}$, the CR is less than the first (minimum) CR threshold (vt_1). So, we continue running both to see which solver reaches the next state. In this case, even before reaching the first CR threshold (vt_1), the actual SAT was able to find the result. So, we skip the MPNN, and the SAT attack goes to the next SAT iteration. In Fig. 4.8, after $SATtime_{th}$, the CR is still less than the first (minimum) CR threshold (vt_1). However, before finding the SAT assignment by the main actual SAT solver, the MPNN reaches the first CR threshold (vt_1). When the MPNN reaches a CR threshold, it generates a prediction as a SAT assignment, and variables with the highest CR (Those variables with CR_{bit} >90%) will be extracted as a set of the learned clause. Then a new actual SAT solver (the second instance) will be started guided by the extracted learned clause acquired by the MPNN. Now, two actual SAT solvers and MPNN execute simultaneously, and in this case, after a while, the second SAT instance that was guided by the MPNN were able to find the SAT assignment far sooner, and two other instances (the first actual SAT solver and MPNN) will be skipped. Similar scenarios are demonstrated in Fig. 4.9, where the third

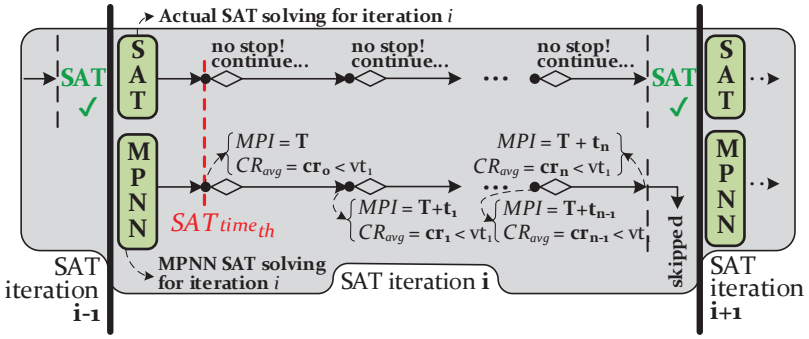


Figure 4.7: Parallel Execution in NNgSAT with Different Scalar Votes - Scenario 1.

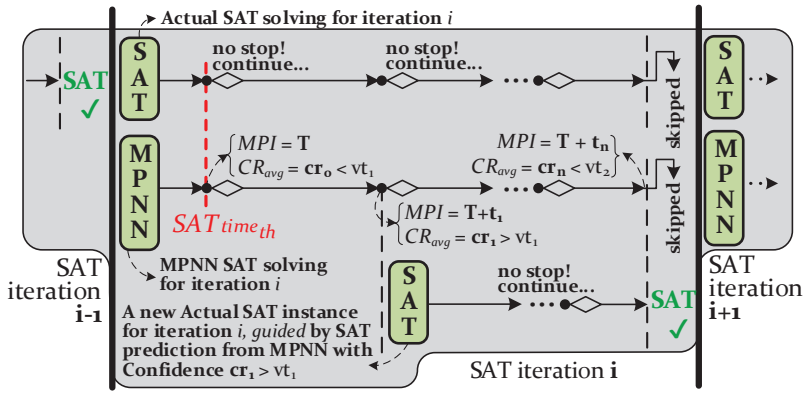


Figure 4.8: Parallel Execution in NNgSAT with Different Scalar Votes - Scenario 2.

SAT solver returns faster.

It should be noted that in the MPNN-based SAT solver, there is no dependency between the learned parameters and the size of the SAT problem. However, since the SAT attack iteratively adds a new double circuit, as well as a lot of learned clauses found in the previous iteration, the size of the SAT problem, would be increased extremely, which makes it very memory-intensive. Hence, to query the MPNN with more scalability, building the sparse matrix $\mathcal{G}$ would be based on a limited number of non-eliminated variables and clauses. Although we would like to pass all clauses/variables to the MPNN, in many cases, the problem would not fit into the available memory. Hence, we traverse the learned clauses in ascending size order, collecting part of them that does not exceed a fixed cutoff. Also,

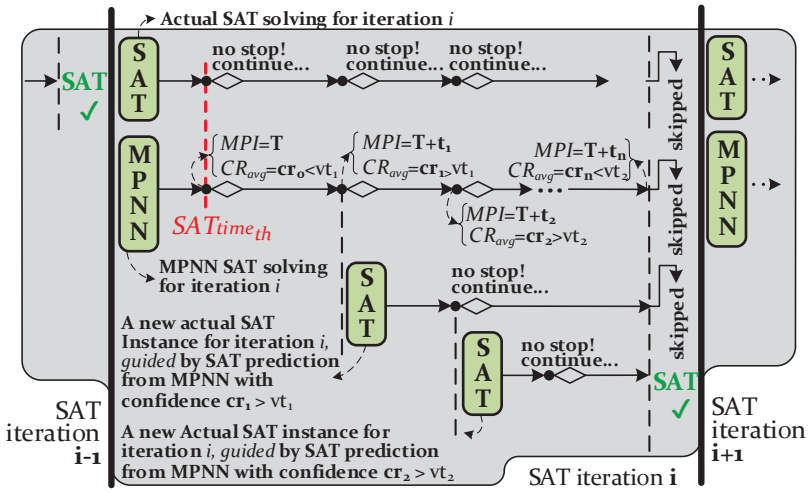


Figure 4.9: Parallel Execution in NNgSAT with Different Scalar Votes - Scenario 3.

we only query the MPNN on random subsets of the clauses for problems that exceed the cutoff.

Since both SAT/UNSAT will happen in each SAT attack, the MPNN should be able to predict UNSAT as well. To help construct proofs for UNSAT problems, the MPNN that has been trained on a large dataset, in which every unsatisfiable problem contains a small contradiction, learns to detect these contradictions instead of searching for satisfying assignments. Then, the variables involved in the contradiction can be extracted and enable constructing a resolution proof more efficiently. The NN cannot predict that a SAT problem is UNSAT with high confidence, and it almost never guesses SAT on an UNSAT problem. It only predicts SAT, once it has found one satisfying assignment. Hence, it could be considered as a certificate of satisfiability.

4.3.1 Training Dataset Generation

With extensive analysis, we observe that our proposed NN-based SAT attack works best if the NN is trained in two phases. First, the NN needs to be initially trained using simple and small random SAT problems with single-bit supervision to learn how distinguishing between SAT and UNSAT. Also, with a minor change compared to the NeuroSAT, as illustrated in

Table 4.1: Specifications of the Modified (_m) Benchmark Circuits.

Circuit	# PIs	# POs	# Gates	# Mults	# Crossbars	# LUTs	# AND-trees
c2670_m	287	161	5,793	1 of 12×14, 1 of 18×18	1 of 24×20	4×LUT-6, 2×LUT-8	2×AND-16
c3540_m	87	47	7,605	2 of 7×10, 1 of 20×16, 1 of 18×22	1 of 18×15, 1 of 20×16	4×LUT-5, 3×LUT-8	2×AND-15, 1×AND-26
c5315_m	206	152	10,851	3 of 11×14, 1 of 12×22	2 of 19×14, 1 of 36×30	6×LUT-4, 2×LUT-6, 2×LUT-7	1×AND-32
c6288_m*	85	74	7,378	1 of 20×25	3 of 9×9, 1 of 12×15	2×LUT-8, 1×LUT-9, 2×LUT-12	4×AND-22, 1×AND-52
c7552_m	224	121	12,722	1 of 12×18, 1 of 22×26	2 of 15×15, 1 of 30×36	3×LUT-6, 1×LUT-15	2×AND-15, 3×AND-27
b14_m	289	308	20,407	2 of 8×12, 2 of 12×14, 1 of 36×34	2 of 26×22	2×LUT-11	2×AND-16, 1×AND-44
b15_m	488	526	19,348	1 of 29×25	1 of 30×32, 1 of 36×36	2×LUT-11, 1×LUT-16	3×AND-33, 1×AND-36
b17_m	1452	1512	47,972	2 of 17×21, 1 of 18×22	1 of 26×22, 1 of 32×30	4×LUT-6, 2×LUT-12	2×AND-42
b20_m	527	516	38,856	1 of 8×12, 1 of 32×26	1 of 24×26, 1 of 60×64	2×LUT-11, 1×LUT-14	1×AND-12, 1×AND-48
b22_m	767	757	35,658	1 of 12×14, 2 of 11×11, 1 of 14×18	2 of 17×14, 1 of 24×22	4×LUT-5	1×AND-16, 4×AND-29

*: c6288 contains 16×16 multiplier by itself.

Fig. 4.6, the network is not only fine-tuned using single-bit supervision, but it also followed by training with a single set of hyper-parameters as a coarse heuristic that broadly assigns higher CR to selected variables [74].

Then as the second phase of the training, the NN is trained specifically for a set of only small-size circuits that contain small-size of the complex structures, listed as follows:

1. $n \times m$ bit-wise multiplier built by *AND/XOR* trees ($m, n \leq 8$).

2. $n \times m$ crossbar network built by 2-to-1 MUXes ($m, n \leq 16$).

3. $n-$input look-up-tables (LUT) built by 2-to-1 MUXes ($n \leq 8$).

4. n-to-1 *AND*-tree structures, built by *AND2* (binary-tree).

We insert these structures into the circuit as a part of the original circuit, as well as a part of the obfuscation module (key-programmable crossbars or LUTs). The sizes that are selected for this phase of the training must be small enough to make them solvable by the traditional SAT attack. Also, to maximize what is learned by the network from these special structures, we engage circuits as simple as possible. To do that, we add these structures into a circuit that only contains wires connecting outputs to the inputs. In fact, this dataset only consists of these structures.

4.4 NNgSAT Performance Evaluation

To investigate the performance and effectiveness of NNgSAT, we selected and used large circuits from ISCAS-85 and ITC-99 benchmark suite, as summarized in Table 4.2. For the neural network, we used *NeuroSAT* with minor changes in training and message passing calculations described in Section 4.3. To compare the results of NNgSAT with the actual SAT attack, we use the traditional SAT attack [1]. We also use MiniSAT as the SAT solver of both traditional SAT and NNgSAT. The timeout has been set to $2{\times}10^5$ seconds ($\sim$ two days). All experiments have been done on a Dell PowerEdge R620 equipped with 28-core Intel Xeon E5-2670 2.50GHz and 64GB of RAM.

For the training phase, we built two different training datasets. First, as initial training, to force the network to learn problems substantive, we define a distribution $\mathbf{SR}(n)$ over pairs of random SAT problems, where one element is SAT, and the other is UNSAT, and they differ by negating only a single literal occurrence in a single clause. After fine-tuning over these pairs, it is also trained on a single set of hyper-parameters as a coarse heuristic that broadly assigns higher CR to selected variables, and training is done using ADAM optimizer [75] with a constant learning rate of 10^{-4}. Second, we built 20,000 obfuscated circuit samples (based on the small sizes from Section 4.3.1), each obfuscated using key-programmable LUTs/crossbars and solved by the actual SAT solver. Then, we train the NN based on the output of the actual SAT solver per each iteration.

To show the effectiveness of NNgSAT on complex structures, we modify the selected benchmark circuits by embedding some large-size complex structures into the design. To test the efficacy of the NNgSAT in finding satisfiable assignments in real-circuits, we then embedded larger-than-trained structures into the raw benchmark circuits as follows:

1. $n \times m$ bitwise multipliers built by *AND/XOR* trees ($8¡m, n¡32$).

2. $n \times m$ crossbar network built by 2-to-1 MUXes ($16¡m, n¡36$).

3. $n-$input look-up-tables (LUT) built by 2-to-1 MUXes ($n¡16$).

Table 4.2: Specifications of the Raw Benchmark Circuits

Circuit:	c2670	c3540	c5315	c6288*	c7552	b14	b15	b17	b20	b22
# of Inputs	233	50	178	32	207	277	485	1452	522	767
# of Outputs	140	22	123	32	108	299	519	1512	512	757
# of Gates	1,193	1,669	2,307	2,416	3,513	9,014	8,367	36,770	19,682	29,162

*: c6288 is a 16×16 multiplier by itself.

4. n-to-1 *AND*-tree structures, built by *AND2* (binary-tree).

By using these structures, we built new and modified benchmark circuits to be evaluated using NNgSAT, as described in Table 4.1. As shown, for each design, 1,2, or 3 multipliers with different sizes, 1, or 2 crossbars, few LUTs, and *AND*-tree(s) have been inserted to be considered as a part of the original design. It is worth mentioning that in a few cases, particularly for smaller circuits, since the wiring is limited, we had to add extra primary inputs/outputs into the design to provide the required nets. A comparison between the original circuits listed in Table 4.2 and modified (_m) ones listed in Table 4.1 shows that some of the modified circuits have more PI/PO. Furthermore, these modules are embedded with the highest conservation to avoid emerging any form of incompatibilities, such as avoiding the creation of nonsense logic function, bypass logic, and combinational cycles. It should be noted that the size of these structures must be large enough to be considered as *hard-to-be-solved* instances. On the other hand, the size must be small enough to keep the overhead less than the acceptable overhead threshold (e.g. %5-%10)[4].

After building the modified benchmark circuits, one more step is the obfuscation to be assessed by NNgSAT. For obfuscating the benchmark circuits, we also used the key-programmable form of these structures. We insert large-size LUTs[5] and key-programmable crossbars with different sizes. More precisely, crossbars with size 18×16, 24×18, and 36×30, as well as LUTs with 10 and 12 inputs are added into the design.

[4]Although these structures incur up to 5× overhead in the benchmark circuits, in the real applications, such as well-known microprocessors, the overhead of these structures would be less than %5.

[5]For LUTs, the configuration bits are assumed as the key inputs.

Table 4.3: Execution Time Comparison between NNgSAT and the SAT Attack.

Circuit:	c2670_m		c3450_m		c6288_m		c7552_m	
Attack:	NNgSAT	SAT	NNgSAT	SAT	NNgSAT	SAT	NNgSAT	SAT
1C18×16, 4LUT10	1607.8	3858.6	1205.7	1205.7	3055.7	timeout	2490.5	2490.5
2C18×16, 2LUT12	7207.4	timeout	4184.7	timeout	3112.9	timeout	8679.2	timeout
1C24×18, 4LUT10	2494.7	12029.1	4284.5	4284.5	1285.1	timeout	2570.7	16372.8
2C24×18, 2LUT12	5670.8	timeout	5991.6	5991.6	2976.1	timeout	7708.4	timeout
1C36×30, 4LUT10	6850.7	23074.5	5573.9	timeout	2403.3	timeout	6189.5	21688.8
2C36×30, 2LUT12	5034.7	timeout	6408.3	timeout	3606.6	timeout	11684.8	timeout
2C18×16, 2C24×18	7920.7	timeout	6270.8	18604.4	1967.2	timeout	5608.5	12685.8
2C18×16, 2C36×30	8134.8	timeout	6303.1	timeout	3007.2	timeout	6122.7	timeout
2C24×18, 2C36×30	8507.3	timeout	7190.8	timeout	4622.9	timeout	8642.8	timeout
Circuit:	b15_m		b17_m		b20_m		b22_m	
Attack:	NNgSAT	SAT	NNgSAT	SAT	NNgSAT	SAT	NNgSAT	SAT
1C18×16, 4LUT10	4661.8	4661.8	2207.4	timeout	2764.9	11706.6	7294.7	timeout
2C18×16, 2LUT12	13212.9	timeout	10766.2	timeout	12647.1	timeout	7554.6	timeout
1C24×18, 4LUT10	7206.4	12473.8	5622.1	timeout	9223.7	timeout	timeout	timeout
2C24×18, 2LUT12	11382.4	timeout	10684.5	timeout	10700.5	timeout	14382.8	timeout
1C36×30, 4LUT10	7681.1	timeout	9034.8	timeout	9927.8	timeout	11042.1	timeout
2C36×30, 2LUT12	4896.4	timeout	7221.5	timeout	10082.8	timeout	11894.7	timeout
2C18×16, 2C24×18	6672.3	timeout	4956.7	timeout	5501.2	timeout	5974.6	timeout
2C18×16, 2C36×30	9501.8	timeout	6079.3	timeout	6442.6	timeout	timeout	timeout
2C24×18, 2C36×30	timeout	timeout	12820.3	timeout	timeout	timeout	10941.5	timeout

* timeout = 2×10^5 Seconds.
+ n_cC$m\times n = n_c$ crossbars with size $m \times n$ controlled by the key.
+ n_lLUT$n = n_l$ LUTs with n inputs configured with the key.

Table 4.3 illustrates the average execution time of the traditional SAT attack [1] and our proposed NNgSAT on obfuscated benchmark circuits. As shown, in many cases, the original SAT attack fails to extract the satisfying assignment(s) within the allowed time threshold when a large-size of such complex structures are in place. However, after being trained over single-bit supervision and hyper-parameters and expanding the training over only small-size complex structures, NNgSAT could break almost all obfuscated circuits within a few hours. Also, NNgSAT is affected far less by the size of the circuit or the size of the obfuscation module, showing that correctly being trained, the network could also improve itself for especially large circuits that never saw during the training. Besides, unlike accelerated SAT solutions (e.g. parallelism of SAT) that linearly improve the performance, in almost all cases, the NNgSAT improvement is super-linear.

In some cases, the execution time of the traditional SAT attack and that of NNgSAT

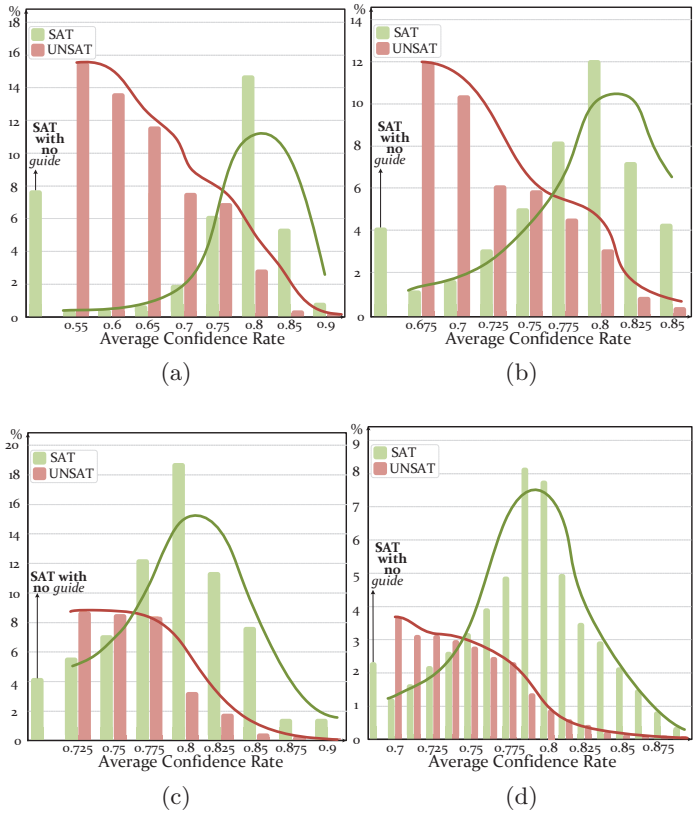


Figure 4.10: The Impact of Confidence Rate on the Success of MPNN predictions.

is equal. This shows that in a non-negligible part of SAT iterations, the network might not provide useful guidance for the actual SAT attack. Hence, when the execution time is identical, it means that none of the network's predictions was helpful for the SAT attack for that case and within N SAT iterations. Also, since the neural network guidance is prediction-oriented, due to misguiding that will happen in a few cases, regardless of the size of the circuit, the performance of NNgSAT might be varied.

This observation shows that the predictions could be categorized based on their effectiveness: (1) *guiding prediction* that correctly guides its corresponding SAT solver to find a SAT assignment, (2) *misguiding prediction* that incorrectly guides its SAT solver to an *UNSAT*, (3) *skipped prediction* that are skipped/terminated because one of the parallel SAT solvers on this problem found a solution. The ratio of each category is dependent on the

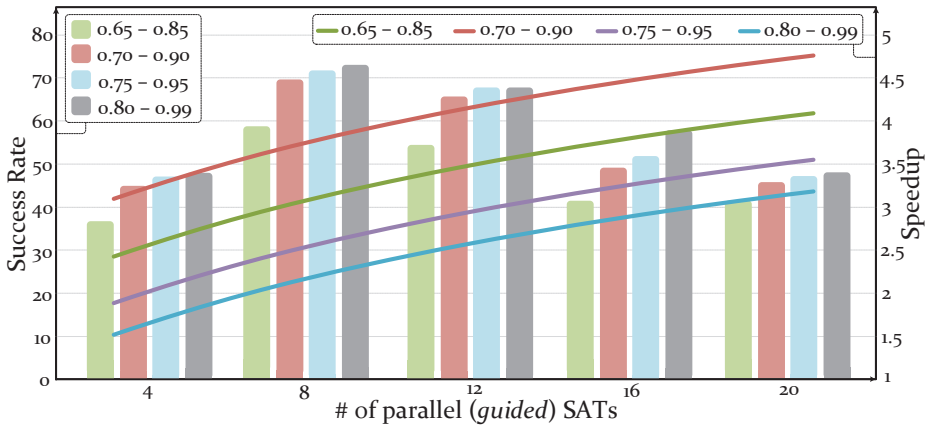


Figure 4.11: SAT Parallelism Efficiency on Success Rate and Speedup of NNgSAT.

CR we choose for each run. Fig. 4.10 shows the distribution of two first categories[6] based on different values of CR. As shown in Fig. 4.10(a, b), when we start the prediction with lower average CR, the rate of misguiding prediction is considerably higher. The predictions help more when the average confidence ratio is 0.7< CR <0.9. Hence, as shown in 4.10(c, d), to get the most benefit of the network, CR must be 0.7< CR <0.9.

As we discussed previously, based on the CR, a different number of parallel SAT solvers guided by the MPNN would be executed. To examine the impact of the number of parallel SAT executions on the performance of NNgSAT, we compare the NNgSAT runtime for different numbers of parallel executions over different CR ranges. Fig. 4.11 shows that when we set the CR on a higher range (e.g. 0.8-0.99), as discussed previously, it requires more message passing to guide the SAT. Hence, the performance of the higher ranges will be decreased. However, when we use more cores, it could help to improve the speed-up. On the other hand, for higher ranges, the success rate is not considerably higher. For CR 0.8-0.99, the success rate is only 5% higher than that of CR 0.7-0.9. However, since for 0.7< CR <0.9, it is less restricted, the performance is significantly higher[7][8]. Hence, the

[6]Due to the parallelism, there are some SAT solvers per each iteration, in which only one might find the SAT assignment and all others must be skipped. Hence, the ratio of skipped prediction is far higher and omitted from Fig. 4.10

[7]For CR 0.65-0.85, although it is less restricted than CR 0.7-0.9, the performance is lower because in many cases, the prediction is misguided leading to UNSAT.

[8]All the results and the speedup reflected in Fig. 4.11 are for those cases in which both attacks, the

number of cores is set to 8, and CR is 0.7-0.9 to have the best tuning for the NNgSAT. Based on these observations, to gather results for Table 4.3, we used 8 cores (8 different CRs) and the CR range is set to 0.7-0.9.

4.5 What we Learnt in this Chapter

In this Chapter, we proposed a neural-network-guided SAT attack, called NNgSAT attack, which uses a Message Passing Neural Network to predict satisfying assignments that could help and significantly speed up the conventional SAT attack in solving the design that contains complex *hard-to-be-solved* structures. Our experimental results showed that NNgSAT could de-obfuscate 93.5% of the logic circuits locked with or contained complex structures within a reasonable time, while the original SAT attack fails to de-obfuscate them.

traditional SAT and NNgSAT, successfully de-obfuscate the locked circuits. This cannot be used as the total speedup of NNgSAT vs. the traditional SAT because for many cases NNgSAT solves problems that were not solvable by the traditional SAT, and in these cases, the speedup cannot be calculated.

Chapter 5: Data Flow Obfuscation: A new Paradigm for Obfuscating Circuits

Comparison of the state-of-the-art logic locking techniques show that a reliable logic obfuscation technique must provide *TWO* main features: (1) the logic locking technique must be resilient against different attacks, including both combinational and sequential SAT attack; (2) it should be added without compromising the test flow. Table 5.1 summarizes almost all state-of-the-art logic locking techniques, each suffering from a big shortcoming. As we discussed in Chapter 2, although numerous logic locking countermeasures have been introduced after the SAT attack, many of them are vulnerable to newer attacks, e.g. removal, bypass, and functional analysis attack. Since the SAT attack is only applicable to combinational circuits (sequential circuits with open access to the scan chain), many recent studies have investigated the possibility of blocking/obfuscating the scan chain. However, these solutions inflict significant limitations, such as compromising test flow. In this Chapter, we introduce data flow obfuscation [76], which could be considered as a new logic locking paradigm. In data flow obfuscation, by benefiting from the handshaking mechanism of asynchronous circuits, the system's FFs/latches will operate out of sync. So, the sequential SAT attack is no longer applicable to a data flow obfuscated circuit. Also, due to the inherited asynchronicity, the exact time of writing/capturing data into/from the scan chain becomes hidden. Hence, the SAT attack cannot be applied even while scan chain access is open.

5.1 Combinational De-obfuscation

In Table 5.1, for categories known as *pre-sat*, *point function*, *cyclic and behavioral*, and *routing-based obfuscation*, an important threat model assumption is that the attack model

Table 5.1: Comparison of State-of-the-art Logic Obfuscation Techniques.

Category	Defense	Corruptibility	Attacked by	Overhead	Test/Implementation Issues		
					Limitation	Test Complexity	Test Time
Pre-SAT	RLL [16]	high	Sensitization [18], SAT [1]	low	NO change	NO change	low
	FLL [37]		SAT [1]				
	SLL [18]		SAT [1]				
Point Function	SARLock [22]	low	Different Attacks [24, 25, 40]	low	NO change	NO change	low
	Anti-SAT [21]	low	Different Attacks [24, 25, 40]	low			
	SFLL [23]	variant	FALL [26]	moderate			
	+ Primitive (Compound)	high	Different Attacks [24, 27, 39, 40]	low			
Cyclic and Behavioral	Cyclic [43]	high	CycSAT [44], icySAT [46]	high	NO change	NO change	low
	SRCLock [28]		SMT [77]				
	DLL [31]		SMT [77]				
Routing Obfuscation	Cross-lock [32]	very high	NNgSAT [78]	very high	NO change	NO change	low
	Full-Lock [33]		NNgSAT [78]	very high			
Blocked or Locked Scan Chain	EFF+RLL [47]	high	Different Attacks [3, 53, 54]	low	NO change	NO change	low
	R-DFS+SLL [5]		Different Attacks [48, 53, 54]	moderate	test coverage	NO change	low
	MSSD+RLL [48]		Shift&Leak [48, 49]	moderate	test coverage	key init	high
	DynScan+SLL [57]		Scan Unlock [3, 4]	high	trusted tester	NO change	low
Data Flow Obfuscation		high	NO Attack	low	NO change	NO change	low

is *oracle-guided*. In an oracle-guided attack model for ***combinational de-obfuscation***, as previously described in Chapter 2, the adversary has access to an unlocked/activated chip (oracle) with open scan chain access, as well as the reverse-engineered yet locked netlist. In the SAT attack, for any arbitrary *obfuscated* combinational logic (c_{comb_lock}), by getting inspiration from the *miter* circuit used in formal verification, a (distinguishing) *miter* circuit has been built as $miter \equiv c_{comb\_lock}(dip, k_1) \neq c_{comb\_lock}(dip, k_2)$, which returns a specific *discriminating input pattern* (dip) that produces different output for two different keys k_1 and k_2. Then, this dip is queried on the oracle, c_{comb}, $eval \leftarrow c_{comb}(dip)$ and the I/O-constraint $c_{comb\_lock}(dip, k_1) = c_{comb\_lock}(dip, k_2) = eval$ is stored back in the SAT solver and the *miter* circuit would be solved again. When the $miter + constraints$ problem has no longer satisfying assignment, it could identify the correct key.

5.2 Sequential De-obfuscation

Since the SAT attack is only applicable when the access to the scan chain is open, the studies in *blocked or locked scan chain* category evaluate the security of primitive logic obfuscation

techniques [16, 79] while the access to the scan chain is blocked/locked. In this case, the adversary has only access to the PI/PO, and PO would be a function of PI and the state of the circuit, which makes it impossible for the SAT attack to formulate it at once.

As previously described in Section 2.6.12 of Chapter 2, to still exploit the combinational SAT attack while the scan access is restricted, few recent studies have engaged *unrolling* or *BMC* as a pre-processing step to formulate the sequential obfuscation using the combinational SAT attack [3, 4, 53, 54]. As shown in Fig. 5.1, the adversary unrolls the sequential circuits τ times. A τ-time unrolled circuit is an equivalent combinational model of a sequential circuit for τ clock cycles. It takes in τ input patterns (as a sequence), and produces τ outputs, while the intermediate states are cascaded [1]. After unrolling, similar to the combinational de-obfuscation, the SAT attack would find the sequences of inputs $(i_0, i_1, i_2, ..., i_{\tau-1})$, called *distinguishing input sequence* (dis) with two different keys k_1 and k_2 such that the outputs $(o_0, o_1, o_2, ..., o_{\tau-1})$ will differ. Every time the unrolled *miter* becomes unsatisfiable at some depth d (no more dis), the adversary extends the unrolling until a termination condition is satisfied. Termination conditions are **unique completion (UC)**: when there is only one key that satisfies the I/O-constraints for unrolled circuit (correct key); **combinational equivalence (CE)**: where the transition function is combinationally equivalent between two duplicated circuits with two keys (k_1 and k_2) (shadow key), and **unbounded model check (UMC)**: if a call to an unbounded model checker ($\tau = \infty$) concludes that the result is invariant in the reachable statespace [53, 54].

The point is that the unrolling step relies on the synchronicity of FFs in the obfuscated sequential circuit. When the sequential circuit is synchronous, moving forward from any arbitrary clock cycle to the next one (cycle $t \rightarrow t+1$) updates the FFs only once at positive (negative) edges of the clock signal. Hence, in the unrolling step, the combinational parts would be replicated only once per each clock cycle. But, for circuits and systems that *asynchronously* control the data flow in the circuit, the unrolling-based SAT or BMC faces

[1] Similarly, model checking could be used to check the satisfying assignment in a sequential (transition) system. A model checker with bounded depth corresponds to bounded model checking (BMC) and an unbounded one to unbounded model checking (UMC).

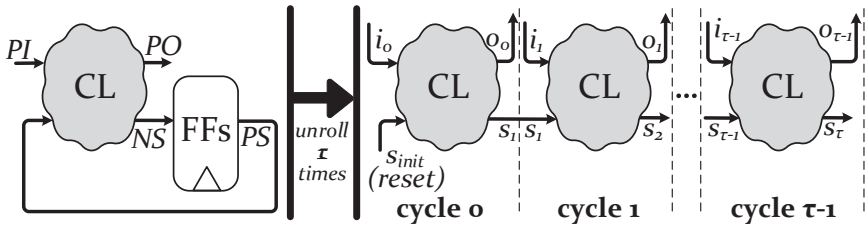


Figure 5.1: Sequential Circuit vs. its Combinational Counterpart (τ Cycles).

a big obstacle during the unrolling step to build the equivalent combinational model for a specific number of clock cycles.

5.3 Asynchronicity

To asynchronously control the data flow in a circuit (partially or fully), one could adopt the asynchronous circuit paradigm. The asynchronous circuits have multiple advantages over synchronous circuits, particularly for newer technology nodes, such as no clock skew problems, robustness towards process variations, as well as advantages in terms of power consumption and electromagnetic emissions [80].

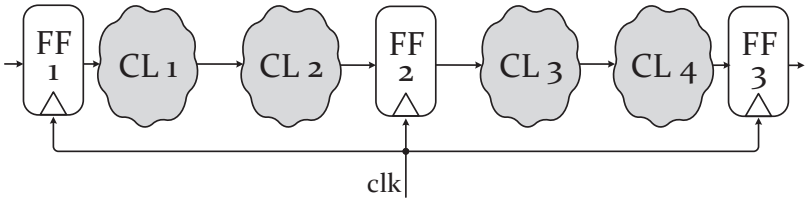
For two main reasons, most designers consider asynchronous circuits as a perilous approach: (1) the lack of electronic design automation (EDA) tools, and (2) opposition to change designers' mentality towards asynchronicity. However, the ever-increasing attention on these circuits results in introducing powerful synthesis and verification tools for asynchronous circuits [81–83]. It allows any designer to non-disruptively incorporate asynchronicity in an EDA flow, and there is no need for the designer to change the synchronous mentality/structure. As an instance, AnARM is an ultra energy-efficient asynchronous ARM processor that is successfully implemented and fabricated using the STMicroelectronics 28nm technology, using standard cells and conventional CAD tools while achieving a 59% improvement in energy when compared with the ARM Cortex-A7 [84]. As of today, widespread application of asynchronous circuits could be seen in IoTs, NoCs, mixed-signal circuits, etc. [85, 86].

5.4 from Synchronicity to Asynchronicity

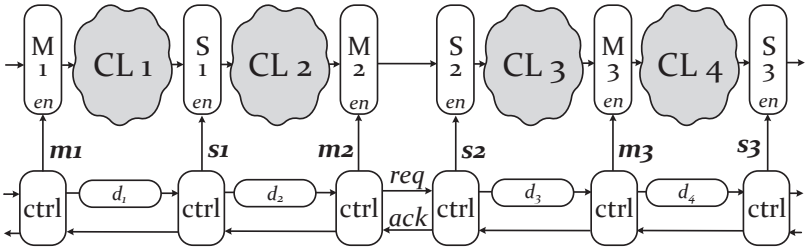
Signal transition graph (STG) is the formal specification of the asynchronous circuits, which is used in most asynchronous synthesis and verification tools [87]. The STG could be drawn from scratch by the designer based on the specification of the design. However, one could use the *desynchronization* paradigm that generates the equivalent asynchronous model of any synchronous circuit. By providing formal proofs of correctness based on the theory of Petri nets [88], the *desynchronization* [89] provides a fully automated flow for building the flow-equivalent asynchronous counterpart.

To build the flow-equivalent asynchronous model of any synchronous circuit using *desynchronization*, as shown in Fig. 5.2, after removing the clock signal, FFs would be replaced with master (M) and slave (S) latch pairs. All latch enable signals (en) must be controlled using new macros, called asynchronous latches controllers, which uses a handshaking structure (req, ack) to emulate FFs' behavior. For example, in four-phase handshaking, as the most prevalent handshaking protocol, $\phi 1$ is enabling req by a sender for the a valid data. $\phi 2$ is enabling ack by the receiver, acknowledging the arrival of the new data. $\phi 3$ is lowering (disabling) previous req, and finally $\phi 4$ is lowering the corresponded ack. Handshake signals are not related to a global clock and are based on the local, relative timing relationships between the opening and neighboring latch enable signals. Also, during *desynchronization*, delay elements must be added per each combinational logic (CL) to mimic the delay of all timing paths and asynchronous latch controller. Also, the first/last latches of the asynchronous part (m_1 and s_3 in Fig. 5.2b) that are dealing with other (synchronized) parts of the circuit will be handled by some controlling signals, e.g. a specific state of the circuit, or controlling signals like FFs enables.

It is also worth mentioning that all latches are operating based on their controllers. By using *desynchronization*, one latch will be enabled when tokens are ready, and will be disabled after receiving the ack corresponded to the new data (lowering req). It will prevent



(a) Synchronous Circuit



(b) Desynchronized Circuit

Figure 5.2: Synchronous to Asynchronous Conversion.

extra propagation when inputs of the latch change, which avoids increasing power consumption. Furthermore, latches controllers are the only added parts when the *desynchronization* is accomplished; However, we show that since the proposed solution is required to be accomplished on a small part of the circuit, it does not incur large area/resource overhead.

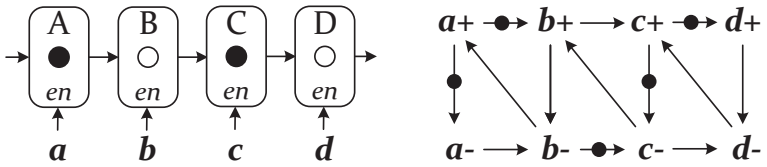
5.5 Desynchronization

Three main steps of the *Desynchronization*, which provides a fully automated methodology to build the flow-equivalent asynchronous model of any synchronous circuit are: (1) Converting FFs to M and S latches, with decoupled enable signals (e.g. in Fig. 5.2b FF_i is replaced with M_i and S_i whose controllers are m_i and s_i). (2) Matched delays generation for combinational logics (CLs), based on their timing path delays. (e.g. d_{1-4} are matched delay for FF_{1-4} to mimic the delay of their timing paths as well as the delay of each asynchronous latch controller (m_{1-3} and s_{1-3})). (3) Implementation of the asynchronous controller of each latch, e.g. *ctrl*s in Fig. 5.2b, based on the data flow dependencies in the original netlist.

In step 1, after replacing FFs with latches, re-timing is often used as a performance improvement technique [90]. By using re-timing, latches are moved across CLs (e.g. M_3 is placed before CL_4. S_1 is placed after CL_1 in Fig. 5.2).

In step 2, matched delay elements are generated for emulating the timing path delay of their corresponding CLs. These will be connected to corresponding controllers in the next step. In this step, the netlist is synthesized for the target cycle time T_T, using a conventional synthesis tool. The T_T is captured using $\{T_T \geq T_{CQ} + T_C + T_L\}$, in which the T_T is a delay between two rising edges of control signal of the latch, T_{CQ} is the delay of local clock propagation through a latch, T_C is the delay of the CL, and T_L is the latch controller delay. By using this inequality, and based on the delay of critical paths in each CL, these matched delays are generated. When T_Cs are equal in all CLs (balanced timing paths), then the separation time between adjacent rising edges of every local clock equals T_T. Also, in any desynchronized circuit, the i^{th} rising transition of a local clock cannot appear later than $(i-1) \times T_T$, showing that the temporal behaviors of the desynchronized circuits are also similar to synchronous counterpart [89].

Step 3 implements the asynchronous controller for each latch. These controllers are connected to the controllers of neighboring latches with the delay elements built during step 2. A variety of desynchronization models exist to implement these asynchronous controllers. The behavior of these models can be typically specified using STG, which is a decision-free subclass of Petri nets [88]. An STG, as shown in Fig. 5.3b, may be defined as a 3-tuple (Φ, $\rightarrow$, I_0), where Φ is the set of events, and events are the latch enable values (high/low) in asynchronous controllers. $\rightarrow$ corresponds to an arc, which illustrates event transitions, and for a latch controller, it determines changes in latch enable values. I_0 is the initial marking, called token, and denotes the initial event signal states. In *desynchronization*, it is crucial to properly define I_0, as the initial tokens, and it is fully dependent to handshaking protocol used for *desynchronization* [89]. Tokens determine which data is ready. In STGs, as in Petri nets, these tokens could be updated (moved) based on the interaction between different latches. For example, a signal is enabled when all its predecessor arcs are marked



(a) Four Cascaded Latches (b) Corresponding STG

Figure 5.3: Desynchronization Model for a Pipeline Structure.
"•" are tokens (ready data). "∘" are bubbles (not-ready data). Latches are transparent when "*en*"s are high. "***a+***" means rising transition, and "***a-***" means falling transition.

with a token. An enabled signal can fire, removing tokens from all its predecessors' arcs, and populating tokens to its successors' arcs. For instance, Fig. 5.3a shows a part of a pipeline with cascaded latches. Fig. 5.3b depicts an STG representing the behavior of these latches. Over the time, based on the location of data, tokens move around determining which latch will catch a new data. Without loss of generality, we use *semi-decoupled four-phase control* for handshaking [91], which represents a good trade-off between simplicity and performance, however, any valid desynchronization latch controller may be used instead [89].

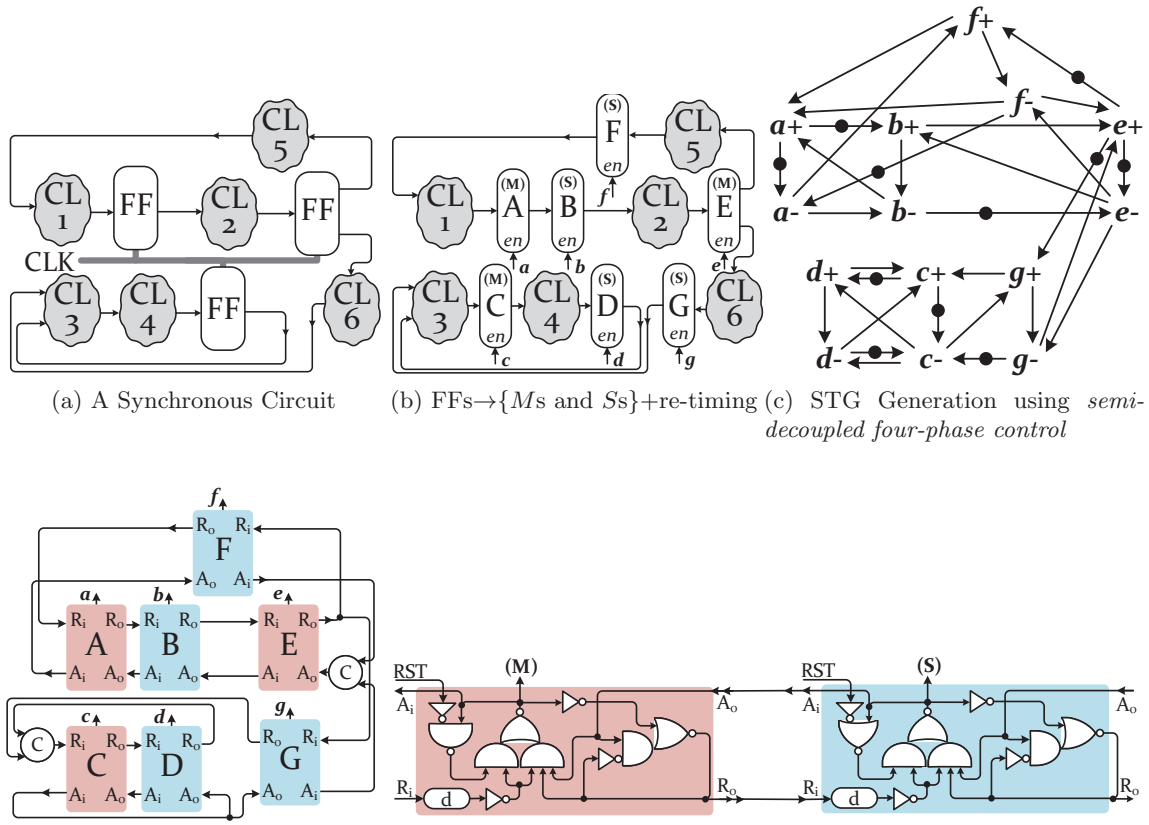
Based on the generalization of *semi-decoupled four-phase control*, there are four rules imposed on the latch control signals of the STG. Using these four rules, the designer can specify the corresponding STG for any circuit: (1) $a+ \rightarrow a-$: rising of each signal (each latch enable) should be followed by falling of that signal. (2) $b- \rightarrow a+$: For latch A (master) to read a new data, latch B (slave) must have completed the read of previous token from A. (3) $a- \rightarrow b-$: For latch B (slave) to complete the reading of a data token coming from A (master), it must first wait for latch A to complete the reading of that data token (4) $a+ \rightarrow b+$: For latch B (slave) to read a new data, it must wait for latch A to read that new data token.

Considering these four rules, we now illustrate an example of the desynchronization methodology based on *semi-decoupled four-phase control* for a small circuit. Fig. 5.4a is an arbitrary synchronous netlist with three FFs. In Fig. 5.4b, all FFs are replaced with latches subjecting to re-timing (e.g. CL_4 is placed between latches C and D). Since the

fan-out of rightmost FF in Fig. 5.4a is *two*, it is converted to one M, (E), and two Ss, (F and G). After converting FFs to latches, the corresponding STG is generated based on all four aforementioned rules (Fig. 5.4c). Then, based on the drawn STG, the corresponding asynchronous controller for latches enables are implemented. Considering that we use the *semi-decoupled four-phase control*, the circuit depicted in Fig. 5.4e must be engaged for each latch controller. It is latch controller (left (red) one for Ms and right (blue) one for Ss) based on *semi-decoupled four-phase control*. The handshaking signals between Ms and Ss are connected directly. However, for multiple dependencies in STG (i.e. one to many latches, or vice versa), the handshaking must be handled by merging *req* or *ack* signals respectively. This merge is performed using C-elements, which is an event-driven AND gate. A possible implementation is using the function $Z = AB + ZA + ZB$, where A and B are the C-element inputs, and Z is its output. For example, as shown in Fig. 5.4c, latch C is dependent on both D and G. So, in Fig. 5.4d, R_i of latch C is driven using a conjunction of R_os of D and G using a C-element.

After implementing the asynchronous latch controller, it must be connected to the circuit depicted in Fig. 5.4b to build the complete desynchronized counterpart of the synchronous circuit. It is proven in [89] that: (1) A *desynchronized* circuit never halts (*liveness* property); and (2) The sequence of data values of a *desynchronized* circuit are identical to its synchronous counterpart (*flow-equivalence*).

Also, by using *desynchronization*, physical design, verification, and design for testability can be accomplished "as is," using conventional synchronous EDA tools [89]. For instance, for design for testability, a low-frequency clock may be distributed to latches in test mode [92]. For testing the asynchronous controllers, as rising and falling of *req*s and *ack*s follow each other, in the presence of a stuck-at fault on either *req* or *ack*, either the environment or the circuit will wait forever, and cause a deadlock, which may be easily detected during design for testability. Circuits that have the property that they halt for all faults are called self-testing.



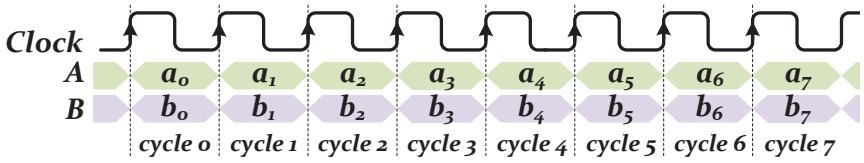
(a) A Synchronous Circuit (b) FFs→{Ms and Ss}+re-timing (c) STG Generation using *semi-decoupled four-phase control*

(d) Building the Asynchronous Controller based on its generated STG (e) Latch Controller (Left: Ms, Right: Ss) with matched delays (d)

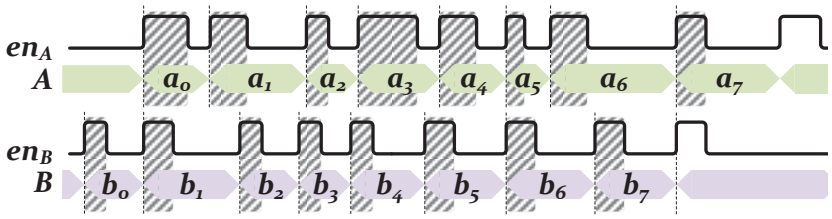
Figure 5.4: Top Illustration of Desynchronization Flow.

5.6 Concept of Data Flow Obfuscation

Since the sequential SAT attack relies on the synchronous unrolling mechanism, the preserved *flow-equivalency* after desynchronization, motivates us to propose a new obfuscation paradigm. In general, two circuits could be called flow-equivalent if there is no difference between the sequence of values stored at each latch. The observation is done independently for each latch. As an example of flow-equivalent circuits Fig. 5.5 demonstrates two flow-equivalent circuits. Fig. 5.5a, shows the synchronous behavior, while Fig. 5.5b shows the desynchronized behavior. Using this characteristic of asynchronous circuits, in section 5.8, we show how our proposed obfuscation technique could get benefit from this



(a) Synchronous Behaviour



(b) Asynchronous (Desynchronized) Behaviour

Figure 5.5: Timing Diagram of Synchronous vs. Asynchronous Circuits.

flow-equivalency concept to introduce a new obfuscation paradigm, called data flow obfuscation.

5.7 Threat Model

Similar to categories known as *pre-sat*, *point function*, *cyclic and behavioral*, and *routing-based obfuscation* in Table 5.1, we make the following assumption about the adversary capabilities: (1) The adversary can successfully do the reverse-engineering on the chip, and retrieve the gate-level netlist (yet locked). (2) The adversary can purchase an activated/unlocked chip (oracle) from the market. (3) The access to the scan chain of the oracle is not restricted. So, the adversary could apply any query to FFs using SI and read the updated values through SO after one clock cycle (*capture* mode).

5.8 Proposed *Data Flow* Obfuscation

As discussed previously, in all existing logic obfuscation techniques, the synchronicity is kept intact during manufacturing stages. However, due to the synchronicity, these techniques were vulnerable to unrolling-based SAT or BMC even while the scan chain is restricted.

It should be noted that, for the most potent attacks on logic obfuscation, there exists a big inspiration from a formal verification method such that the attack relied on and adopted from the verification method to successfully de-obfuscate circuits, e.g. miter circuit in traditional SAT [1], or BMC/unrolling in sequential SAT [53]. Hence, the main aim of this new obfuscation paradigm is to add ambiguity in a way such that it turns the obfuscated circuit into a completely new form that cannot be modeled using any of the existing formal verification methods. We target part of the data flow in a circuit to be obfuscated using asynchronicity. When the asynchronicity is used in a circuit, due to the high non-deterministic behavior, it is extremely challenging to come up with an automated approach to establish invariance properties, which are vital in proving the correctness of a circuit with asynchronous parts. There exist a few methods that ease the formal verification in asynchronous parts [93, 94], helping the designers to do formal verification for datapath of asynchronous circuits. However, to prevent any form of easing, in our proposed data flow obfuscation, we target to obfuscate the asynchronous controllers, which is the source of desynchronization with the self-testable feature. Also, since we assume that the scan access must be still fully open, the proposed obfuscation must be in a way that conceals the writing/capturing into/from the storage elements. Hence, in the proposed solution, the controller of latches is obfuscated such that without the correct key, the temporal characteristics of the datapath will be hidden.

5.8.1 How Data Flow Obfuscation Works?

The main steps of our proposed data flow obfuscation are:

(1) Converting the targeted synchronous part(s) of the circuit to its (their) flow-equivalent asynchronous counterpart using *desynchronization* described in Sections 5.4 and 5.5[2][3].

(2) Inserting *false paths* into the desynchronized circuit. Each false path could be an

[2]The conversion could be done fully (whole circuit) or partially (part(s) of the circuit). However, targeting and obfuscating only critical part(s) of the circuit (partially) guarantees the security at lower overhead.

[3]Re-timing will be done as a part of de-synchronization in data flow obfuscation. However, it might not change the location of latches (relocation). So, we need to apply a minor relocation for some latches to avoid any form of re-synchronization-based attack described in Sections 5.10.1 and 5.10.2.

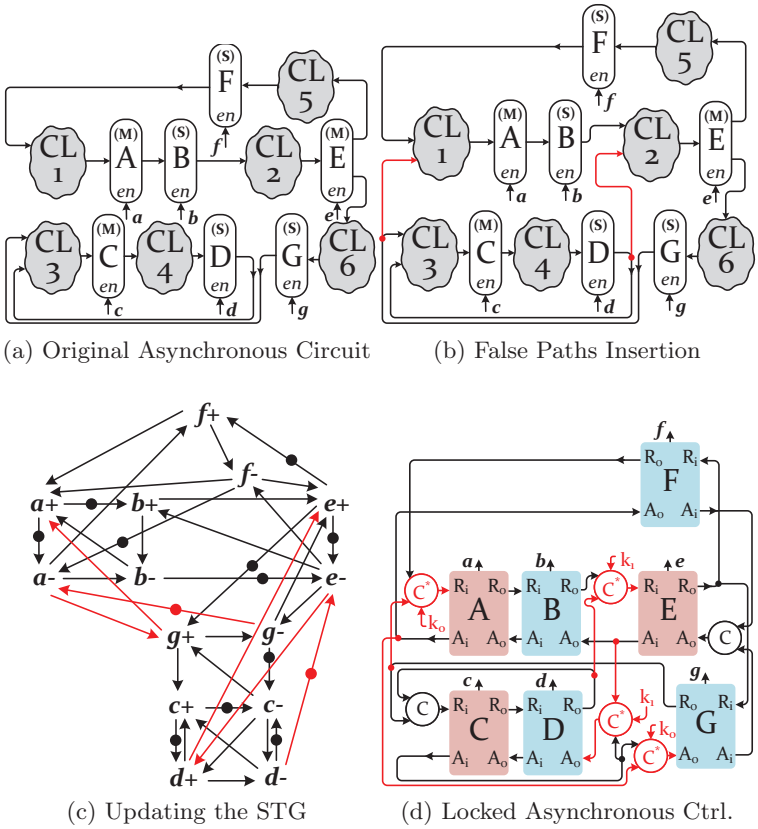


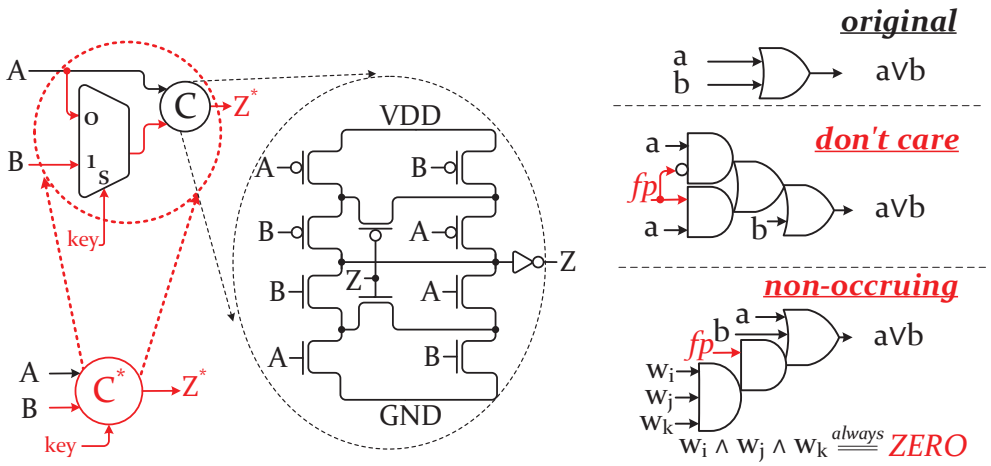
Figure 5.6: An Example of ***Data Flow*** Obfuscation with *only* False Paths.

extra wiring from the output of one latch to any arbitrary combinational logic.

(3) Updating the corresponding STG based on the added false paths. For each false path, few extra transitions with initial token must be added to the STG to reflect the changes.

(4) Obfuscating the asynchronous latches controller circuit (based on STG w.r.t. the new false paths) by using proposed C^*-element that is a key-controlled event-driven AND gate.

Fig. 5.6 demonstrates step-by-step implementation of the data flow obfuscation on the circuit from Fig. 5.4a. First, the targeted parts of synchronous circuit (Fig. 5.4a) are converted to their asynchronous counterpart (Fig. 5.6a). Then, in the desynchronized circuit a specific number of false paths are inserted (e.g. $D \rightarrow CL_2(E)$ and $G \rightarrow CL_1(A)$ in



(a) C*-element (b) Function Bypassing

Figure 5.7: Essential Modules for False Path Insertion.

Fig. 5.6b). Since the connectivity between latches is altered, the STG should be updated (Fig. 5.6c). Also, the changes must be reflected into the asynchronous controllers. For instance, before adding the false paths, F was the only predecessor of A. So, R_o of F was directly connected to R_i of A. However, after adding the false path $G \rightarrow A$, both F and G are the predecessors of A. Thus, a C-element must be added to merge their req signals. The C-element here implies that latch A may only be opened whenever data from both G and F are ready. However, for any false path like $G \rightarrow A$, it should have no impact on timing when the key is correct. To achieve this, we introduce a C^*-element, in which a key-controlled MUX is used to control the C-element's inputs. As shown in Fig. 5.7a, based on the key value, the C-element input is either $\{A, B\}$ or $\{A, A\}$, and based on the C-element's definition, if both inputs are the same, the output will be equal to the identical input pair: $Z = AA + ZA + ZA = A$, meaning that with correct key, the added false paths will have no timing effect.

As shown in Fig. 5.6, the only obfuscated part in data flow obfuscation is the usage of C^*-elements that alters the behaviour of controllers based on the key value. However, these C^*-elements only control the timing (behavior) of latches. Hence, regardless of the key value, each false path is connected directly to one arbitrary chosen CL, and could affect

its functionality. For instance, for false path $G \rightarrow A$, regardless of the key value (k_0), latch G would affect the functionality of CL_1, and the key value only controls the time of the act. To avoid this problem, the false paths can be connected to the chosen CL as *don't cares*, or *non-occurring* inputs. Fig. 5.7b shows these two models for a simple circuit. As shown, the output of all cases is the same, i.e. $a \vee b$, and the added false path *fp* does not affect the output in both cases. For instance, in *non-occurring*[4], *fp* is ANDed with $w_i \wedge w_j \wedge w_k$, which is always *ZERO*, and has no impact on the logic.

5.8.2 Shortcomings of False Path Insertion

With inserting *only* false paths, the data flow obfuscation is, however, vulnerable to re-synthesis and removal attack. Since the false paths are connected to corresponding CLs as *don't care* or *non-occurring*, they explicitly have no impact on the circuit's functionality. Hence, the attacker could re-synthesis the reverse-engineered netlist, and by using logic optimization effort during the synthesis, the false paths will be removed during optimization. Then, the adversary can find some extra elements/connections in the controller that have no corresponding part in the datapath (already removed). So, he/she can distinguish between the original parts and the extra logic added for the false paths in the controller and retrieve the original circuit. So, to combat this issue, we add one more step which adding extra false latches on false paths.

5.8.3 Adding False Latches on the False Paths

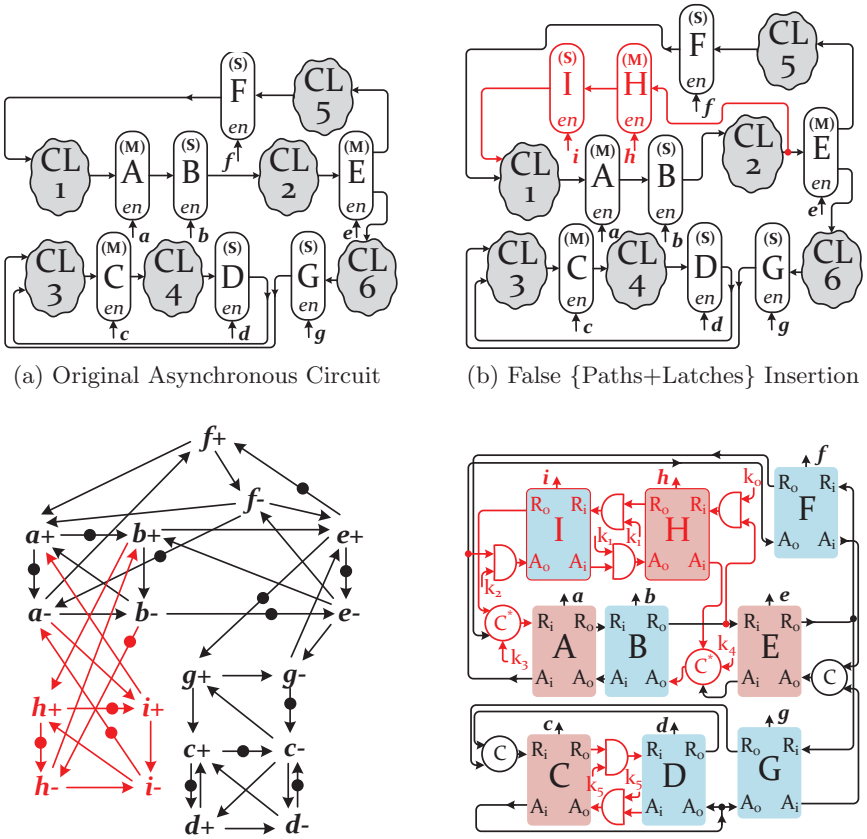
We updated and added one more step in our proposed data flow obfuscation to support adding false latches:

(1) (Same Step) *desynchronization* + re-timing (relocate).

(2) (Same Step) Inserting *false paths* into the circuit.

(3) **(New Step) Inserting *false latches* (M/S pairs) on the *false paths* + an asynchronous controller for each added *false latch* to control its behavior.**

[4] *non-occurring* cases can be found using SAT solver. First, few wires must be selected, then its condition clause must be added, and solved by SAT. If SAT solver returns *UNSAT*, that condition is *non-occurring*.



(a) Original Asynchronous Circuit (b) False {Paths+Latches} Insertion

(c) Updating the STG (d) Locked Asynchronous Ctrl.

Figure 5.8: An Example of ***Data Flow*** Obfuscation with False {Paths + Latches}.

(4) (Same Step) Updating the STG based on new insertions.

(5) (Same Step) Obfuscating the controller via C^* elements.

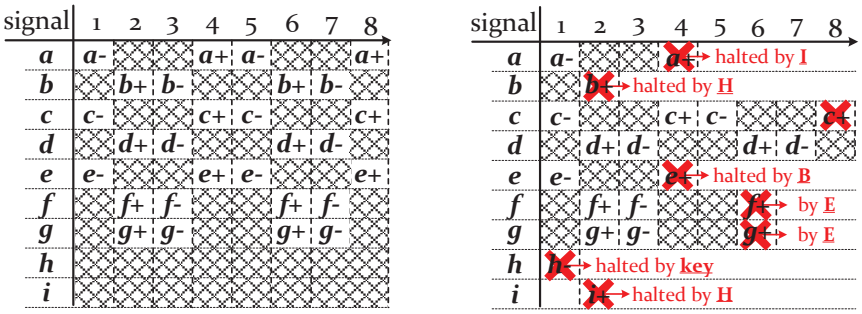
Inserting pairs of *false latches* in false paths allows us to control the logic value of these paths. So, there is no longer a need to add false paths as *don't care* or *non-occurring*, and the re-synthesis and removal attack is no longer a valid attack. The concept of insertion the false {paths + latches} is visualized in Fig. 5.8. Similar to the previous example, first, the circuit must be converted to its asynchronous counterpart (desynchronized). Fig. 5.8a shows the asynchronous model of our simple circuit from Fig. 5.4a. After that, one false path is added from $CL_2(B)$ to $CL_1(A)$, then a pair of master and slave latches (I and H), are added in this path (Fig. 5.8b). Based on these changes, the STG is updated (Fig.

5.8c). Compared to the STG of the previous example, not only new transitions are added, the STG has new nodes describing the events of new false latches. Finally, these updates must be reflected into the obfuscated asynchronous controller (Fig. 5.8d). Note that extra controller modules are added for false latches. Also, the Key-controlled gates must be added to properly control the behavior of latches H and I. When the key is correct, the added false path must have a value that does not affect the functionality of CL_1. For this purpose, the behavior of the false latches is controlled using k_{0-2}. So, while the k_{0-2} is correct (000 in this case), the AND gates mask the handshaking of H and I with their neighboring latches. Hence, these latches are disabled (and no new data will be captured in them). So, the initial value of these latches will be kept intact, and will be used as the new input of CL_1. In CL_1, based on the initial value of these latches, this false path will be connected to an arbitrary gate. (e.g., with initial value 0, it could be connected to an OR or XOR gate). However, while the key is not correct, the behavior of these latches would be changed repeatedly, resulting in corrupting the functionality of CL_1.

Additionally, false {paths + latches} must not affect their neighboring latches when the key value is correct (e.g. latches H and I must have no effect on A and B in Fig. 5.8d). This is achieved by adding two C^*-element before A and B, controlled by k_3 and k_4 respectively to effectively eliminate this temporal relation (e.g. in path $I \rightarrow A$, C^*-element skips the effect of the behavior of I on the behavior of A).

5.8.4 Key Classification

The keys added to asynchronous latches controllers, will be categorized into two main groups: (1) *handshake-in keys*: keys that control the impact of incoming signals to false latches (k_{0-2} in Fig. 5.8d), (2) *handshake-out keys*: keys that control the impact of outgoing signals from false latches (k_{3-4} in Fig. 5.8d). Based on the value of these two groups, different scenarios could happen: (1) **Correct Functionality**: While both groups are correct. In this case, similar to the timing diagram depicted in Fig. 5.9a, the firing of latches alternates appropriately. (2) **Halt in data flow**: While *handshake-in keys* are



(a) Original Circuit (b) Locked Desynchronized Circuit

Figure 5.9: Comparison of Timing Diagram of Latch Enables.

correct, but *handshake-out keys* are incorrect, halts will happen (e.g., if $k_{0-2} = 000$ and $k_{3-4} \neq 00$, H and I would halt). As shown in Fig. 5.9b, after latch H controller (h) is halted, more halts are happened in other paths and results in a complete deadlock in the whole circuit. (3) **Incorrect Functionality**: While the *handshake-in keys* are incorrect, regardless of the *handshake-out keys*, the function will be incorrect.

5.9 Security/Testability Analysis of *Data Flow* Obfuscation

5.9.1 Security versus the SAT Attack

Since the adversary has access to the scan chain in data flow obfuscation, he/she is able to apply combinational de-obfuscation for any accessible part of the circuit using the SAT attack. However, for two important reasons, the traditional SAT attack cannot be applied on data flow obfuscated circuit: (1) In the SAT attack, it is crucial to know the exact time of writing/capturing into/from the scan chain; But, in data flow obfuscation, this timing is controlled (locked) by an asynchronous controller. The adversary cannot determine when he/she must write into the scan, and when the updated data is ready to be observed. (2) Due to the nature of asynchronous controllers, the latch enable controller consists of many stateful cycles. The SAT solver works perfectly fine if the circuit is a *directed acyclic graph* (DAG), and only structural cycles could be analyzed using a pre-processing engine before

Table 5.2: Data Flow Obfuscation against Different Attacks on Logic Locking.

Attack	To break	The reason why this attack fails breaking the proposed data flow obfuscation
Traditional SAT [1]	Primitive Logic [16, 79]	It requires to write/capture into/from scan flip flops, but the exact time of writing/capturing into/from the latches (replaced with FFs) and that of neighboring FFs is hidden (locked) by the obfuscated asynchronous controller.
Removal [24]	SARLock, AntiSAT [21, 22]	When the added logic for locking is completely separated from the original circuit, it finds and removes the locking part to retrieve the original circuit. In the data flow obfuscated circuit, after removing all key-related parts from an asynchronous controller, extra latches+paths will remain active in the datapath affecting the original functionality.
Approximate SAT [27]	AntiSAT or SARLock + Primitive [24]	It guesses a key with low error rate when low and high-corruption techniques are compounded. But, since the data flow obfuscation is not added on a specific point, the output corruptibility is high, and in this case, the error rate of the approximate key will be high (useless).
FALL [26]	SFLL [23]	This attack is an upgraded version of removal attack combined with functional analysis, and specialized for SFLL (SFLL is an expanded version of SARLock). However, since the false {latches+paths} are not detectable in data flow obfuscation, after applying removal on data flow obfuscated circuit, the functional analysis cannot deal with the malfunction caused by false {latches+paths}.
cyclic-SAT [44–46]	cyclic locking [28, 43, 95]	As a basic assumption, in a cyclic-based SAT attack, the stateful cycles will be skipped before pre-processing the combinational cycles. However, in data flow obfuscated circuit, the keys are located in stateful cycles, and this attack cannot formulate this form of cycles.
SMT [77]	Delay Locking [31]	Using a graph theory solver, the SMT formulates setup/hold time to guarantee the timing constraint. In data flow obfuscated circuit, there is no timing violation when the key is not correct making this attack useless.
CP&SAT [96]	Cross-lock [32], Full-lock [33]	It finds the CNFs corresponded to routing modules, and simplifies them using cardinality constraints. Then It uses the SAT attack on the simplified CNF. It only works on routing-based obfuscation and has no efficiency on data flow obfuscation.
Sequential SAT [53, 54]	Sequential or Scan Lock [5, 47, 97]	It requires unrolling, and the time of unrolling in synchronous circuits was at rising/falling clock edge, and synchronous for all FFs. But, in data flow obfuscated circuit, the time that latches are updated is desynchronized and hidden using an obfuscated asynchronous controller.
Shift/Leak [48, 49]	Scan Block [5, 48]	It is very specialized for only one case that has leaking possibilities. Not Applicable to data flow obfuscated circuit.
Scan Unlock [3, 4]	Scan Lock [47, 57]	It detects the structure of LFSR used for dynamicity. No special deterministic structure is used in data flow obfuscation to be detected using the same approach.

running the SAT solver. Depending on whether the cycle is oscillating or stateful, the SAT solver will either be trapped in an infinite loop or will return UNSAT. Moreover, in attacks such as BeSAT [45] that can track and detect non-structural cycles, the very first assumption is that the circuit has no stateful combinational cycle by itself.

5.9.2 Security versus the Sequential SAT Attack

The adversary may attempt to engage either an unrolling-SAT attack or SAT integrated with BMC (SAT-BMC) by creating the unfolded combinational equivalent circuit to find

*dis*es. The length of *dis*es determines the number of unrolling required before running the SAT solver. When the circuit is synchronous, per each clock cycle, the FFs will be updated only once. So, the adversary can replicate whole CLs iteratively (per each clock cycle) to build the unrolled circuit. However, when we use asynchronicity in data flow obfuscation, the adversary needs to know the list of enabled latches continuously and cycle-accurately to unroll those parts that are triggered with new latched data. But, in the data flow obfuscation paradigm, the asynchronous controller that determines which latches must be enabled/disabled is obfuscated. So, the adversary cannot build the unrolled circuit to still get the benefit of the SAT attack, and thus, the sequential SAT attack cannot be applied to this technique.

5.9.3 Security versus the Re-Synch. + Sequential SAT Attack

Since the sequential SAT attack cannot be applied directly to data flow obfuscation, the adversary might add a pre-processing step, such as *re-synchronization*, to make this attack valid. In re-synchronization, which is the reverse of desynchronization, the asynchronous controller is removed, M and S latches are merged as FFs, and all FFs are connected to a global clock. However, for a few reasons, re-synchronization of obfuscated desynchronized netlist is not possible:

First, since each controller requires a local clock tree in asynchronous circuits, and these local trees do not have the same delay [98], the adversary needs to confirm two constraints below, which contradict each other: (1) Theoretically, the adversary must use a clock period larger than any delay element, to avoid metastability happening in the asynchronous controllers from the delay element. The delay constraints of an asynchronous circuit could be modeled using the following formulas obtained from Fig. 5.10:

$$D_{R_i} + T_{r_{i+1}} + D_{L_{i+1}} > D_{L_i} + T_{c_i} + D_{P_i} + T_{s_{i+1}} \tag{5.1}$$

$$D_{A_i} + T_{a_i} + D_{L_i} + T_{c_i} + D_{P_i} > D_{L_{i+1}} + T_{h_{i+1}} \tag{5.2}$$

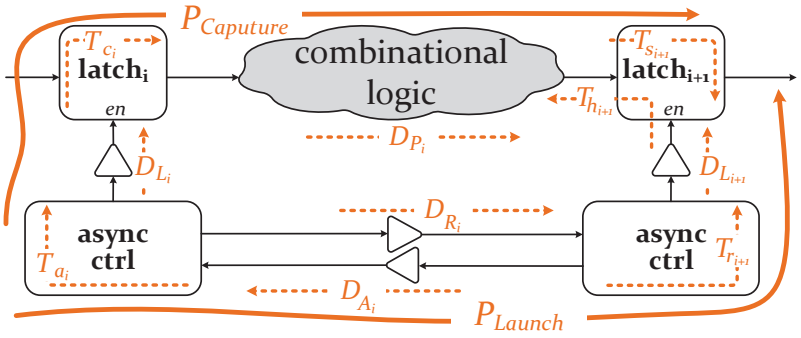


Figure 5.10: Hold and Setup Paths in an Asynchronous Circuit.

(2) But, the adversary must use a very small clock period to utilize the delay element as a time offset. Hence, based on these two constraints, the timing correctness conditions cannot be satisfied. Also, the timing constraints between the datapath and the asynchronous controller must be preserved making it more challenging [98].

Second, as a step during *re-synchronization*, the attacker must analyze every connectivity in the netlist, and effectively create a mapping problem using bipartite graphs between pairs of $\{M, S\}$ to FFs. To accomplish this, the attacker must have prior knowledge of design methodology used for creating the asynchronous circuit (2-phase latches or 3-phase latches, handshaking protocol, initial marking (tokens), etc.). Even while the adversary has access to this prior knowledge, since the connectivity is obfuscated using false paths, the false mapping will be added to this bipartite graph, leading to failure of correct matching between Ms and Ss.

Additionally, as shown in Fig. 5.2 and Fig. 5.4, during *desynchroniztion* flow, re-timing has been engaged in data flow obfuscation by moving parts of CLs before/after the latches. By doing so, re-synchronization cannot be accomplished directly. Re-timed asynchronous circuit can be converted to a 2-phase *non-overlapping* synchronous design, with two clock signals, $\phi 1$ and $\phi 2$. However, false paths with extra latches makes this 2-phase *non-overlapping* synchronous netlist malfunction. For example, latches H and I in Fig. 5.8b would be connected to clock signal $\phi 1$ and $\phi 2$ (*non-overlapping* clock signals). By connecting these two false latches to clock signals, their values would be updated which

alters the functionality of CL_1.

5.9.4 Security versus the Structural-based Attacks

The attacker might try to guess the value of the keys based on the overall structure of the locked netlist. For instance, all handshaking signals to/from latches H and I in Fig. 5.8d are controlled using C^*-element and key-gates (ANDs). Hence, the attacker might guess that this pair of latches are false latches located on a false path. So, the value of the keys can be retrieved easily. However, to avoid such circumstances, these key-gates and C^*-element will be added for a set of arbitrary (actual) latches in the netlist. For example, in Fig. 5.8d, the same key-gates are added between C and D. However, they always must be active. Also, the original C-element before C and E could be replaced with C^*-element. So, unlike *C^*-element* before A and B, in which only one of the inputs is valid, in these C^*-elements, both inputs are valid. By using this simple mechanism, the attacker cannot start guessing/detecting the false {paths + latches} based on the location/type of key gates. Fig. 5.11 shows a more complicated instance, in which all C-element are replaced with C^*-element. Also, similar to the latches H and I, C^*-element and key-gates (ANDs) are used for $\{A, B\}$ and $\{C, D\}$. So, the attacker cannot apply any form of key-guessing structure to find the false {paths + latches}.

5.9.5 Security versus Other State-of-the-Art Attacks

As was shown in Table 5.1, there exist many attacks on different logic obfuscation techniques, each is modeled to break one or more specific techniques. However, Table 5.2 explains why none of these attacks is applicable to the proposed solution. The biggest advantage of the proposed data flow obfuscation is the usage of extensive non-determinism of asynchronicity for obfuscation purposes. In data flow obfuscation, the main source of this non-determinism, which is the asynchronous controller, is the main target of obfuscation. Obfuscating an asynchronous controller makes every step of simplification dependent on the key value, and it extremely boosts up the state space in the asynchronous part. This

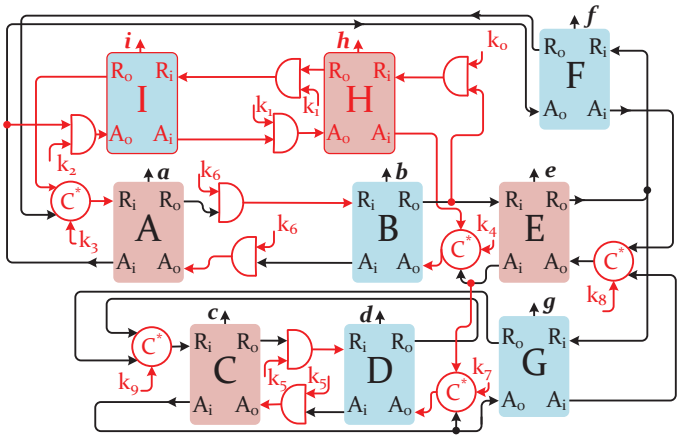


Figure 5.11: Data Flow Obfuscation Example with More Deceiving Paths.

implies the difficulty of attacking the proposed data flow obfuscation, where it requires an extensive (likely impossible) investigation on how the existing formal verification methods might be fitted and useful to be adopted in this case.

5.9.6 Testability of Data Flow Obfuscation

As discussed previously, the handshaking asynchronous controller is a self-testing circuit. However, since we use the halt in false paths for logic locking purposes, data flow obfuscation prevents the self-testing and contradict this property. To protect against an untrusted test, this contradiction enforces the designer to use an incorrect key to keep the self-testing property of these circuits. For instance, k_{0-2} in Fig. 5.8d must be 111 to avoid any halt (the correct key is 000). As another example, similarly, k_5 in Fig. 5.8d must be 1 to avoid halt on the other path (But in this case, the correct key is 1). Also, since C^*-element does not create a halt in any path, keys connected to C^*-element could be an arbitrary value to choose a path in latches controllers. It shows that there is no relation, e.g. bit flipping, between the correct key and key used for the test. Using incorrect key allows false latches located in the false paths to be updated. Hence, false paths can affect the CLs' functionality. Although the designer can generate test patterns that avoid making them driving, since the incorrect key only adds false paths to the original netlist, few more test

Table 5.3: Specifications of the Benchmark Circuits.

Small Circuit:	s298	s526	s1423	s5378	s9234	s13207	s15850	s35932	s38584
# of Inputs	3	3	17	35	36	62	77	35	38
# of Outputs	6	6	5	49	39	152	150	320	304
# of Gates	119	193	657	2779	5597	7951	9772	16165	18253
# of FFs	14	21	74	179	211	638	534	1728	1426

Large Circuit:	b17	b18	b19	MC8051	AES-GCM	SPARC
# of Inputs	37	37	24	52	116	95
# of Outputs	97	23	30	112	15	108
# of Gates	∼28K	∼95K	∼190K	∼6.6K	∼49.5K	233K
# of FFs	∼1.5K	∼3.3K	6.6K	∼1K	∼5.1K	12K

patterns are required to test these paths, which has no impact on test patterns generated for original parts of the netlist. Hence, there is no restriction for test pattern generation.

5.10 Experimental Results for *Data Flow* Obfuscation

We evaluate the data flow obfuscation over three sets of benchmark circuits, all listed in Table 5.3. The experiments are all executed on a 24-core Intel Xeon processors running at 2.4GHz with 256 GB of RAM. Area, power, and delay overhead of the data flow obfuscation are obtained using conventional Synopsys Design Compiler along with Synopsys generic 32nm library. We evaluate the security/overhead of the data flow obfuscation based on: (1) **obfuscation overhead:** Selection, desynchronization, and insertion of false {paths + latches} depend on the circuit size. (2) **key size:** Regardless of the circuit size, for a key size, a nearly fixed part of a circuit will be selected, desynchronized, and false {paths + latches} will be inserted.

5.10.1 Modeling of Attacks on Data Flow Obfuscation

Since the proposed data flow is dependent to the locked asynchronous controller, the timing of writing/capturing into latches is hidden, and the traditional SAT attack does not work even while the scan chain is available. So, to evaluate the security of the data flow obfuscation, we deploy two new versions of sequential SAT: (1) S_SAT: Resync + BMC + SAT,

Table 5.4: Runtime of Re-synch. + Sequential SAT with Different Configurations.

Circuit	Obfuscation Overhead = 5%			Obfuscation Overhead = 10%			Key Size = 100			Key Size = 200		
	S_SAT	S_BeSAT	S_icySAT	S_SAT	S_BeSAT	S_icySAT	S_SAT	S_BeSAT	S_icySAT	S_SAT	S_BeSAT	S_icySAT
s298	~~463.5~~ UNSAT	*UNSAT*	inf/l	w/k	*UNSAT*	*UNSAT*	~~207.1~~ w/k	inf/l	inf/l	*UNSAT*	inf/l	inf/l
s526	w/k	*UNSAT*	inf/l	*UNSAT*	*UNSAT*	inf/l	*UNSAT*	*UNSAT*	inf/l	w/k	*UNSAT*	inf/l
s1423	*UNSAT*	*UNSAT*	inf/l	w/k	inf/l	inf/l	~~568.7~~ UNSAT	inf/l	inf/l	w/k	*UNSAT*	inf/l
s5378	w/k	inf/l	*UNSAT*	*UNSAT*	*UNSAT*	inf/l	*UNSAT*	*UNSAT*	inf/l	*UNSAT*	inf/l	inf/l
s9234	*UNSAT*	inf/l	*UNSAT*	*UNSAT*	*UNSAT*	inf/l	w/k	inf/l	inf/l	*UNSAT*	*UNSAT*	inf/l
s13207	*UNSAT*	*UNSAT*	inf/l	w/k	*UNSAT*	inf/l	*UNSAT*	*UNSAT*	*UNSAT*	w/k	inf/l	*UNSAT*
s15850	w/k	inf/l	inf/l	w/k	inf/l	*UNSAT*	*UNSAT*	inf/l	inf/l	w/k	*UNSAT*	inf/l
s35932	w/k	inf/l	inf/l	w/k	*UNSAT*	inf/l	w/k	*UNSAT*	inf/l	w/k	inf/l	*UNSAT*
s38584	w/k	*UNSAT*	*UNSAT*	*UNSAT*	inf/l	*UNSAT*	w/k	inf/l	*UNSAT*	*UNSAT*	*UNSAT*	*UNSAT*
b17	*UNSAT*	*UNSAT*	inf/l	w/k	inf/l	*UNSAT*	w/k	*UNSAT*	inf/l	w/k	*UNSAT*	inf/l
b18	*UNSAT*	*UNSAT*	*UNSAT*	*UNSAT*	inf/l	inf/l	w/k	*UNSAT*	inf/l	w/k	*UNSAT*	inf/l
b19	*UNSAT*	inf/l	inf/l	*UNSAT*	*UNSAT*	inf/l	w/k	*UNSAT*	*UNSAT*	w/k	inf/l	*UNSAT*
MC8051	w/k	*UNSAT*	inf/l	w/k	inf/l	inf/l	w/k	inf/l	inf/l	w/k	inf/l	inf/l
AES-GCM	w/k	inf/l	inf/l	*UNSAT*	inf/l	*UNSAT*	w/k	inf/l	inf/l	w/k	*UNSAT*	inf/l
SPARC	*UNSAT*	*UNSAT*	*UNSAT*	timeout	*UNSAT*	*UNSAT*	w/k	*UNSAT*	inf/l	w/k	*UNSAT*	inf/l

- All numbers are in *Seconds*.
- timeout: 10^5 Seconds $\approx$ one day (Stop the attack process when time reaches timeout)
- {S_SAT/S_BeSAT/S_icySAT}: Re-Synch + Sequential SAT integrated with {Pure SAT/BeSAT/icySAT}.
- UNSAT: Could not find the satisfiable assignment (The attack was not able to formulate the data flow obfuscated circuit correctly).
- inf/l: Infinite Loop (for stateful cyclic, the attack (cyclic-based) cannot find a decision for a stateful cycle).
- w/k: wrong key (The attack was not able to formulate, or the cycles lead to an incorrect formulation, both leading to a wrong key.

(2) S_BeSAT/S_icySAT: Resync + BMC + BeSAT/icySAT [46]/[45].

Regarding the former version of the deployed attack, due to the failure of unrolling on desynchronized circuits, we need a pre-processing step to re-produce the re-synchronized version of the obfuscated circuit. We discussed in Section 5.9.3 that the exact re-synchronization is almost impossible. In this section, to validate our claim, regardless of the timing criteria, we developed an intuitive re-synchronization technique. The steps of the re-synchronization are as follows: (1) Removing the obfuscated asynchronous controller; (2) Merging each pair of M and S latches based on the connectivity of them (moving them across CLs). (3) Replacing each pair of M and S latches with a FF. (4) Connecting FFs to a synchronous clock signal. (5) Adding an extra key-controlled MUX for each path that comes from the output of FFs. (For each MUX, an input comes from the output of the FF, and one input is an extra key. The selector of the MUX is another key input.) Using these 5 steps, Fig. 5.12 shows the re-synchronized version of the obfuscated circuit from Fig. 5.8b. Now, this re-synchronized version could be the input of the Sequential SAT attack. By using this model, if a path is a false path, then the logic value of this path is always fixed. So, the MUX must select the extra key input with corresponded value; However, if a path is an

Table 5.5: Key Size and Overhead (OO) Relation in Different Scenarios.

Circuit	Key Size			Obfuscation (Area) Overhead	
	OO = 1%	OO = 5%	OO = 10%	Key Size = 100	Key Size = 200
s298	18	22	27	132.47%	306.71%
s526	20	24	31	68.20%	154.24%
s1423	17	20	32	27.71%	43.54%
s5378	24	34	54	12.67%	21.02%
s9234	29	37	67	6.29%	14.21%
s13207	31	95	201	5.52%	9.35%
s15850	27	88	176	4.26%	7.31%
s35932	52	221	472	1.34%	2.80%
s38584	47	237	460	1.24%	2.01%
b17	43	307	561	0.57%	0.90%
b18	137	408	835	0.12%	0.31%
b19	221	557	926	0.09%	0.17%
MC8051	34	107	271	5.51%	8.28%
AES-GCM	189	643	1017	0.21%	0.40%
SPARC	324	1004	1722	0.07%	0.14%

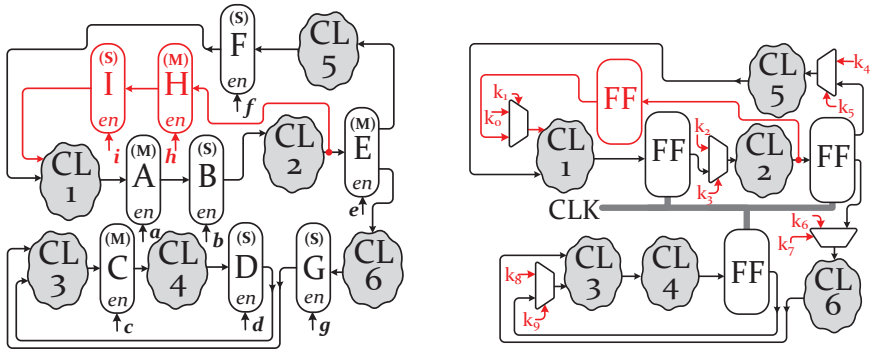
actual one, the other input of the MUX must be selected.

Regarding the latter version of the deployed attack, since the asynchronous controller is in place with lots of combinational cycles, we replaced the traditional SAT attack with existing cyclic-SAT attacks, i.e. BeSAT and icySAT [45, 46].

5.10.2 Attack Results

Table 5.4 shows the results of two attacks. For all cases, both attacks failed to break the obfuscated desynchronized circuit. The result of the attacks, in both versions, might return a wrong key, might return UNSAT, or might trap in an infinite loop. These three scenarios happen for a few main reasons: (1) regarding the wrong key (w/k) and UNSAT in S_SAT, the unrolling could not build the correct unrolled version due to asynchronicity; (2) regarding facing an infinite loop in S_BeSAT and S_icySAT, all are because of facing lots of stateful cycles in asynchronous controller; and (3) regarding the UNSAT in S_BeSAT and S_icySAT, before facing an infinite loop, the solver is trapped in a wrong decision leading to UNSAT (because of incorrect formulation).

As shown in Table 5.4, in some rare cases, we see some numbers are struck out and



(a) Obfuscated Asynchronous Circuit (b) Re-synchronized Circuit

Figure 5.12: Re-synchronization using MUX-based Path Selection.

replaced with *w/k* and *UNSAT* when we apply the S_SAT on re-synchronized circuits. In these cases, the S_SAT was able to find the correct key values. However, we found that re-timing did not relocate the latches after desynchronization for such cases. So, we force the re-timing step to do a minor relocation for set of latches to eliminate the possibility of applying any form of re-synchronization. In this case, after enforcing those minor relocation, the S_SAT fails to break them. Also, for some cases, we face time-out (10^5 seconds), implicitly showing the complexity of SAT circuit. For all other cases, since we remove all stateful combinational cycles during re-synch., we only faced with w/k or UNSAT. Table 5.4 implies that the existing attacks cannot formulate the proposed solution properly to break it, regardless of the size/portion of the circuit, and regardless of the number of false paths inserted into the circuit.

Based on the two obfuscation metrics, we evaluated these deployed attacks in 5 different scenarios, in which a specific value for one of the metrics has been fixed. Table 5.5 shows that for each scenario with a fixed metric, what the value of the other metric is, which helps us to have an estimated relationship between these two metrics.

Table 5.6: Data Flow Obfuscation Overhead (Locking Overhead = 10%).

Circuit	Area $_{um^2}$			Max Delay $_{ns}$			Power $_{uW}$		
	original	locked	↑↓%	original	locked	↑↓%	original	locked	↑↓%
s298	152.1	165.2	8.62%	0.31	0.33	6.45%	14.17	15.34	8.23%
s526	284.3	307.4	8.16%	0.26	0.25	-3.84%	13.34	14.49	8.64%
1423	1018.7	1101.8	8.16%	1.22	1.25	2.45%	44.84	48.40	7.94%
s5378	2181.5	2316.5	6.19%	0.62	0.58	-6.4%	103.2	109.45	6.1%
s9234	2895.1	3019.0	4.28%	0.38	0.37	-2.65%	131.3	136.92	4.28%
s13207	5023.5	5361.0	6.72%	1.28	1.23	-3.9%	214.9	228.13	6.1%
s15850	6077.1	6404.0	5.38%	1.25	1.18	-5.5%	272.6	286.1	4.9%
s35932	15413.4	16657.3	8.07%	1.15	1.24	7.7%	1609.4	1730.18	7.5%
s38584	23118.6	24639.5	6.58%	1.14	1.13	-0.8%	1786.9	1902.15	6.4%
b17	46872.9	49319	5.22%	1.34	1.31	-2.24%	1927.6	2025.7	5.1%
b18	134829	139737	3.64%	1.82	1.83	0.55%	2269.9	2360.4	3.9%
b19	252945	262001	3.58%	1.97	1.94	-1.52%	2982.7	3070.1	2.9%
MC8051	4982.9	5474.7	9.87%	1.29	1.31	1.55%	188.6	206.3	9.3%
AES-GCM	105319	113681	7.94%	1.76	1.77	0.57%	1876.4	2018.8	7.6%
SPARC	298231	313650	5.17%	1.38	1.41	2.17%	3380.7	3533.4	4.5%

5.10.3 Area/Power/Delay Overhead Comparison

In this section, we evaluate the post-synthesis overhead of our data flow obfuscation. Table 5.6 compares the power, performance (delay), and the area (PPA) of the original vs. obfuscated circuits while the obfuscation overhead is set to 10%. 10% obfuscation overhead means that 10% of all FFs in a circuit must be converted to latches using *desynchronization*. Also, for any obfuscation overhead percentage, the number of extra {paths + latches} is set to be less than 10% of the total latches. Further, the actual latches that are obfuscated using the same key gates (to prevent any key-guessing or structural attacks) are set to be less than 10% of the total latches. For example, for b17 with ∼1.5K FFs, for 10% obfuscation overhead, we replace 150 FFs with latches; we insert up to 15 false {paths + latches}, and up to 15 actual latches are obfuscated using the same key gates. PPA overhead in the data flow obfuscation is the consequence of two operations: (1) *desynchronization* and (2) adding false {paths + latches}. The overhead of *desynchronization* is dominant while the ratio of FFs to the total number of gates is higher in the original netlist. For instance, for s13207, whose FFs' ratio to all gates is 638/7951 = 8.03%, the area overhead is 6.72%. However, in s9234 with a ratio of 3.76%, the area overhead is only 4.28%. Table 5.7 demonstrates the

Table 5.7: Area Breakdown (Locking Overhead = 10%).

Circuit	Combinational Logic (%)	FF (%)	Latches (%)	Delay Units (%)	Asynchronous Controller (%)
s13207	52.01%	33.80%	7.97%	3.59%	2.63%
s35932	42.76%	40.31%	9.51%	4.28%	3.14%
b17	59.77%	28.33%	6.68%	3.01%	2.21%
AES-GCM	44.06%	39.39%	9.30%	4.18%	3.07%
b18	69.67%	21.36%	5.04%	2.27%	1.66%
SPARC	59.33%	28.64%	6.76%	3.04%	2.23%

Table 5.8: Data Flow Obfuscation Overhead (Key Size = 200).

Circuit	Area $_{um^2}$			Max Delay $_{ns}$			Power $_{uW}$		
	original	locked	↑↓%	original	locked	↑↓%	original	locked	↑↓%
s298	152.1	618.6	306.71%	0.31	0.32	3.7%	14.17	26.41	86.4%
s526	284.3	722.8	154.24%	0.26	0.28	8.2%	13.34	22.76	70.6%
s1423	1018.7	1462.2	43.54%	1.22	1.12	-8.3%	44.84	51.05	13.9%
s5378	2181.5	2640	21.02%	0.62	0.64	2.9%	103.2	114.48	10.9%
s9234	2895.1	3306.6	14.21%	0.38	0.36	-6.3%	131.3	141.35	7.7%
s13207	5023.5	5493	9.35%	1.28	1.25	-2.0%	214.9	227.50	5.9%
s15850	6077.1	6521.6	7.31%	1.25	1.16	-6.8%	272.6	281.37	3.2%
s35932	15413.4	15844.9	2.80%	1.15	1.25	8.6%	1609.4	1648.8	2.4%
s38584	23118.6	23583.1	2.01%	1.14	1.18	3.7%	1786.9	1814.7	1.6%
b17	46872.9	47295.4	0.90%	1.34	1.37	2.3%	1927.6	1944.7	0.89%
b18	134829	135242	0.31%	1.82	1.74	-4.1%	2269.9	2282.6	0.56%
b19	252945	253382	0.17%	1.97	1.98	0.5%	2982.7	2994.0	0.38%
MC8051	4982.9	5395.4	8.28%	1.29	1.23	-4.7%	188.6	201.4	6.83%
AES-GCM	105319	105741	0.40%	1.76	1.79	1.7%	1876.4	1894.6	0.97%
SPARC	298231	298644	0.14%	1.38	1.33	3.6%	3380.7	3390.5	0.29%

area breakdown of some of the circuits when the obfuscation overhead is 10%.

Regarding the delay overhead, since re-timing is used during *desynchronization*, in some cases we even achieved slight delay improvement. However, in some cases, it imposes only a very slight difference in cycle time by up to 8%. Regarding the power overhead, due to moving from edge-triggered design to level-triggered, the power overhead is less compared to area overhead. As seen in Table 5.6, our data flow obfuscation paradigm increased the power consumption by up to 10%.

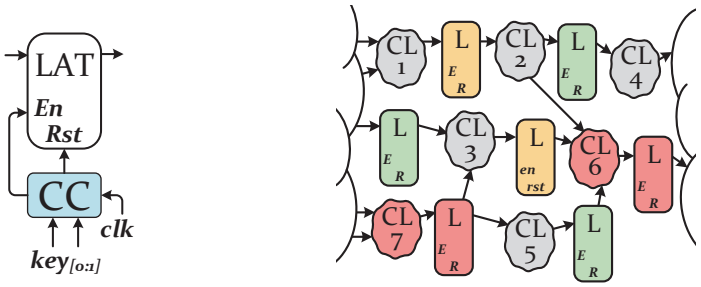
Table 5.8 compares the PPA of the original circuits vs. obfuscated circuits when the key size is 200. As shown in Fig. 5.11, and as implied in Table 5.5, in data flow obfuscation, for each extra {paths + latches}, as well as for any actual latch that are obfuscated to

Table 5.9: Data Flow Obfuscation Overhead Comparison

Circuit	Overhead	CAT 1	CAT 2		CAT 3		CAT 4		**Proposed**
		SFLL [23]	DLL [31]	Cyclic Obf. [95]	Full Lock [33]	Inter Lock [96]	R-DFS +SLL [5]	DisORC +TRLL [99]	**Data Flow Obf.**
s35932	Area ↑↓%	22.6%	5.3%	24.4%	28.1%	25.3%	9.7%	8.5%	**2.8%**
	Power ↑↓%	59.6%	3.7%	37.7%	34.9%	21.1%	14.7%	17.8%	**2.4%**
	Delay ↑↓%	0.5%	34.2%	17.5%	56.2%	5.8%	6.5%	8.7%	**8.6%**
s38584	Area ↑↓%	28.9%	4.7%	18.6%	26.9%	21.4%	10.4%	7.8%	**2.0%**
	Power ↑↓%	47.6%	2.4%	37.7%	30.1%	17.5%	14.8%	13.5%	**1.6%**
	Delay ↑↓%	0.8%	36.7%	13.7%	52.7%	14.2%	6.2%	9.5%	**3.7%**
b18	Area ↑↓%	5.6%	4.7%	13.9%	18.7%	3.4%	7.1%	2.7%	**0.3%**
	Power ↑↓%	11.6%	3.1%	28.8%	23.2%	1.8%	9.8%	6.9%	**0.6%**
	Delay ↑↓%	0.6%	39.9%	9.6%	53.9%	5.7%	5.9%	8.4%	**-4.1%**
b19	Area ↑↓%	2.7%	4.8%	9.7%	14.2%	2.8%	6.9%	1.6%	**0.2%**
	Power ↑↓%	6.2%	3.5%	22.3%	21.7%	2.2%	9.2%	1.2%	**0.4%**
	Delay ↑↓%	0.6%	44.7%	9.0%	52.6%	4.8%	5.4%	7.5%	**0.5%**
Overhead		Low	High Delay	High Power	Very High	High	Moderate	Moderate	Low
Attacked by		FALL [26]	SMT [77]	BeSAT [45]	CP&SAT [96]	–	Shift & Leak [5]	–	–

disable key-guessing, 3-5 keys could be added. So, when the key size is 200, regardless of the size of the circuit, 50-60 latches are needed. Accordingly, when the size of the key is 200, the area overhead is much higher in small circuits. However, for larger circuits, the ratio of false latches compared to the size of the circuit is significantly low, and since the real applications (ICs) are far larger than small circuits listed in Table 5.3, the area overhead is low in this approach. For instance, in AES-GCM, the area overhead is even less than 1% (0.4%). To reflect a better evaluation of overhead, in Table 5.9, we compare the overhead of data flow obfuscation when the key size is set to 200 with state-of-the-art logic obfuscation techniques[5]. As shown, on average, the overhead incurred by data flow obfuscation is much lower compared to almost all techniques. In some techniques, the overhead of one metric might be better than that of data flow obfuscation, but on average, it could be concluded that the overhead of the proposed technique is completely acceptable.

[5]Since the strength of the data flow obfuscation does not depend on the number of {paths + latches}, we could add as much as the key size (e.g. ≥ 64) to prevent breaking them using attacks like brute-force.



$key_{0:1}$	R	E
{00}	1	0
{01}	0	clk
{10}	0	$\overline{\text{clk}}$
{11}	0	1

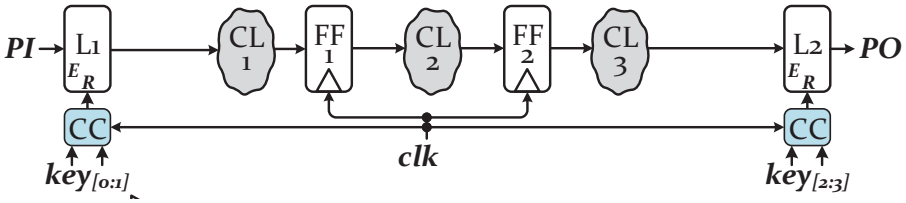
(a) Clock-Controlled Latch **(b)** Latch Insertion **(c)** Latch Functionalities

Figure 5.13: Latch-based Logic Locking Technique [6].

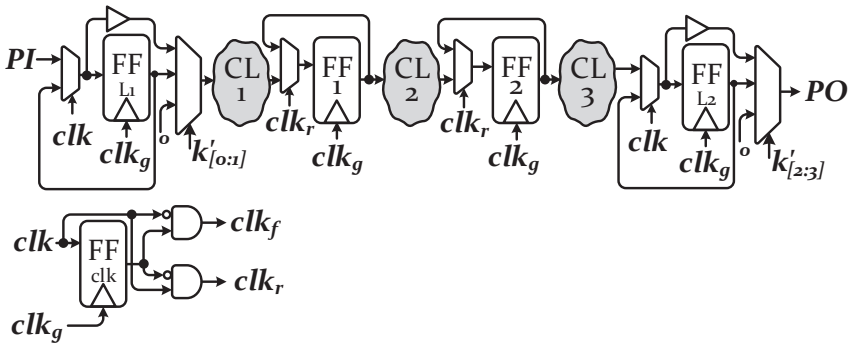
5.10.4 Comparison with State-of-the-art

Most recently, a new study has evaluated the possibility of latch-based architecture as a new means of logic obfuscation [6]. In this study, as shown in Fig. 5.13, key-programmable latches could be used as: (1) regular storage elements (FFs that are replaced with green latches subjected to re-timing), (2) programmable logic decoys (red latches/CLs) with constant output (always zero with no driving effect), and (3) programmable path delay decoy for delay manipulation (yellow latches). However, unlike our proposed data flow obfuscation, it is still fully dependent on the clock signal (reset and enable of latches are a function of the clock signal) allowing us to convert this solution to a synchronous obfuscated problem. Clock-dependent latches operate as storage elements with synchronized gated clock, and due to this synchronicity, by integration with two pre-processing steps, it still could be modeled and broken using sequential SAT integrated with cyclic model: (step 1) Generating a single-clock synchronized locked circuit using a generalized/automated model, and (step 2) Detection of (some) programmable logic decoys with constant output through (pseudo-exhaustive) test patterns on testable points of oracle.

Unlike latch-based logic locking that uses BMC with two copies of circuit per each clock cycle (due to level-triggering of latches) [6], we first generate a single-clock model to avoid this duplication per cycle. Fig. 5.14 shows we could build a single-clock fully synchronized



(a) Latch-based Logic Locked Circuit



(b) Single-Clock Synchronized Latch-based Logic Locked Circuit

Figure 5.14: Re-synchronization in Latch-based Logic Locking.

circuit from the latch-based logic locked circuit. Based on the modes of latches demonstrated in Fig.5.13c, Latches are replaced with a FF with two MUXes (one 2-to-1 preceding and one 4-to-1 following). The 4-to-1 MUX builds all three modes illustrated in Fig.5.13c, and the 2-to-1 MUX is for keeping FF values when clk is not triggering (latching). Also, a 2-to-1 MUX will be added before each neighboring FF (immediate neighbors of latches). Now, all FFs are connected to a low frequency generalized clock signal (clk_g). The selector of 2-to-1 MUXes of FFs corresponded to latches will be connected to the original clock signal, and the selector of that of neighboring FFs will be connected to a gated clock signal for falling (clk_f) or rising (clk_r) levels. This generalized model could be used for different scenarios with more complexities, such as multi-clock systems, and circuits with gated clock, all could be synchronized using generalized clock signal clk_g [100]. It allows us to use BMC with *ONLY* one copy of the circuit per each clock cycle, which improves the scalability significantly.

Table 5.10: Attack Time on the existing latch-based logic locking.

Circuit	key size = 20	key size = 50	key size = 100	key size = 200
	exe time	exe time	exe time	exe time
s1423	468.5	521.6	321.1	1450.8
s5378	1422.1	1475.9	1855.6	2991.3
s15850	225.8	3006.5	2284.5	4712.6
s35932	3824.5	2071.9	10225.4	11452.8

timeout: 10^5 Seconds $\approx$ one day (Stop the attack when time reaches timeout)
Attack Time: Resync. + Decoy COI Reduction + Sequential SAT integrated with BeSAT

Also, since programmable logic decoys (red latches and red CLs) always generate constant output (zero output), and since testable pin are dedicated for latches (using extra MUXes and duplicate FFs) in the existing latch-based approach [6], the adversary would be able to apply test patterns (stuck-at-fault or pseudo-exhaustive) on testable points at Cone-Of-Influence (COI) of oracle to detect latches with constant (zero) output. For some fundamental reasons, the programmable logic decoys could not be large enough to make this test infeasible: (1) programmable logic decoys are hardware overhead, which must be limited, (2) these logic decoys add difficulties to P&R which compromises the performance, and (3) it should not have an impact on maximum frequency of the circuit before adding obfuscation. Detecting these latches helps reducing the cone-of-influence, and consequently the SAT circuit before running the sequential SAT attack. Table 5.10 shows the execution time of the sequential SAT attack when it is integrated with these two pre-processing steps. However, since our proposed data flow obfuscation is truly desynchronized (token-based with no dependency on a clock signal), this form of re-synchronization is not applicable to it.

Table 5.11 also shows some major advantages of the proposed solution against existing latch-based logic locking [6]. For example, we use a self-testable asynchronous controller; however, latch-based adds extra MUXes/FFs to make the latches testable, resulting in extra overhead. We add false latches *pair by pair*, which makes any structural/functional analysis exponentially harder (any pair of neighboring latches is a point of analysis); However,

Table 5.11: Proposed Data Flow Obfuscation vs. Latch-based Logic Locking.

Scheme	**Test Unit**	**False Latches**	**Clock Dependency**	**Synchronizable**
Proposed	Self Testable	Paired	Truly Asynchronous	✗
Latch-based [6]	Duplicate FFs	Single	Fully Clock-dependent	✓

Scheme	**BMC Model ?**	**Cyclic ?**	**Cyclic Model ?**
Proposed	✗	Stateful/Oscillating in Ctrls	hard
Latch-based [6]	✓ (with resynchronizing)	structural in datapath	easy

Scheme	**Overhead**	**Area*** ↑↓%	**Power*** ↑↓%	**Delay*** ↑↓%
Proposed	Latches + Ctrls	1.32%	1.25%	2.17%
Latch-based [6]	Latches + Ctrls + Decoy Logic	13.7%	5.4%	2.11%

∗: Average on s35932, s38584, b18, and b19 when key size = 200.

latch-based adds decoy latches *one by one*, which allows analyzing them linearly (each latch is a point of analysis). In the proposed scheme, the latch enables controller consists of many stateful cycles that boost the difficulties for the adversary (no cyclic modeling). However, only easy-to-track structural cycles might be added into the existing latch-based logic locking when the key is not correct. Also, the overhead is much higher in the existing approach due to adding programmable logic decoys.

5.11 What we Learnt in this Chapter

In this Chapter, to combat state-of-the-art attacks on logic obfuscation, we introduced a new obfuscation paradigm called data flow obfuscation. By exploiting the concept of asynchronicity, in data flow obfuscation, we showed how the flow of the data could be obfuscated in any arbitrary circuit. In data flow obfuscation, we engage false {paths + latches} using the asynchronous structure to control the flow of data in specific timing paths. Using this mechanism, we showed that the SAT attack has no longer an advantage for the adversary even while the scan access is not restricted. Also, we showed that how asynchronicity combat the sequential SAT attack by invalidating the unrolling step in these attacks. We comprehensively investigated the effectiveness of data flow obfuscation over wide-range benchmark

families. Our experiments showed the resiliency of this new paradigm against all existing attacks at significantly low overhead.

Chapter 6: COMA: Communication and Obfuscation Management Architecture

As discussed in Chapter 1, to remain hidden, in addition to resisting the attacks against its obfuscated circuit(s), the IC should also resist passive, active, or destructive attacks that could be deployed to read the key values. Hence, neither the activation of such devices nor the storage of key values in them should expose or leak the key information. Activation of an obfuscated IC requires storing the activation key in a secure and tamper-proof memory [34, 101]. However, there exist a group of applications that could use an alternative key storage. This alternative solution is to store the key outside the IC, where the IC is activated every time it is needed. This option requires constant connectivity to the key management source and a secure communication for key exchange to prevent any leakage of the key. This solution allows an IC designer to store the chip unlock key outside of an untrusted chip. So, no secure and tamper-proof memory is needed. Since the key is stored outside the untrusted chip, a constant connectivity to an obfuscation key-management solution is an indispensable requirement for this category of devices. This requirement could be easily met for two prevalent groups of architectures: (1) 2.5D package-stack devices where a single trusted chip is used for key management and activation of multiple obfuscated ICs manufactured in untrusted foundries, and (2) IoT devices with constant connectivity to the cloud/internet. In this Chapter, we propose the COMA [102] as a key-management and communication architecture for secure activation of obfuscated circuits that are manufactured in untrusted foundries and meet the constant connectivity requirement, namely ICs that belong to a) 2.5 package-integrated and b) IoT solutions. We describe two variants of our proposed solutions: The first variant of COMA is used for secure activation of IPs within 2.5D package-integrated devices (similar to DARPA SPADE). The second variant of COMA is

used for secure activation of connected IoT devices. The proposed COMA allows us to (1) push the obfuscation key and obfuscation unlock mechanism off of an untrusted chip, (2) make the key a moving target by changing it for each unlock attempt, (3) uniquely identify each IC, (4) remove the need for implementing a secure memory in an untrusted foundry, and (5) utilize two novel mechanisms for ultra-secure or ultra-fast encrypted communication.

6.1 COMA Advantage in 2.5D and IoT Devices

In 2.5D package-integrated ICs, similar to DARPA SPADE architecture [103], a chip which is fabricated in a trusted foundry, but in a larger technology node, is packaged with an untrusted chip fabricated in an untrusted foundry in a smaller technology node. The resulting solution benefits from the best features of both technologies: The untrusted chip may be used as an accelerator, providing the resulting hybrid solution with the much-needed scalability (higher speed and lower power), while the trusted chip provides the means of trust and security. The untrusted chip is isolated from the outside world and any exchange of information to/from the untrusted chip passes through the trusted chip.

In the IoT devices on the other hand, the constant connectivity is a characterizing feature in a wide range of IoT devices. In these solutions, the obfuscation key could be stored in the cloud, and activation of an IoT device could be done remotely. This model allows custom, monitored, and service-oriented activation (Activation As A Service). An additional advantage is removing the possibility of extracting an unlock key from a non-volatile memory that otherwise would have to be used for storing the obfuscation unlock key. Examples of which are IoT devices used for providing various services, military drones activated for a specific mission, video decryption services for paid pay-per-view transactions, etc., where a device has to operate in an unsafe environment and is at risk of capture and reverse engineering. In these applications, the IC fabricated in an untrusted foundry is activated either every time it is powered up, or for certain time intervals. The key vanishes after the service is performed, or when the device is powered down. The activation of such

devices is performed using a remote key management service (in the cloud or at a trusted base-station), and the key exchange to/from these devices should be secured.

In both 2.5D system solutions and IoT devices, the need for implementation of a tamper-proof memory, for storage of IC activation key, in an untrusted process is removed. Some reasons why implementing a secure memory in an untrusted foundry may be undesired, or practically unfeasible include:

Availability: The targeted foundry may not offer the required process for implementing a secure memory with the desired features. An example could be the requirement for storing sensitive information in magnetic tunnel junction (MTJ) memories to prevent probing and attacks that could be deployed against flash-based NVMs. Fabricating such ICs requires a hybrid process that supports both CMOS and MTJ devices, which may be unsupported by the targeted foundry.

Verified Security: The secure memory may be available in the targeted technology, however not be fully tested and verified in terms of its resistance against different attacks.

Cost: Implementing secure memory requires additional fabrication layers and processing steps, increasing the cost of manufacturing. Increasing the silicon area is a far cheaper solution than increasing the number of fabrication layers.

Reusability: In 2.5D system solutions, a trusted chip could be shared by multiple untrusted chips, manufactured in different foundries. Moving the secure memory to the trusted chip removes the need for implementing the secure memory in all utilized processes. The trusted chip could be designed once with utmost security for the protection and integrity of data. This also reduces the cost of manufacturing untrusted chips by removing the need for additional processing steps for implementing secure memory.

Ease of Design: Implementing secure memory requires pushing the design through non-standard physical design flow to implement the tamper-proof layers in silicon and package. In addition, the non-volatile nature of tamper-proof memory requires read and write at elevated voltages, increasing the burden on the power-delivery network design. Reuse of a trusted chip with a tamper proof memory that could manage activation of other obfuscated

ICs, relaxes the design requirement of ICs to standard physical design and fabrication process.

6.2 Prior Art Key-Oriented Architectures

Active metering, Secure Split-Test, and logic obfuscation have been proposed to protect ICs from supply chain-related security threats by initializing the HW control to a locked state at power-up and hiding the design intent [12,16–18,104–107]. Some of these techniques support single activation, while others support active metering mechanisms. Active metering techniques [12, 18, 104, 106] provide a mechanism for the IP owner to lock or unlock the IC remotely. In these solutions, the locking mechanism is a function of a unique ID generated for each IC, possibly and preferably by a *Physical Unclonable Function* (PUF) [34]. Only the IP owner knows the transition table and can unlock the IC. Active metering, combined with a PUF, makes the key a moving target from chip to chip. However, there exist a few issues with previous metering techniques: first, the key(s) to unlock each IC remains static. Second, these techniques unlock the chips before they are tested by the foundry. Hence, the IP owner can control how many ICs enter the supply chain, but not how many properly tested ICs exit the supply chain. Finally, these techniques do not respond well to the threat of the foundry requesting more IDs by falsifying the yield to be lower during the test process. Such shortcomings can potentially allow the foundry to ship more out-of-spec or defective ICs to the supply chain.

Many of these shortcoming were addressed in FORTIS [97] shown in Fig. 6.1. In FORTIS the registers that hold the obfuscation key are made a part of the scan chain, allowing the foundry to carry structural test by assigning test values to these registers prior to the activation of the IC. Authors of [97] argue that placing a DFT compression logic, not only reduces the test size, but also prevents the readout of the individual register values. After testing the IC, the obfuscation key is transferred and applied to unlock the circuit using two types of cryptographic modules: a public-key crypto engine, and a One Time Pad (OTP) crypto engine.

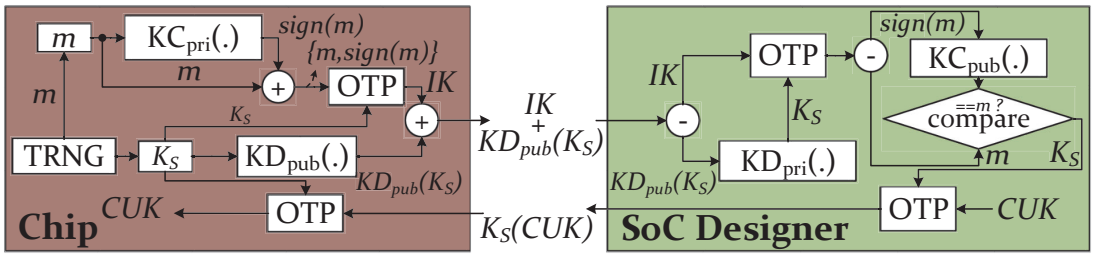


Figure 6.1: FORTIS: Overall Architecture.

In FORTIS, the public and private keys are hardwired in the design. A TRNG is used to generate a random number (m) that is treated as a message. This message is encrypted using the private key of the chip to generate a signature sig(m). The actual message and its signature are concatenated and later used as a mean for the authentication of the chip. At the same time, the TRNG generates another random number K_S. This random number is used as the key for OTP, and at the same time is encrypted using the public key of the designer to generate $KD_{pub}(K_S)$. OTP uses K_S for encrypting the $(m, sig(m))$, and the output of OTP is concatenated with the $KD_{pub}(K_S)$. The resulting string of bits is transmitted to the SoC designer. The SoC designer uses an OTP to obtain m and $sig(m)$ for the purpose of authentication. She then uses the private key of the designer to recover K_S. Finally, K_S is used by OTP to encrypt the chip unlock key (CUK). The encrypted CUK is transmitted to the chip, decrypted using OTP, and applied to the obfuscation unlock key registers to unlock the circuit.

FORTIS, however, suffers from several security issues including 1) using identical public and private keys in all manufactured chips, and thus its inability for unique device authentication, 2) being vulnerable to modeling attack in which the FORTIS structure is modeled in software for requesting the CUK from SoC designer 3) being vulnerable to side channel attacks on public-key encryption engine aimed at recovering the private key of the chip, 4) being vulnerable to fault attacks in which the value of K_S is fixated, 5) requiring a secure

memory for storage of the obfuscation unlock key, and 6) not addressing the mechanism for generating a unique and truly random seed to initialize PRNG. After describing our proposed solution, in section 6.7, we explain how these vulnerabilities are addressed in our proposed solution.

Our proposed solution fits the category of active metering techniques. The key is neither static nor stored in the untrusted chip. A key that is used to activate the IC at the test time cannot be reused to activate the same or a different IC in the future. Hence, the test facility is able to accomplish the test process using ATPG tools with a key which is valid for structural/functional test and it is not valid for any subsequent activation. Additionally, the communication to/from IC is secured using a side-channel protected cryptographic engine, combined with a dynamic switching and inversion structure that enhances the security of the chip against invasive and side-channel attacks. We demonstrate that COMA provides two useful means of secure communication to/from the untrusted chip, one for added security, and one for supporting a higher throughput. The proposed architecture is a comprehensive solution for the key management of the obfuscated IPs, where the challenges related to the activation of the IC and secure communication to/from the IC are addressed at the same time. However, as discussed earlier, it is not a universal solution and would fit within the context of IoT-based solutions or within 2.5D package-integrated solutions, as this solution requires constant connectivity.

6.3 Proposed COMA Architecture

The primary goal of the COMA is to remove the need for storing the *obfuscation key* (OK) on an untrusted chip while securing the communication flow used for activation of the obfuscated circuit in the untrusted chip. The additional benefits of the proposed architecture are the implementation of two new modes of 1) highly secure and 2) very high-speed encrypted communication.

We propose two variants of the COMA architecture: The first variant is designed for

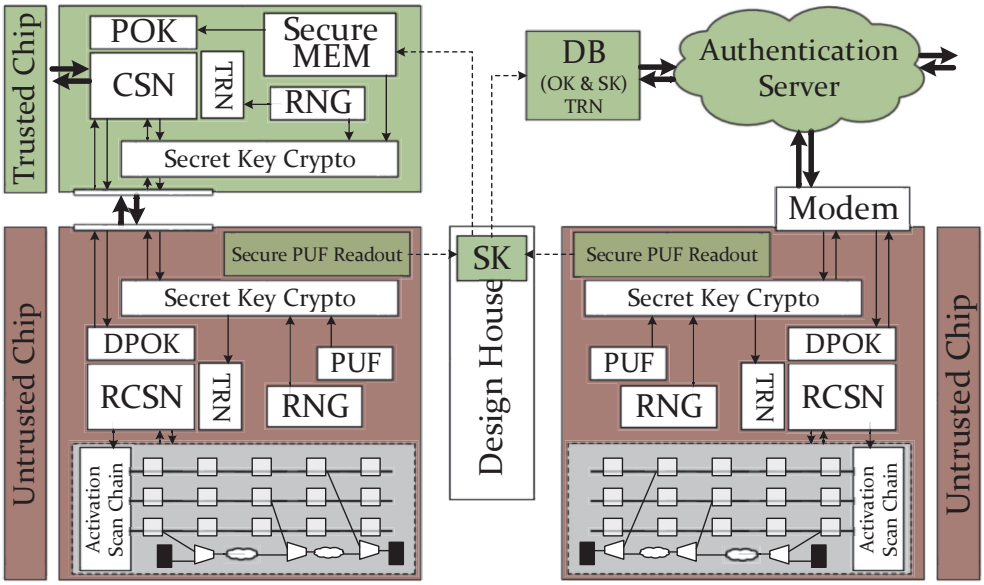


Figure 6.2: Proposed COMAs for (left) 2.5D and (right) IoT-based/remote devices.

securing the activation of the obfuscated IP and communication to/from an untrusted IC in 2.5D package-integrated architectures similar to the DARPA SPADE architecture [103] (denoted by 2.5D-COMA). The second proposed architecture is designed for protecting IoT-based or remotely activated/metered devices (denoted by R-COMA). Fig. 6.2 captures the overall architecture of two variants of the proposed COMAs.

6.3.1 2.5D-COMA: Protecting 2.5D package integrated system solutions

The DARPA SPADE project [103] explores solutions in which an overall system is split manufactured between two different technologies, In this solution, a trusted IC which is constructed in an older yet secure technology is packaged with an IC fabricated in an untrusted foundry in an advanced geometry. The purpose of this solution is to provide the best of two worlds: the security of older yet trusted technology and the scalability, power, and speed of the newer yet untrusted technology. The 2.5D-COMA is designed to work with an architecture similar to the DARPA SPADE architecture. The proposed solution allows an entire or partial IP in an untrusted chip to be obfuscated, while pushing the mechanism

for unlocking and secure activation of the untrusted chip out to a trusted chip. In this solution, the trusted chip encapsulates the sensitive information, verifies the integrity of the untrusted chip, performs sensitive logic monitoring, and controls the activation of the untrusted chip. Also, the key to unlock the obfuscated circuit changes per activation, details of which will be explained shortly.

As shown in Fig. 6.2, the two variants of COMA contain two main parts, the trusted side (*green*) and the untrusted side (*red*). In both variants, the architectures of untrusted chips are identical, and only the architectures of trusted sides are different. In 2.5D-COMA, only the trusted chip is equipped with a secure memory. The secure memory stores the *Obfuscation Key* (OK) and the Secret Key (SK) used for encrypted communication between the trusted and untrusted chips. The SK is generated using a PUF in the untrusted chip, thus it is unique for each untrusted chip, and the untrusted chip does not need a secure memory to store the SK. The *Configurable Switching Network* (CSN) and *Reverse* CSN (RCSN) are logarithmic routing and switching networks. They are capable of permuting the order and possibly inverting the logic levels of their primary inputs while these signals are being routed to different primary outputs. The RCSN is the exact inverse of the CSN. Hence, passing a signal through CSN-RCSN (or RCSN-CSN) will recover the original input. The switching and inversion behavior of CSN-RCSN is configured using a *True Random Number* (TRN). This TRN is generated in the trusted chip to avoid any potential weakening/manipulating of the TRNG. In addition, since the TRNG in COMA is equipped with standard-statistical-tests applied post-fabrication, such as Repetition-Count test and the Adaptive-Proportion test, as described in NIST SP 800-90B [108], any attempt at weakening the TRNG during regular operation (i.e. fault attack) can be detected by continuously checking the output of a source of entropy for any signs of a significant decrease in entropy, noise source failure, and hardware failure. By using TRN for the CSN-RCSN configuration, any signal passing through the CSN is randomized, and then by passing through the RCSN is recovered. Additional details are provided in section 6.4.1.

The untrusted chip unlock process in COMA is as follows: Prior to each activation,

the CSN and RCSN are configured with the same TRN. Since the SK is a PUF-based key generated at the untrusted side, first the SK must be securely readout from untrusted chip. This is done by deploying public key cryptography, the details of which are described in section 6.4.4. Then, the trusted chip encrypts the TRN using the SK and sends it to the untrusted chip. To perform an activation, as shown in Fig. 6.2, the OK is read in segments, denoted as *Partial Obfuscation Key* (POK), and is passed through the CSN and encryption on the trusted side and the decryption and RCSN on the untrusted side. This process is repeated every time the obfuscated circuit in the untrusted chip is to be activated, each time using a different TRN for configuring the CSN-RCSN. Usage of a different TRN as the configuration input for the CSN-RCSN for each activation randomizes the input data to the Secret key crypto engine. Hence, by using a different TRN for each activation, the obfuscation key (after passing through CSN) is transformed into a one-time license, denoted as *Dynamic Activation License* (DAL). Since the OK is read and sent in segments (from the trusted chip), the DAL will be received (at untrusted chip) in segments, denoted as *Dynamic Partial Obfuscation Key (DPOK)*, shown in Fig. 6.2, and is used as an input to RCSN. Passing DPOKs through RCSN recovers the POKs, and concatenating the POKs will generate the OK. Note that the DAL is only valid until the TRN is changed. So, the DAL cannot be used to activate other chips or the same chip at a later time.

In 2.5D-COMA, the untrusted chip(s) is used as an accelerator, and for safety reasons should not be able to directly communicate to the outside world. Hence, all communication to/from the untrusted chip must go through the trusted chip. In addition, it is possible that the computation, depending on the sensitivity of processed data, is divided between the trusted and untrusted chips. Hence, there is a need for constant communication between the trusted and untrusted chips. The communication needed is sometimes for limited but highly sensitive data, and sometimes for vast amounts of less sensitive data. As illustrated in Fig. 6.3, the proposed architecture is designed to provide two hybrid means of encrypted communication : (1) Double-Cipher Communication (DCC) as ultra-secure communication, and (2) Leaky-Cipher Communication (LCC) as ultra-fast communication mechanism [109].

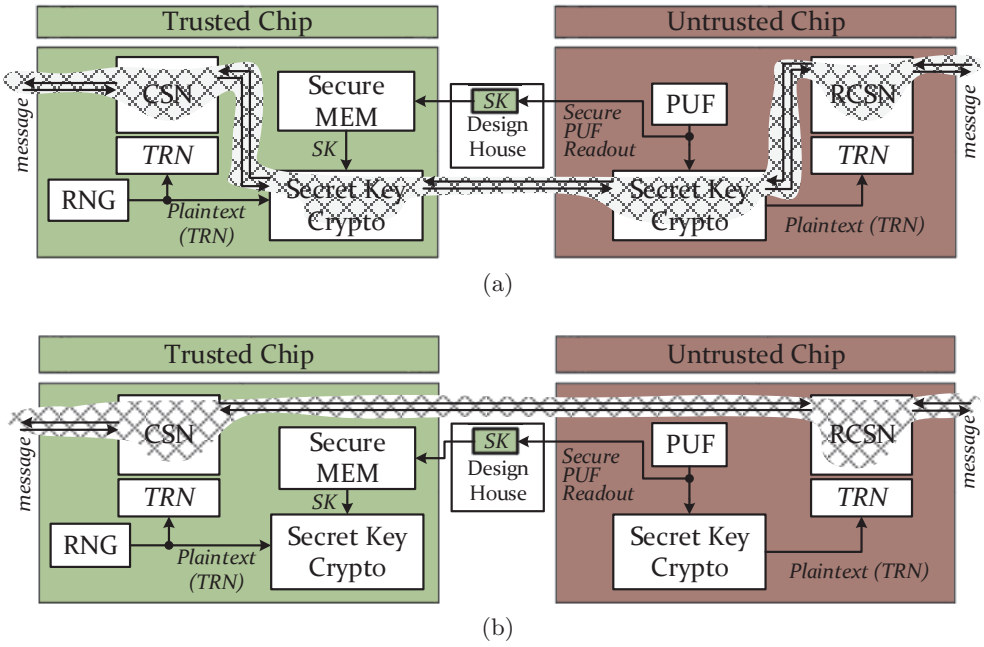


Figure 6.3: Modes of Encrypted Communication in COMA: (a) DCC, (b) LCC.

Double-Cipher Communication (DCC)

As shown in Fig. 6.3(a), in **DCC** each message passes through both CSN-RCSN and the secret key cryptography engine, where the TRN used in CSN-RCSN is renewed every U cycles. DCC provides the ultimate protection against side-channel attacks. In DCC mode, two necessary requirements for mounting a side channel attack are eliminated. The side channel attack aims to break the cryptography system by analyzing the leaked side channel information for different *input patterns*. Hence, (1) the degree of correlation between the input and the leaked side-channel information, and (2) the intensity of side-channel variation, are important. In COMA, the attacker cannot control the input to the secret-key cryptography. In addition, the input to the CSN is randomized using a TRN and then passed to the secret-key cryptography, removing the correlation between leaked side channel info (from secret-key cryptography) and the original input to the CSN. Additionally, the secret-key cryptography engine is side-channel protected to pass a t-test [110]. So, the intensity and variation in side-channel information is significantly reduced, making the DCC

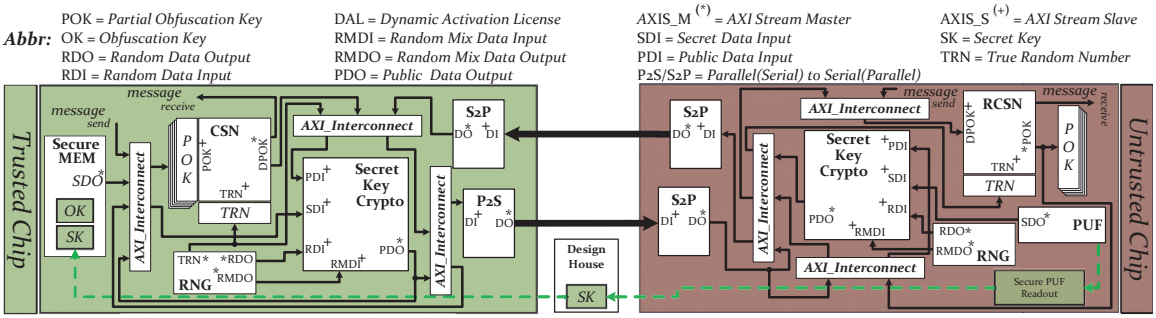


Figure 6.4: 2.5D-COMA Architecture.

an extremely difficult attack target.

Leaky-Cipher Communication (LCC)

LCC is a fast and energy-efficient mode of communication between the trusted (or remote device) and the untrusted chip. As illustrated in Fig. 6.3(b), in this protocol, the CSN-RCSN pair is used for exchanging data. The secret key cryptography engine is used to transmit a TRN from one chip to the other. Since the throughput of TRNG is the bottleneck point compared to the performance of CSN-RCSN, the TRNG is used as a seed generator to the PRNG (which offers higher performance) on both sides, Hence, in LCC mode, PRNG is used to configure the CSN-RCSN to avoid any performance degradation on transmitting data. For U consecutive cycles, the PRNG is kept idle allowing the CSN to use the same PRNG output for U cycles. It not only reduces the power consumption of PRNG and TRNG, it also provides faster communication in LCC mode. However, using this model of communication is prone to algebraic and SAT attacks as each communicated message leaks some information about the TRN used to configure the CSN-RCSN pair. If an attacker can control the message and observe the output of the CSN, each communicated message leaks some information about the key, reducing its security. Extracting the key from such observations is possible by various attack models, including Satisfiability attacks. Hence, an attacker with enough time and enough traces could extract the TRN and retrieve the communicated messages. Preventing such attacks poses a minimum limit to U (the update

frequency of the PRNG). U should be small to prevent SAT and other trace-based learning or analysis attacks, but large enough to be energy efficient. In Section 6.6, we deploy a SAT attack against LCC and will further elaborate on the required TRN update frequency.

6.3.2 R-COMA: Protecting IoT devices

The R-COMA architecture in the untrusted chip is identical to that of 2.5D-COMA. However, the trusted chip is replaced with a remote key management service. The R-COMA provides a mechanism for an IP owner to remotely activate parts or entire functionality of the hardware. Similar to 2.5D-COMA, the DAL is different from chip to chip and from activation to activation. In R-COMA, the obfuscation unlock key is stored in a central database, while the CSN, the TRNG for configuring CSN-RCSN, and the secret key cryptography engine are implemented in software.

In *R-COMA*, an authentication server (AS) first securely receives the PUF-based SK from the untrusted chip. Then, it generates a TRN and sends it to the untrusted chip for RCSN configuration. Then, the AS starts sending the obfuscation key (OK). For the activation phase, the communication is double encrypted and authenticated using the CSN-RCSN and side-channel protected cryptography engine. Each COMA-protected device needs to be registered with the AS to receive the obfuscation key. The registration is done using the PUF-ID of the untrusted chip. Hence, the PUF is used for both authentication and generation of the secret key for communication. In R-COMA, the generation of *DAL* is granted after PUF authentication, and is based on the generated TRN, and the stored OK, which is generated at design time. The generation of DAL is algorithmic and takes linear time.

6.4 Implementation Detail of COMA

Fig. 6.4 captures the overall architecture of COMA and the relation and connectivity of its macros. As discussed, COMA supports both key-management and secure data communication. Based on the selected mode of communication (LCC/DCC), the message passes

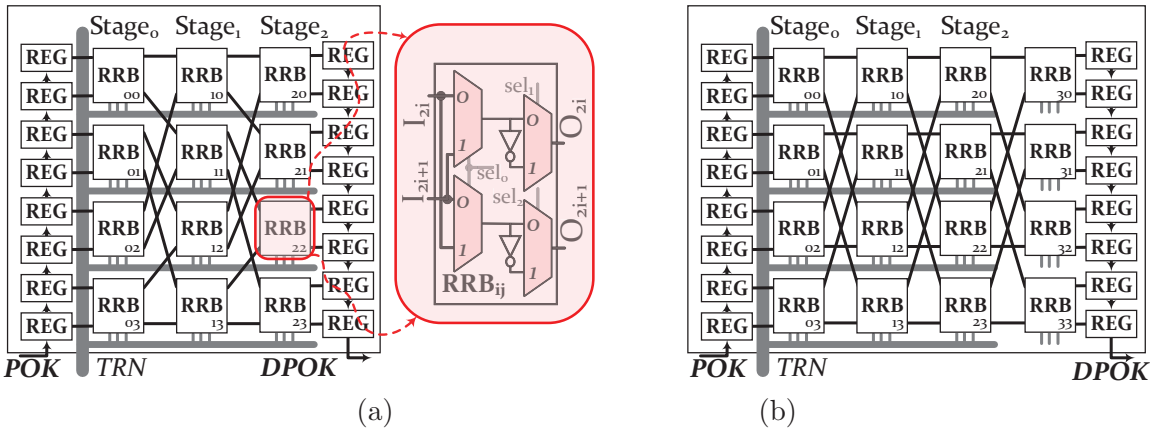


Figure 6.5: Logarithmic Network (a) Omega-based Blocking, (b) near Non-blocking.

through {CSN → RCSN} *or* {CSN → encryption → decryption → RCSN}. RNG, which contains both TRNG and PRNG, is used on both sides. In the trusted chip, RNG is used for implementing a side-channel protected cryptography engine, as well as generating the configuration of the CSN-RCSN (TRN). On the untrusted side, it is used only for implementing the side-channel protected cryptography engine. Finally, PUF is engaged in the untrusted chip for both unique IC authentication and for the generation of the secret key for encryption. As shown in Fig. 6.4, all modules employ an AXI-stream interface to maximize the simplicity of the overall design and minimize the overhead incurred by the controller of the top module on each side. The description of the behavior of each macro in COMA is provided next.

6.4.1 Configurable Switching Network (CSN)

The CSN is a logarithmic routing network that could route the signals at its input pins to its output pins while permuting their order and possibly inverting their logic levels based on its configuration. Fig. 6.5(a) captures a simple implementation of an 8-by-8 CSN using *OMEGA* [111] network. The network is constructed using routing elements, denoted as Re-Routing Blocks (RRB). Each RRB is able to possibly invert and route each of the input signals to each of its outputs. The number of RRBs needed to implement this simple CSN

for N inputs (N is a power of 2) is simply $N/2 * logN$. Each CSN should be paired with an RCSN. The RCSN, is simply constructed by flipping the input/output pins of RRB, and treating the CSN input pins as its output pins and vice versa.

The *OMEGA* network along with many other networks of such nature (Butterfly, etc.) are blocking networks [111], in which we cannot produce all permutations of input at the network's output pins. This limitation significantly reduces the ability of a CSN to randomize its input. Also, we will show that a blocking CSN can be easily broken by a SAT attack within few iterations.

Being a blocking or a non-blocking CSN depends on the number of stages in CSN. Since no two paths in an RRB are allowed to use the same link to form a connection, for a specific number of RRB columns, only a limited number of permutations is feasible. However, adding extra stages could transform a blocking CSN into a strictly non-blocking CSN. Using a strictly non-blocking CSN not only improves the randomization of propagated messages through the CSN, but also improves the resiliency of these networks against possible SAT attacks for extraction of a TRN used as the key for a CSN-RCSN cipher. A non-blocking logarithmic network could be represented using $LOG_{n,m,p}$, where n is the number of inlets/outlets, m is the number of extra stages, and p indicates the number of copies *vertically cascaded* [112].

According to [112], to have a strictly non-blocking CSN for an arbitrary n, the smallest feasible values of p and m impose very large area/power overhead. For instance, for $n = 64$, the smallest feasible values, which make it strictly non-blocking, are $m = 3$ and $p = 6$, which means there exists more than $5\times$ as much overhead compared to a blocking CSN with the same n, resulting in a significant increase in the area and delay overhead. To avoid such large overhead, we employ a *close to non-blocking CSN* described in [112] to implement the CSN-RCSN pair. 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 needs $log_2(n) - 2$ extra stages and no additional copy. Fig. 6.5(b), demonstrates an example of such a close-to-non-blocking CSN with $n = 8$. In the results section, we demonstrate

that using these close-to-non-blocking CSNs enhances the resiliency of a CSN against SAT attack, even in small sizes of CSNs with significantly lower power, performance and, area (PPA) overhead.

6.4.2 Authenticated Encryption with Associated Data

The Authenticated Encryption with Associated Data (AEAD) is used in the DCC mode for communicating messages, and in the LCC mode for the initial transmission of the CSN-RCSN key (TRN). Authenticated ciphers incorporate the functionality of confidentiality, integrity, and authentication. The input of an authenticated cipher includes Message, Associated Data (AD), Public Message Number (NPUB), and a secret key. The ciphertext is generated as a function of these inputs. A Tag, which depends on all inputs, is generated after message encryption to assure the integrity and authenticity of the transaction. This tag is then verified after the decryption process. The choice of AEAD could significantly affect the area overhead of the solution, the speed of encrypted communication, and the extra power consumption. To show the performance, power, and area trade-offs, we employ two AEAD solutions: a NIST compliant solution (AES-GCM), and a promising lightweight solution (ACORN).

AES-GCM is the current National Institute of Standards and Technology (NIST) standard for authenticated encryption and decryption as defined in [113]. ACORN is one of two finalists of the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR), in the category of lightweight authenticated ciphers, as defined in [114]. An 8-bit side-channel protected version of AES-GCM and a 1-bit side-channel protected version of ACORN are implemented as described in [115]. Both implementations comply with the lightweight version of the CAESAR HW API [116].

Our methodology for side channel resistance is threshold implementation (TI), which has wide acceptance as a provably secure Differential Power Analysis (DPA) countermeasure [117]. In TI, sensitive data is separated into shares and the computations are performed on these shares independently. TI must satisfy three properties: 1) Non-completeness: Each

share must lack at least one piece of sensitive data, 2) Correctness: The final recombination of the result must be correct, and 3) Uniformity: An output distribution should match the input distribution. To ensure uniformity, we refresh TI shares after non-linear transformations using randomness. We use a hybrid 2-share/3-share approach, where all linear transformations in each cipher are protected using two shares, which are expanded to three shares only for non-linear transformations.

To verify the resistance against DPA, we employ the Test Vector Leakage Assessment methodology in [110]. We leverage a "fixed versus random" non-specific t-test, in which we randomly interleave first fixed test vectors and then randomly-generated test vectors, leading to two sequences with the same length but different values. Using means and variances of power consumption for our fixed and random sequences, we compute a figure of merit t. If $|t| > 4.5$, we reason that we can distinguish between the two populations and that our design is leaking information. The protected AES-GCM design has a 5-stage pipeline and encrypts one 128-bit input block in 205 cycles. This requires 40 bits of randomness per cycle. In ACORN-1, there are ten 1-bit TI-protected AND-gate modules, which consume a total of 20 random reshare, and 10 random refresh bits per state update. In a two-cycle architecture, 15 random bits are required per clock cycle.

6.4.3 Random Number Generator (RNG)

An RNG unit is required on both sides to generate random bits for side channel protection of AEAD units, a random public message number (NPUB) for AEAD, and TRNs for CSN-RCSN. We adopted the ERO TRNG core described in [118], which is capable of generating only 1-bit of random data per over 20,000 clock cycles. In our TI implementations, AES-GCM needs 40 and ACORN 15 bits of random data per cycle. So, we employed a hybrid RNG unit combining the ERO TRNG with a Pseudo Random Number Generator (PRNG). TRNG output is used as a 128-bit seed to PRNG. The PRNG generates random numbers needed by other components. The reseeding is performed only once per activation.

The choice of PRNG depends on the expected performance and overhead. To support

COMA, we adopted two different implementations of PRNG: (1) AES-CTR PRNG, which is based on AES, is compliant with the NIST standard SP 800-90A, and generates 12.8 bits per cycle. (2) Trivium based PRNG, which is based on the Trivium stream cipher described in [119]. The Trivium-based PRNG is significantly smaller in terms of area and much faster than AES-CTR PRNG. It can generate 64 bits of random data per cycle, however, it is not compliant with the NIST standard.

6.4.4 PUF and Secure PUF Readout

The response of the PUF to a challenge selected randomly by Enrollment Authority (SoC designer) is used as the secret key in AEAD. Hence, the readout of the PUF-response should be protected. The simplest solution for the safe readout of a PUF-generated key is to enable the readout by burning one set of fuses, and disabling it by burning a second set of fuses. However, this solution, especially when combined with a weak PUF, is not likely to be resistant against the untrusted foundry, which may possibly burn the first set of fuses, read out PUF key, and then repair fuses before releasing the chip. To avoid this problem, we implement a lightweight one-sided public key cryptography (encryption only) based on Elliptic-Curve Cryptography (ECC). Considering the PUF readout is a one-time event, the performance of the public-key cryptography engine is not critical.

In order to prevent any attempts at fully characterizing a PUF in the untrusted foundry, only strong PUFs, e.g. an arbiter PUF, are considered. The secure readout of the PUF key is allowed only at the device enrollment time, in the secure facility. During the secure readout, the strong PUF is fed with multiple challenges selected by the Enrollment Authority. The corresponding PUF responses are encrypted by the untrusted chip using the public key of the Enrollment Authority, that is embedded in the chip layout or stored in the one-time programmable memory. Only the Enrollment Authority has access to the decrypted responses. Afterward, one of the previously applied challenges is randomly selected and used for the generation of the secret key. This challenge is then hardwired on the untrusted chip, and the PUF response to that challenge is recorded by the Enrollment Authority. This

PUF response is then stored in the secure memory of the trusted chip in 2.5D-COMA, or in the secure cloud directory in R-COMA. This process makes each PUF key unique to a given device, and resistant to any unauthorized readout by the untrusted foundry.

Still, additional precautions must be taken to protect this scheme against an attack aimed at replacing a real PUF with a pseudo-PUF, generating randomly looking responses that can be easily calculated by an attacker. An example of such a pseudo-PUF may be a lightweight symmetric-key cipher, with a fixed key known to the untrusted party, encrypting each challenge and outputting a ciphertext as the PUF response.

Such pseudo-PUF should be treated as a Trojan and detected by Enrollment Authority using the best known anti-Trojan techniques, e.g., those based on the measurement and analysis of the power consumption during the operation of the device [120]. Additional methods may be used to differentiate the outputs of a strong PUF from encrypted data, e.g., using known correlations between the PUF responses corresponding to closely-related challenges, such as challenges differing on only one bit position, or being mutual complements of each other [121]. These kinds of PUF-health tests may be specific to a particular strong PUF type, e.g., to an arbiter PUF, and will be the subject of our future work.

6.5 COMA Resistance against various Attacks

6.5.1 Assumed Attacker Capabilities

Different sources of vulnerability are considered in this section to demonstrate the COMA security. The attacker can be an adversary in the manufacturing supply chain, and has access to either the reverse engineered or design house-generated netlist of the COMA-protected untrusted chip. The attacker can purchase an activated COMA-protected IC from the market. The attacker can monitor the side channel information of chips at or post activation. The attacker can observe the communication between untrusted and trusted (or remote manager) chips and could also alter the communicated data. An Attack objective may be (1) extracting the obfuscation key (OK), (2) illegal activation of the obfuscated

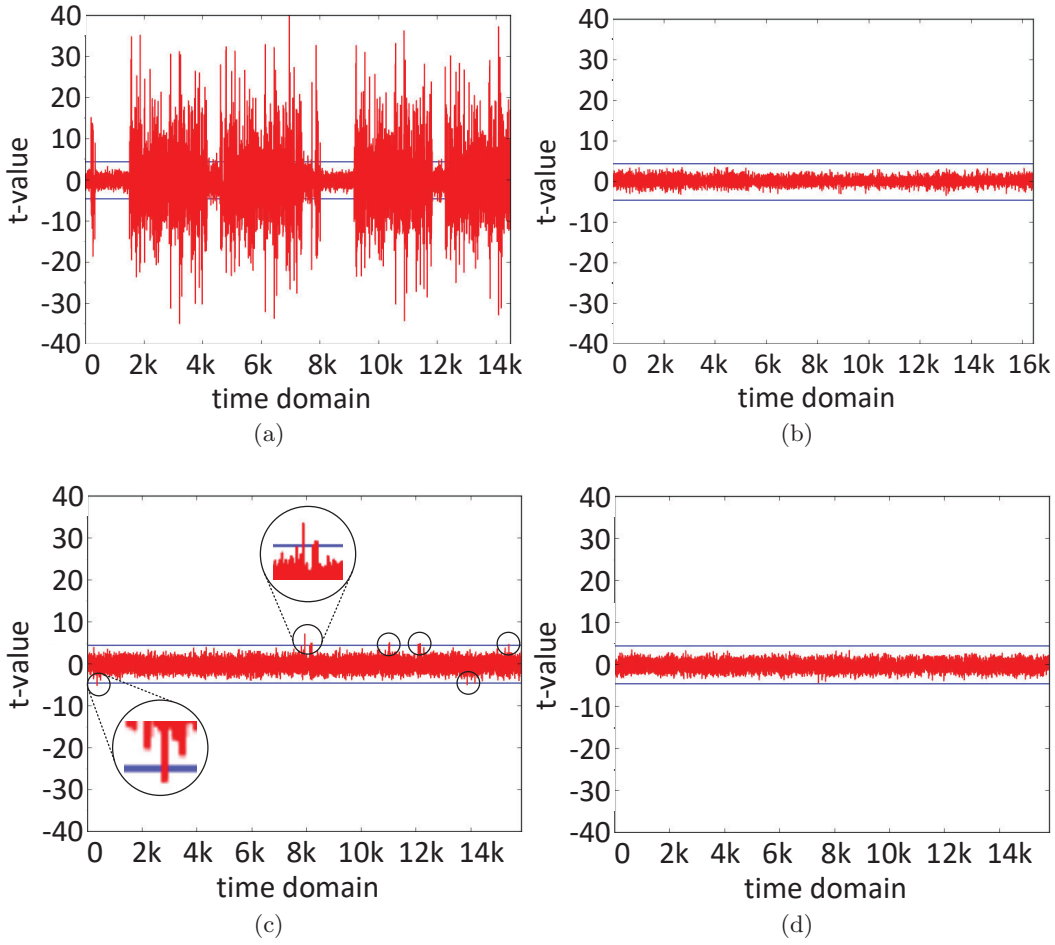


Figure 6.6: The t-test Results for Pr and UnPr version of AES-GCM and ACORN.

circuit without extracting the key, (3) extracting the long-term secret key (SK), (4) extracting short-term CSN keys (TRNs), (5) eavesdropping on messages exchanged between the untrusted chip and the external sources, (6) removing the COMA protection, or (7) COMA-protected IC overproduction.

6.5.2 Side Channel Attack (SCA)

The objective of SCA on COMA is to extract either the secret key (SK) used by AEAD or the TRN used by CSN. Extracting a SK is sufficient to break the obfuscation; extracting a TRN reveals only messages sent in the LCC mode.

DCC significantly increases the SCA difficulty, since (1) the AEAD is side-channel protected, and (2) the attacker loses access to the input of AEAD. Fig. 6.6 captures our assessment of side channel resistance of AEAD using a t-test for unprotected and protected implementations of AES-GCM and ACORN [122]. As illustrated, both implementations pass the t-test, indicating increased resistance against SCA. On the other hand, the inability to control the input to AEAD comes from the COMA requirement of encryption in the DCC mode where a message first passes through the CSN. Hence, there exists no relationship between the power consumption of the AEAD and the original input due to CSN randomization. CSN power consumption is also randomized as it is a function of n inputs (possibly known to the attacker) and $3n \times (log_2 n - 1)$ TRN inputs unknown to the attacker, while the TRN is repeatedly updated based on the value of U. Note that during the physical design of COMA, the side channel information on power and voltage noise (IR drop) could be further mitigated using timing aware IR analysis [123–125], and voltage noise aware clock distribution techniques [126, 127].

The LCC mode is prone to side-channel, algebraic, and SAT attacks aimed at extracting the TRN. However, the attack must be carried out in a limited time while the TRN of the CSN/RCSN is unchanged. As soon as the TRN is renewed, the previous side-channel traces or SAT iterations are useless. The period of TRN updates (U) introduces a trade-off between energy and security and can be pushed to maximum security by changing the TRN for every new input. In section 6.6.3 we investigate the time required to break the LCC using side-channel or SAT attack and accordingly define a safe range for U to prevent such attacks.

6.5.3 Reverse Engineering

In COMA, reverse engineering (RE) to extract the secret key from layout is useless as the secret key is not hardwired in the design and is generated based on PUF. RE to extract the key from memory in an untrusted chip is no longer an option as the key is not stored in the untrusted chip. RE to extract the key from the trusted chip's memory is limited by

the difficulty of tampering with secure memory in the trusted technology.

6.5.4 Algebraic Attacks

Algebraic attacks involve (a) expressing the cipher operations as a system of equations, (b) substituting in known data for some variables, and (c) solving for the key. AES-GCM and ACORN have been demonstrated to be resistant against all known types of algebraic attacks, including linear cryptanalysis. Therefore, in the absence of any new attacks, the DCC mode is resistant against algebraic attacks. Using CSN and RCSN for fast encryption is new and requires more analysis. CSN can be expressed as an affine 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 input TRN. Although recovering A and b is not equivalent to finding the TRN, it may enable the successful decryption of all blocks encrypted using a given TRN. We protect against this threat in two ways: (1) The number of blocks encrypted using a given TRN is set to the value smaller than n, which prevents generating and solving a system of linear equations with A and b treated as unknowns, (2) We partially modify the TRN input of CSN with each block encryption (by simply shifting the input TRN bits), so the values of A and b are not the same in any two encryptions, without the need of feeding CSN with two completely different TRN values.

6.5.5 Counterfeiting and Overproduction

COMA can be used to prevent the resale of used ICs, usage of illegal copies, and reproduction of a design. During packaging and testing, each COMA protected IC is first tested and then is matched with a trusted chip. So, the untrusted chip can only be activated by the matched trusted chip or the registered remote manager. Building illegal copies that work without the secure chip (or remote activation) and reproduction of the design requires successful RE. Blind reproduction is useless as its activation requires a matching trusted chip or passing PUF authentication of a remote manager. By receiving one or more DALs for testing, the manufacturer cannot activate additional IPs as the DAL changes from activation to

activation.

6.5.6 Removal attacks

Removal of the TRNG fixates the DAL and breaks the LCC mode. In DCC mode, it gives an attacker control over the input to the AEAD, increasing the chances of SCA on the cryptography engine. NIST standard SP 800-90B [108] dictates that continuous health testing must be performed on the TRNG. These tests include repetition counting to detect a catastrophic failure and adaptive proportion testing to detect loss of entropy. Removal of the TRNG would be detected as this would result in insufficient entropy to satisfy the health test, assuming the test is implemented on the trusted chip. Removal of COMA architectural modules makes the chip non-functional as COMA is not a wrapper architecture, but a fused one. Complete removal of COMA requires successful RE. Removing the PUF can be made challenging by using a strong PUF, with a large number of challenge-response pairs. Replacing such a PUF with a deterministic function is challenging as such functions are likely to have a substantially different area and power, making them detectable.

Table 6.1: Main features of the two proposed COMA variants.

Feature	COMA1	COMA2
AEAD	AES-GCM	ACORN
PRNG	AES-CTR	Trivium
BUS Width	8	8
Pins used for Communication	8	8
CSN-RCSN Size	64	64
Trusted Memory	4 Kbits	4 Kbits
C_{fix}: initialization overhead (cycles)	10,492	20,452
C_{byte}: cycles needed for encrypting each byte	72	17
$PRNG_{perf}$: Throughput of generating PRN	$128bit/10cycles$	$64bit/cycle$

Table 6.2: Resource Utilization of the COMA Architecture.

Name	AES-GCM+AES-CTR			ACORN+Trivium		
	Slice	LUT	FF	Slice	LUT	FF
TRUSTED						
AEAD_EXT	1,336	3,804	4,432	333	1,067	591
RNG	712	2,226	618	215	601	450
CSN	257	540	739	257	540	739
Others	149	345	144	149	345	144
UNTRUSTED						
AEAD_EXT	1,336	3,804	4,432	333	1,067	591
RNG	738	2,352	628	241	683	460
RCSN	252	607	737	252	607	737
ECC	563	1569	1161	563	1569	1161
PUF [128]	177	—	—	177	—	—
Others	209	359	257	209	359	257

On Xilinx Artix-7 (XC7A100T-1CSG324) FPGA.

6.6 COMA Implementation Results

For evaluation, all designs have been implemented in VHDL and synthesized for both FPGA and ASIC. For ASIC implementation we used Synopsys generic 32nm educational libraries. For FPGA verification, we targeted a small FPGA board, Digilent Nexys-4 DDR with Xilinx Artix-7 (XC7A100T-1CSG324).

6.6.1 COMA Area Overhead

We implemented two variants of COMA architecture: a NIST compliant solution (denoted by COMA1) and a lightweight solution (denoted by COMA2). The AEAD and PRNG in COMA1 are based on AES-GCM and AES-CTR respectively. The COMA2 is implemented by using ACORN for AEAD and Trivium for PRNG, The details of these two variants are summarized in Table 6.1. The breakdown of area (in terms of Slices, LUTs, and FFs) for these solutions for an FPGA implementation in Xilinx Artix-7 is reported in Table 6.2. The

Table 6.3: Resource Utilization for ASIC implementation of COMA.

Name	AES-GCM+AES-CTR				ACORN+Trivium			
	Cells	Area_{um^2}	Tclk_{ns}	Power_{mW}	Cells	Area_{um^2}	Tclk_{ns}	Power_{mW}
COMA	25338	0.11	1.97	1.62	8681	0.046	1.18	0.84
▷ RNG	5684	0.025	1.43	0.431	1267	0.007	0.27	0.144
▷ CSN/RCSN	1749	0.008	0.08	0.11	1749	0.008	0.08	0.11
▷ AEAD	13675	0.061	1.67	0.704	2257	0.013	0.97	0.251
▷ ECC	3278	0.016	1.34	0.321	3278	0.016	1.34	0.321

Using Synopsys generic 32nm libraries.

breakdown of area (in terms of Cells and um^2), critical path, and power consumption for an ASIC implementation is reported in Table 6.3. Note that the 2.5D-COMA needs both the trusted and untrusted parts of the architecture, while the R-COMA only requires the untrusted part. Table 6.4 reports optimized area and frequency results on FPGA for top-level of trusted and untrusted sides. As illustrated, the total area of a lightweight solution is around 1/3 of the NIST-compliant solution. The reported numbers in Table 6.2 include the overhead of all sub-modules including AEAD, CSN-RCSN, RNG, ECC, etc. Due to the optimization on the boundaries among the units, resource utilization in Tables 6.4 is less than the sum of row values in Table 6.2.

6.6.2 COMA Performance

Fig. 6.7 compares the performance of two solutions in DCC and LCC mode. As illustrated, for small data sizes, the COMA1 outperforms the COMA2 solution. However, as the size of data increases, the COMA2 outperforms the COMA1 solution. It is due to the fact that stream ciphers such as ACORN have a long initialization phase, making them inefficient for small data size. In addition, our AES-GCM implementation benefits from an 8-bit datapath, but the ACORN is realized by a 1-bit serial implementation. The total latency in terms

Table 6.4: Optimized results of COMA Architecture.

Name	AES-GCM+AES-CTR				ACORN+Trivium			
	Slice	LUT	FF	Freq[MHz]	Slice	LUT	FF	Freq[MHz]
Trusted	2,297	7,094	5,892	103	1,030	2,901	1,924	121
Untrusted	2,818	8,781	7,169	109	1451	4,182	3,156	120

On Xilinx Artix-7 (XC7A100T-1CSG324) FPGA.

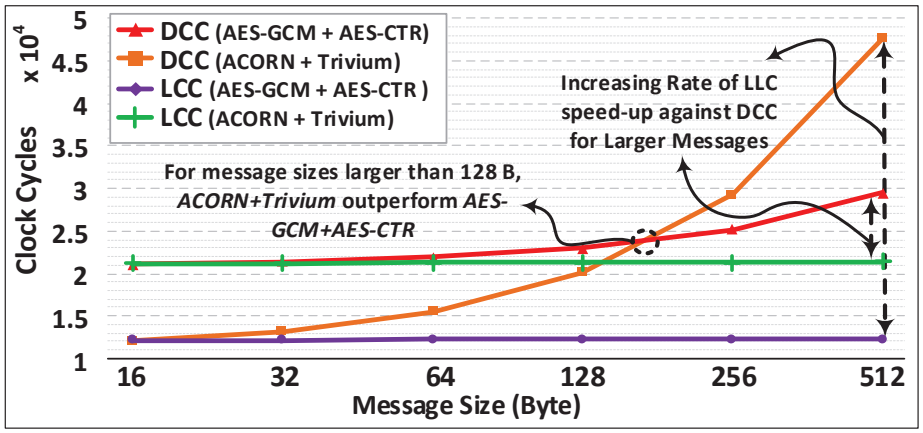


Figure 6.7: Execution Time in AES-GCM+AES-CTR and ACORN+Trivium.

of the number of clock cycles for COMA1 and COMA2 implementations can be calculated using equation (6.1), in which the number of cycles for the initialization and finalization is fixed and is given in Table 6.1. The C_{byte} is the number of cycles needed for encrypting each input message byte, which is 17 and 72 for COMA2 and COMA1, respectively. Hence, in spite of longer initialization, the COMA2 outperforms the COMA1 for message sizes larger than 128 Bytes.

$$T_{comm} = C_{fix} + Message_{size} \times C_{byte} \tag{6.1}$$

6.6.3 COMA performance in LCC mode

In the LCC mode, the AEAD is used to synchronize the initial seed of the PRNG, while the CSN is used for encrypting data. The random (TRN) configuration key for the CSN-RCSN is generated by PRNG, which is updated after transferring every U messages. In COMA, the PRNG has a limited buffer size, and as soon as the buffer is filled with random data, the PRNG stops producing additional bits. The consumption of TRNG output is synchronized (every U messages) and the generation of random inputs is limited to the size of buffer. Hence, the PRNGs in the trusted and untrusted sides are always in sync. The number of cycles it takes to initialize the LCC mode includes the time to initialize the secret key engine (C_{fix}), the encryption and transfer and decryption of PRNG seed (C_{ENC}), and the time for the PRNG to generate enough output from a newly received TRN (C_{PRNG}):

$$C_{LCC-init} = C_{fix} + C_{ENC} + C_{PRNG} \tag{6.2}$$

Depending on the AEAD used for transferring the original seed, the C_{fix} is obtained from Table 6.1. The seed size in our implementation is 16 Bytes, hence the C_{ENC} is simply $C_{bytes} \times 16$, and the C_{PRNG} is:

$$C_{PRNG} = \frac{Bits_{needed}}{PRNG_{perf}} = \frac{3n \times (log_2 n - 1)}{PRNG_{perf}} \tag{6.3}$$

Finally, after initialization, and by using a CSN of size n when the bus width of COMA is BW, the number of cycles to encrypt and transfer one byte of information is:

$$C_{byte}^{LCC} = \frac{8}{n} \times (\frac{n}{BW} + 1) \tag{6.4}$$

Using a 64-bit CSN and BW of 8 bits, the $C_{byte}^{LLC} = 9/8$. Compared to C_{byte}^{DCC} for the COMA1 (C_{byte}^{DCC} =72), and for the COMA2 (C_{byte}^{DCC}=17), the LCC mode is at least an order of magnitude faster. Fig. 6.7 compares the superior performance of LCC mode compare

Table 6.5: SAT Execution Time on Blocking CSN and a Close to Non-blocking CSN .

CSN Size	4		8		16		32		64		128		256		512	
Mode	blk	non-blk	blk	non-blk	blk	non-blk	blk	non-blk	blk	non-blk	blk	non-blk	blk	non-blk	blk	non-blk
SAT Iterations	6	14	7	18	8	25	12	31	14	TO	24	TO	25	TO	TO	TO
SAT Exe. Time (s)	0.01	0.01	0.03	0.15	0.2	2.35	0.8	79.18	5.9	TO	130.5	TO	1136.2	TO	TO	TO

TO: Timeout = 2×10^6 seconds; The SAT attack is carried on a Dell PowerEdge R620 equipped with Intel Xeon E5-2670 2.6 GHz and 64GB of RAM.

with DCC mode in both COMA variants.

Frequency of TRN updates in LCC mode

The frequency of TRN update (U) for LCC is an important design feature. A large U reduces energy as PRNG/TRNG is kept idle for $U - P$ cycles. P is the number of required cycles to refill the PRNG buffer after a TRN read. However, when the TRN is fixated for a long duration of time, the possibility of a successful side-channel, algebraic, or SAT attack on the CSN increases. The minimum number of messages required for an algebraic attack (even if such attack is possible) is n, which is the CSN input size. Our experiments show that a SAT attack could recover the key with an even smaller number of inputs. Knowing the number of encryptions/decryptions needed by such attacks, we can set the U to a safe value smaller than the number of required messages to make it resistant to these attacks. So, the value of U should be between $P \leq U \leq n$.

The SAT attack against CSN is implemented similar to [1]. In this attack, the CSN gate-level netlist and an activated chip are available to the attacker, while the attacker aims to extract the CSN-RCSN configuration signals. Table 6.5 captures the results of the SAT attack against blocking and near non-blocking CSNs. As illustrated, the time to break a near non-blocking CSN is significantly larger. In each iteration SAT test one carefully selected input message. Hence, if the U is kept smaller than the number of required SAT iterations, the SAT attack could not be completed.

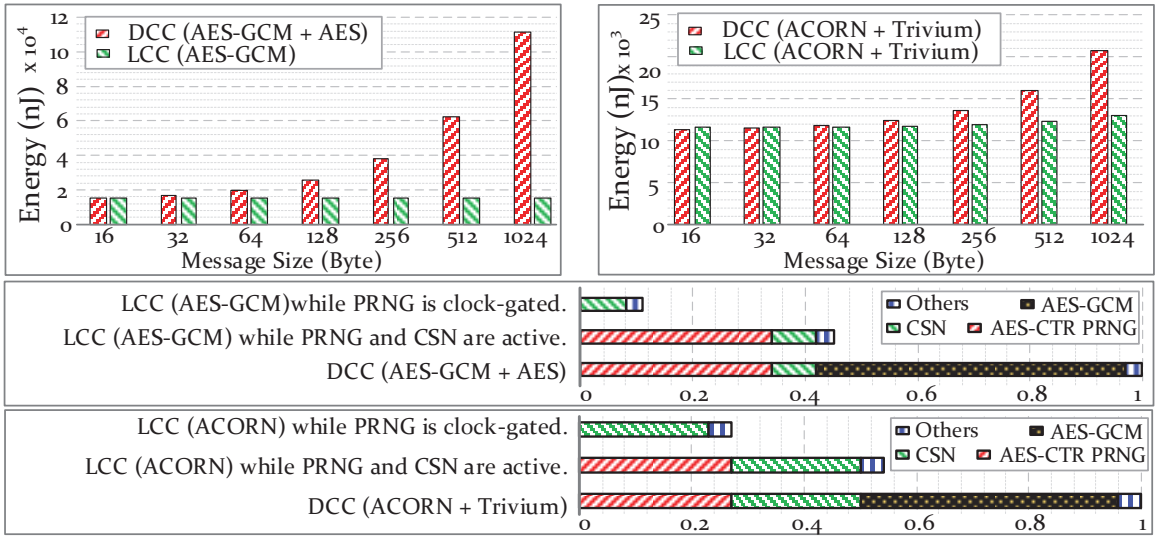


Figure 6.8: Energy Breakdown in COMA.

Energy saving in LCC mode

As illustrated in Fig. 6.9(a), in the LCC mode, the TRN is updated every U cycles. U is determined based on the fastest attack on CSN-RCSN pair, which is the SAT attack. After each TRN update, the PRNG takes P cycles to refill its buffer. Note that P cycles required for PRNG could be stacked at the beginning of U cycles, or distributed over U cycles depending on the size of the PRNG buffer. As long as the TRN completely changes every U cycle, the possibility of attack is eliminated. Hence in each U cycles, for P cycles the PRNG/TRNG and CSN are active, and for $U - P$ cycles, the PRNG is clock gated, and only CSN is active. In both cases, the AEAD is active only for the initial exchange of PRNG seed, allowing us to express the power consumption of the LCC mode as:

$$E_{LCC} = C_{PRNG} \times P_H + \big(U(\frac{n}{BW} + 1) - C_{PRNG}\big) \times P_L \tag{6.5}$$

Obviously, the number of required cycles to refill the PRNG buffer after TRN read (P) affects energy consumption and communication throughput. If $P < U$, as illustrated in Fig. 6.9(a), for $U - P$ cycles the PRNG is kept idle (power-gated). However, if $P > U$, as shown

Table 6.6: COMA vs. FORTIS.

Scheme	Key Management	Data Communication	Private Key	SC Protected
FORTIS	Constant	✗*	Embedded (known to the fab)	✗
COMA	PUF-based Unique	✓+	No private key at untrusted	✓

Scheme	Session Key	Activation	Need to TPM	Source of Randomness
FORTIS	Vulnerable to Fault Attack	Once	at Untrusted Side	Pseudo RNG
COMA	Secure	per Demand (License)	at Trusted Side	True RNG

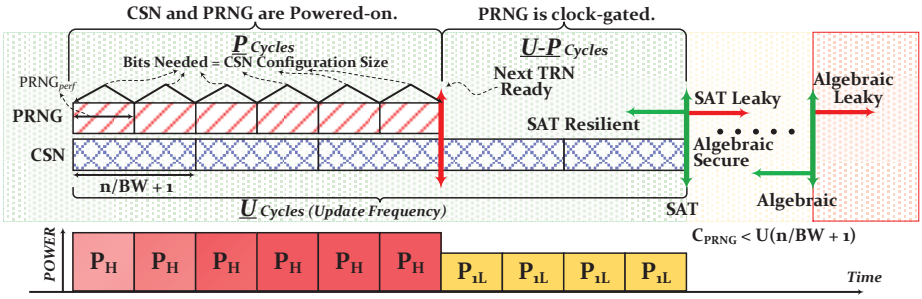
**: Not Implemented, but Naturally available using OTP. Limited Performance Due to Lightweight RSA*
+: Available in Two Variant: DCC (Fully Secure and Limited Performance) and LCC (Leaky yet Secure and High Performance).

Table 6.7: Area Overhead of COMA vs. FORTIS.

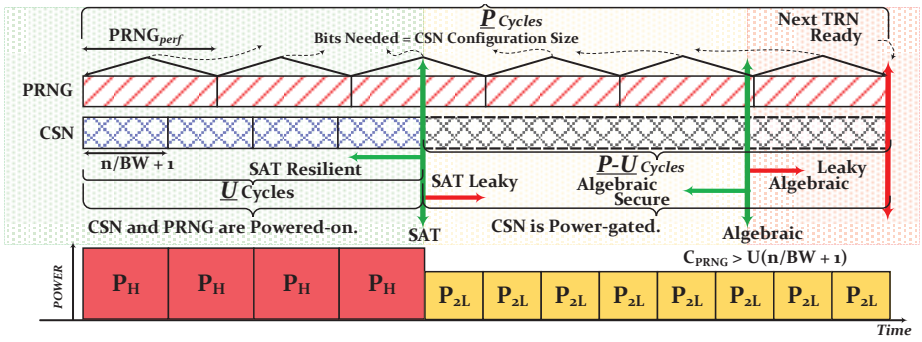
Design	Gate Count	FORTIS/Design	COMA1/Design	COMA2/Design
b19	40,789	24.52%	62.1%	21.28%
VGA_LCD	43,346	23.07%	58.45%	20.02%
Leon3MP	253,050	3.95%	10.01%	3.43%
SPARC	836,865	1.19%	3.02%	1.03%
Virtex-7	2M	0.5%	1.26%	0.43%

in Fig. 6.9(b), the communication should be stopped for $P - U$ cycles till the next TRN is ready and to resist SAT or algebraic attacks.

The energy consumption of LCC mode for COMA architectures constructed using NIST-compliant and lightweight solution when transmitting different size of messages is captured in Fig. 6.8. As illustrated, the LCC mode, for having to synchronize the two sides using a TRNG seed, is burdened with the initialization cost of AEAD. However, when the CSN-RCSN and PRNG are setup, the energy consumed for exchanging additional messages grow at a much lower rate compare to DCC mode (which is dominated by AEAD and PRNG power consumption (as reported in table 6.3).



(a) While $P < U$



(b) While $P > U$

Figure 6.9: Power Consumption of LCC Mode of Communication.

6.7 Comparing COMA with Prior Work

To the best of our knowledge, FORTIS [97] is the only comprehensive key-management scheme that was previously proposed. Table 6.6 compares our proposed solution against FORTIS. COMA addresses several shortcomings of the FORTIS:

1) In FORTIS, all chips use identical keys, hence there is no mean of differentiating between chips. In COMA each chip has a unique key generated by PUF. 2) In COMA, secret key for communication and authentication is generated by PUF, when FORTIS relies on embedding the private key and public key in GDSII. So, the private key in FORTIS will be known to the fabrication posing the risk that the entire process of activation could be faked in software. In COMA, such attack is prevented as secret key is generated by PUF and is securely read out using public key cryptography. 3) In FORTIS, the usage of the private key for chip authentication is vulnerable to SCA. In COMA, the secret-key cryptography

is side channel protected, and the public-key encryption is only used once, making COMA secure against SCA. 4) In FORTIS, there is also the possibility of deploying a fault attack by fixing the value of session key Ks. In COMA, the same attack would require fixing the PUF output or replacing the PUF with a known function. This however could be tested by reading out the output of the PUF using multiple challenges and performing statistical test on the PUF response (PUF health check). 5) In FORTIS, the activation is done once, hence there is a need to store the obfuscation key in the untrusted chip. In COMA, the need to store the obfuscation key in untrusted chip is removed. In R-COMA, the activation takes place on demand, and the key is removed after power down or reset. In 2.5D-COMA, the activation key is stored in a trusted chip. 6) COMA provides two new mechanisms for communication: a) the DCC mode for added security, and b) the LCC mode for high-speed communication. 7) COMA uses a TRNG to produce the seed for PRNG, while FORTIS uses a PRNG without addressing a random source for its seed.

In terms of area overhead, FORTIS [97] provides an estimate for the incurred overhead of their solution, which is around 10K gates. As shown in Table 6.3, the number of cells for implementing the NIST-compliant (COMA1) implementation is 25.4K gates, while the lightweight solution (COMA2) is implemented using 8.7K gates. Table 6.7 compares the area overhead of FORTIS against COMA1 and COMA2 when these architectures are deployed to protect a few mid- and large-size benchmarks. Using COMA2, which improves the overhead by 14% compared to FORTIS, requires between 0.43% and 21.3% of circuit area in selected benchmarks.

6.8 What we Learnt in this Chapter

In this Chapter, we presented COMA, an architecture for obfuscation-key management and metered activation of an obfuscated IC that is manufactured in an untrusted foundry, while securing its communication. The proposed solution removes the need to store the key in

the untrusted chip, makes the obfuscation unlock-key a moving target, allows unique identification of the protected IC, and secures the communication to/from the protected chip using two hybrid cryptographic schemes for ultra-high-speed and ultra-security. Our experimental results showed that compared to the state-of-the-art key management architecture, FORTIS, COMA is able to reduce the area overhead by 14%, while addressing many of the shortcomings of the previous work.

Chapter 7: Conclusion

The ever-increasing cost of integrated circuits (IC) manufacturing, increasing the recurring cost of the fab maintenance, and having access to the most advanced technology node have pushed many high-tech companies to become fabless. Outsourcing the stages of the manufacturing supply chain to the third-party facilities with no reliable monitoring on them results in the introduction of multiple forms of security threats such as IC overproduction, Trojan insertion, Reverse Engineering (RE), Intellectual Property (IP) theft, and counterfeiting. To combat these threats, different design-for-trust (DfTr) countermeasure techniques have been proposed in the literature. Amongst them, logic obfuscation *a.k.a.* logic locking, in which post-manufacturing programming capability could be added into the circuits, received significant attention in recent years as a proactive and robust countermeasure.

logic locking introduces limited programmability into a netlist through inserting additional key programmable gates at design time. After fabrication, the functionality of the IC is programmed by loading the correct key values. To resist the physical attacks, such as reverse-engineering, the key inputs could be stored in and driven by an on-chip tamper-proof memory. The purpose of inserting key gates is to protect the IC design from untrusted foundries. Since the functionality of a design is locked with a secret key, the attacker cannot learn the functionality of the obfuscated netlist after reverse engineering.

Due to the importance of logic locking, many studies have evaluated the effectiveness (strength) of this paradigm in different abstraction levels. Amongst all state-of-the-art threats on logic obfuscation, the Boolean satisfiability (SAT) attack has seriously challenged the effectiveness of the vast majority of existing logic locking solutions. After the introduction of the SAT attack in 2015, researchers have proposed various mechanisms for building SAT hard obfuscation solutions. However, many of such obfuscation schemes were

later broken using newer attacks such as SPS, removal, bypass, and AppSAT, making the current defense schemes unreliable.

Recently, a new breed of obfuscation schemes has been introduced, relied on breaking the SAT assumptions for building SAT hard solutions without having the vulnerabilities of the previous SAT hard solution. For example, Cyclic obfuscation, by introducing cycles into netlist break the SAT model as the netlist can no longer be represented by a directed acyclic graph (DAG). Alternatively, the delay logic locking (DLL) extends the obfuscation representation beyond logic and locks the circuit using its delay and timing properties, attempting to build SAT hard solutions. Another group of countermeasures tries to significantly increase the runtime of *each iteration* of the SAT solver. In such techniques, by exploiting the strength of symmetric routing structures, such as permutation networks or crossbars, the complexity of the SAT circuit per each iteration will be increased significantly.

7.1 What We Learnt in this Thesis

The objective of this thesis was to develop newer attacks, capable of evaluating and assessment of the breaking possibility of the obfuscation solutions that cannot be breakable by state-of-the-art attacks. However, this thesis went one step further, and apart from proposing newer powerful attacks, it also covers two different defense solutions, one as a new robust logic locking solution, and one as a key management architecture for securing the logic locking key. The new logic locking solution could resist the existing attacks on logic locking, and due to its nature, it could be known as a new promising paradigm in the literature. Also, the key management architecture helps double-securing the activation of obfuscated circuits that are manufactured in untrusted foundries to protect the IC from being reverse engineered and stolen.

As the first newer attack with higher capability and performance beyond the existing attacks, in Chapter 3, we introduced a class of satisfiability modulo theory (SMT) attacks that could break behavioral logic locking techniques. The SMT attack benefits from the expressive nature of theory solvers, which allow the attacker to express constraints that are

difficult or even impossible to express using CNF, including timing, delay, power, arithmetic, graph, and many other first-order theories. We first illustrated that a SAT attack could be easily implemented using an SMT solver to prove that SMT attack is a superset of the SAT attack. Then we proposed two variants of SMT attack on obfuscated circuits using the Eager and Lazy approach of SMT solver. We illustrated that using the Eager and Lazy approach, we could break the delay logic locking (DLL) obfuscation that cannot be broken by a SAT attack, proving that SMT attack's capabilities go beyond a SAT attack. It shows that by only using non-logical properties of a netlist for obfuscation, we do not provably increase the security of an obfuscated netlist, indicating the need for further study and exploration in this domain to generate obfuscation schemes with provable security. Then we proposed the Accelerated SMT attack (AccSMT), and we illustrated that by using theory solvers (BitVector theory solver in this experiment), we could significantly speed up the attack against specific obfuscated circuits, and reported a significant reduction in the execution time of the AccSMT compared to the SAT attack. Finally, we illustrated that with a small modification, the AccSMT could be used as an approximate attack, allowing us to find an approximate key for obfuscation schemes that combine a SAT hard obfuscation with high corruption obfuscation.

To further extend the capability of the SAT attack, especially in terms of the scalability when complex modules are in place, in Chapter 4, we proposed a neural-network-guided SAT attack, called NNgSAT attack. In the NNgSAT attack, we use a message passing neural network to predict satisfying assignments that could help and significantly speed up the conventional SAT attack in solving the design that contains complex *hard-to-be-solved* structures (e.g. routing obfuscation [32,33]). In NNgSAT, after being trained as a classifier to predict SAT/UNSAT on a SAT problem, the neural network is used to guide/help the actual SAT solver for finding the SAT assignment(s). By training NN on conjunctive normal forms (CNFs) corresponded to a dataset of logic locked circuits, our experiments showed that NNgSAT could solve 93.5% of the logic locked circuits containing complex structures within a reasonable time, while the existing SAT attack cannot proceed the attack flow in

them.

With the combination of both proposed attacks in this thesis, i.e. the SMT attack and the NNgSAT attack, as an evaluation framework on logic locking, we would be able to assess and break a wide range of state-of-the-art logic locking techniques, including but not limited to, pre-SAT (primitive) techniques, point function that combined with primitive techniques (compound), cyclic and behavioral, and routing-based obfuscation techniques. Also, with the integration of BMC that could be easily integrated with SMT attack, sequential SAT attack could be emulated using the proposed approaches. Having such strong and comprehensive framework allows the designers to evaluate the security of newer logic locking techniques using a unified and well-designed attack framework.

Considering these new attacks, in Chapter 5, to combat state-of-the-art attacks on logic obfuscation, including but not limited to SAT, Sequential SAT, SMT, and NNgSAT, we proposed a new obfuscation paradigm called data flow obfuscation. In data flow obfuscation, by exploiting the concept of asynchronicity, we showed how the flow of the data could be obfuscated in any arbitrary circuit. Locking the flow of the data will affect the timing of moving data within the circuit, making it almost impossible for the adversary to model any form of attack on it. In data flow obfuscation, we engaged false {paths + latches} using the asynchronous structure to control the flow of data in specific timing paths. Using this mechanism, we showed that since the time of data capturing in scan chains is locked, the SAT attack has no longer an advantage for the adversary even while the scan access is not restricted. Also, we showed that how asynchronicity combat the sequential SAT attack by invalidating the unrolling step in these attacks. We comprehensively investigated the effectiveness of this new obfuscation paradigm over wide-range benchmark families. Our experiments showed the resiliency of this new paradigm against all existing attacks at significantly low overhead.

Furthermore, to boost the security of logic locking techniques, considering the possibility of direct access/extract of the logic locking key, in Chapter 6, we went one step further

to protect the IC from being reverse engineered or stolen. We introduced a novel *communication and obfuscation management architecture* (COMA) to handle the storage of the obfuscation key and to secure the communication to/from the obfuscated circuits that are manufactured in untrusted foundries and meet the constant connectivity requirement, namely ICs that belong to a) 2.5D package-integrated and b) IoT solutions. COMA addresses three challenges related to the obfuscated circuits: First, it removes the need for the storage of the *obfuscation unlock key* at the untrusted chip. Second, it implements a mechanism by which the key sent for unlocking an obfuscated circuit changes after each activation (even for the same device), transforming the key into a dynamically changing license. Third, it protects the communication to/from the COMA protected device and additionally introduces two novel mechanisms for the exchange of data to/from COMA protected architectures: (1) a highly secure but slow double encryption, which is used for the exchange of key and sensitive data (2) a high-performance and low-energy yet leaky encryption, secured by means of frequent key renewal. We demonstrated that compared to state-of-the-art key management architectures, COMA reduces the area overhead by 14%, while allowing additional features including unique chip authentication, enabling activation as a service (for IoT devices), reducing the side channel threats on key management architecture, and providing two new means of secure communication to/from an untrusted chip.

7.2 Future Directions

For future work, we anticipate the following research directions:

Extending and Maturation of Attack Framework on Logic Locking: With the combination of different attacks introduced on logic locking, including the SMT attack and the NNgSAT attack, the framework of security evaluation on logic locking will become more powerful, requiring a much more sophisticated countermeasure to resist against this framework. Formal verification tools can play an important role in this area for making a

much more comprehensive attack framework. Many of the existing attacks on logic locking have engaged methods and mechanisms used in such formal tools. However, none of the existing attacks integrates any formal verification tool as a means and major part of the attack on logic locking. Particularly, the availability of commercial formal verification tools with high scalability could significantly mitigate the scalability and performance issue of the existing attack in a more standard way. Boosting the strength of the attack framework using the formal tools helps the researchers to evaluate new logic locking countermeasures with a standard and much stronger tool, helping to move faster towards the sophistication of the logic locking algorithms.

Extending and Maturation of Scan-based Logic Locking Countermeasures: Depending on the state-of-the-art attacks introduced on logic locking, new studies move towards the introduction of methods that are resilient against newer attacks. For instance, since the SAT, SMT, NNgSAT, and many of the other attacks require access to the scan chain to target each combinational logic part separately, many recent studies attempt to evaluate and assess the restricting access to the scan chain architecture for restricting any unauthorized scan chain access. In this case, the adversary has only access to primary inputs/outputs (PI/PO). Since almost all of the circuits are sequential, and the SAT attack cannot formulate flip-flop (the state), the attacker is no longer able to use the SAT attack. However, none of these techniques offer provable security guarantees. For example, the introduction of sequential-based SAT attacks, such as KC2, shows that the attack could be still formulated even while the scan access is restricted. However, the existing sequential SAT attacks suffer from low scalability especially when large circuits are in place. Hence, securing the scan using a well-designed and robust approach is still an open and promising direction in logic locking. The main shortcoming of the existing locking/blocking scan chain technique is that they impose critical restrictions on the circuits. For example, in some cases it is assumed that the tester is a trusted party that should have the correct key; however, it is a hard assumption to maintain in many practical cases. In some other cases, the tester has to rely on observing the PO for any test/debug purposes, such as functional test, which

might reduce the testability coverage depending on the topological hierarchy of the design-under-test. Hence, there is a need to revisit the security of such obfuscation techniques to overcome these drawbacks and to guarantee the resiliency provided by a restricted scan chain architecture, which will be the subject of our future work.

Extending and Maturation of Asynchronous and Latch-based Logic Locking: With the introduction of data flow obfuscation, as a new paradigm for obfuscating the circuit, future studies on logic locking might be redirected towards this new paradigm, in which the concept of latch-based or clock-gating-based logic locking is used for obfuscation purpose. Since most state-of-the-art attacks on logic locking exploit the methods and mechanisms used in existing formal verification tools, focusing on the clock tree or the timing of the circuit, and finding the correlation between the logic (function) and the timing (events) would be hard/impossible to be modeled. However, data flow obfuscation as a preliminary investigation on this breed requires more evaluation/assessment, might resulting in the introduction of extended/advanced techniques relying on this new concept.

Mixture of Analog and Digital Logic Locking: More recent studies show their interest once Boolean (digital) logic locking is mixed with complex analog logic locking. For instance, the usage of chaotic computing as a means of logic locking, helps the designer to add configurable dynamicity into the locked circuit. Adding dynamicity significantly boosts the complexity of the attack formulation. Particularly, in many cases, such as the SAT attack or the SMT attack, any dynamic change during the de-obfuscation will invalidate all previous computations. Hence, the attacks fail to be applied in such cases. Also, a complex analog structure that is hard to be modeled using the SMT attack would be an open research direction.

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.

[**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.

[**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 2019 on Great Lakes Symposium on VLSI, pp. 1-7, 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.

[**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.

Bibliography

Bibliography

[1] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2015, pp. 137–143.

[2] M. El Massad, S. Garg, and M. V. Tripunitara, "IC Decamouflaging: Reverse Engineering Camouflaged ICs within Minutes," in *Network and Distributed System Security Symposium (NDSS)*, 2015, pp. 1–14.

[3] L. Alrahis *et al.*, "Scansat: unlocking obfuscated scan chains," in *Proc of the Asia and South Pacific Design Automation Conf.*, 2019, pp. 352–357.

[4] N. Limaye and O. Sinanoglu, "DynUnlock: Unlocking Scan Chains Obfuscated using Dynamic Keys," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2020.

[5] U. Guin *et al.*, "Robust design-for-security architecture for enabling trust in ic manufacturing and test," *IEEE Trans. on Very Large Scale Integration Systems*, vol. 26, no. 5, pp. 818–830, 2018.

[6] J. Sweeney *et al.*, "Latch-Based Logic Locking," *arXiv preprint arXiv:2005.10649*, 2020.

[7] A. Yeh, "Trends in the global ic design service market," *DIGITIMES research*, 2012.

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

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

[10] K. M. Goertzel and B. Hamilton, "Integrated circuit security threats and hardware assurance countermeasures," *CrossTalk*, vol. 26, no. 6, pp. 33–38, 2013.

[11] S. E. Quadir, J. Chen, D. Forte, N. Asadizanjani, S. Shahbazmohamadi, L. Wang, J. Chandy, and M. Tehranipoor, "A survey on chip to system reverse engineering," *ACM journal on emerging technologies in computing systems (JETC)*, vol. 13, no. 1, pp. 1–34, 2016.

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

[13] A. B. Kahng *et al.*, "Watermarking Techniques for Intellectual Property Protection," in *Design Automation Conference (DAC)*, 1998, pp. 776–781.

[14] F. Imeson, A. Emtenan, S. Garg, and M. V. Tripunitara, "Securing computer hardware using 3d integrated circuit (ic) technology and split manufacturing for obfuscation." in *USENIX Security Symposium*, 2013, pp. 495–510.

[15] R. P. Cocchi, L. W. Chow, J. P. Baukus, and B. J. Wang, "Method and Apparatus for Camouflaging a Standard Cell based Integrated Circuit with Micro Circuits and Post Processing," 2013, uS Patent.

[16] J. A. Roy, F. Koushanfar, I. L. Markov, "EPIC: Ending Piracy of Integrated Circuits," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2008, pp. 1069–1074.

[17] 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.

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

[19] 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.

[20] G. Kolhe, H. M. Kamali, M. Naicker, T. D. Sheaves, H. Mahmoodi, S. M. PD, H. Homayoun, S. Rafatirad, and A. Sasan, "Security and Complexity Analysis of LUT-based Obfuscation: From Blueprint to Reality," in *IEEE/ACM Int'l Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[21] Y. Xie and A. Srivastava, "Mitigating Sat Attack on Logic Locking."

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

[23] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017, pp. 1601–1618.

[24] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal attacks on logic locking and camouflaging techniques," *IEEE Transactions on Emerging Topics in Computing*, no. 1, pp. 1–1, 2017.

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

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

[27] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "Appsat: Approximately deobfuscating integrated circuits," in *Hardware Oriented Security and Trust (HOST), IEEE Int'l Symposium on*, 2017, pp. 95–100.

[28] 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.

[29] 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*, pp. 1–14, 2020.

[30] H. M. Kamali *et al.*, "Chaolock: Yet another sat-hard logic locking using chaos computing," in *International Symposium on Quality Electronic Design (ISQED)*, 2021, pp. 1–8.

[31] Y. Xie and A. Srivastava, "Delay Locking: Security Enhancement of Logic Locking against IC Counterfeiting and Overproduction," in *Proceedings of Design Automation Conference (DAC)*, 2017, pp. 1–6.

[32] K. Shamsi *et al.*, "Cross-lock: Dense layout-level interconnect locking using cross-bar architectures," in *GLSVLSI*, 2018, pp. 147–152.

[33] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Full-Lock: Hard Distributions of SAT Instances for Obfuscating Circuits Using Fully Configurable Logic and Routing Blocks," in *Proceedings of the Annual Design Automation Conference (DAC)*, 2019, pp. 89:1–89:6.

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

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

[36] K. Zamiri Azar, H. Mardani Kamali, H. Homayoun, and A. Sasan, "Threats on logic locking: A decade later," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, 2019, pp. 471–476.

[37] 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.

[38] S. M. Plaza and I. L. Markov, "Solving the third-shift problem in ic piracy with test-aware logic locking," *IEEE Trans. on CAD (TCAD)*, vol. 34, no. 6, pp. 961–971, 2015.

[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] Y. Shen *et al.*, "Sat-based bit-flipping attack on logic encryptions," in *DATE*, 2018, pp. 629–632.

[41] M. Li *et al.*, "Provably secure camouflaging strategy for ic protection," *IEEE Trans. on CAD (TCAD)*, 2017.

[42] M. Yasin *et al.*, "What to lock?: Functional and parametric locking," in *GLSVLSI*, 2017, pp. 351–356.

[43] 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.

[44] H. Zhou, R. Jiang, and S. Kong, "Cycsat: Sat-based attack on cyclic logic encryptions," in *Proceedings of the Int'l Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 49–56.

[45] Y. Shen *et al.*, "Besat: behavioral sat-based attack on cyclic logic encryption," in *ASP-DAC*. ACM, 2019, pp. 657–662.

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

[47] R. Karmakar, S. Chatopadhyay, and R. Kapur, "Encrypt flip-flop: A novel logic encryption technique for sequential circuits," *arXiv preprint arXiv:1801.04961*, 2018.

[48] 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," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019.

[49] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "On designing secure and robust scan chain for protecting obfuscated logic," in *Great Lakes Symposium on VLSI (GLSVLSI)*, 2020, pp. 1–6.

[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 *VLSI Test Symposium (VTS)*, 2020, pp. 1–6.

[51] H. M. Kamali *et al.*, "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.

[52] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "From cryptography to logic locking: A survey on the architecture evolution of secure scan chains," *IEEE Access*, 2021.

[53] 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.

[54] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, "KC2: Key-condition crunching for fast sequential circuit deobfuscation," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2019, pp. 534–539.

[55] Z. Hassan, Y. Zhang, and F. Somenzi, "A Study of Sweeping Algorithms in the Context of Model Checking," *Ganesh Gopalakrishnan University of Utah USA*, p. 30, 2011.

[56] 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.

[57] 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.

[58] D. Mitchell *et al.*, "Hard and Easy Distributions of SAT Problems," in *AAAI*, vol. 92, 1992, pp. 459–465.

[59] G. Nelson and D. C. Oppen, "Fast decision procedures based on congruence closure," *Journal of the ACM (JACM)*, vol. 27, no. 2, pp. 356–364, 1980.

[60] R. E. Shostak, "Deciding combinations of theories," in *Int'l Conference on Automated Deduction*, 1982, pp. 209–222.

[61] C. Tinelli and C. Ringeissen, "Unions of non-disjoint theories and combinations of satisfiability procedures," *Theoretical Computer Science*, vol. 290, no. 1, pp. 291–353, 2003.

[62] C. Barrett and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Model Checking*, 2018, pp. 305–343.

[63] S. Bayless, N. Bayless, H. H. Hoos, and A. J. Hu, "Sat modulo monotonic theories." in *AAAI*, 2015, pp. 3702–3709.

[64] S. Dupuis, P.-S. Ba, G. Di Natale, M.-L. Flottes, and B. Rouzeyre, "A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans," in *IEEE Int'l Symposium on On-Line Testing And Robust System Design (IOLTS)*. IEEE, 2014, pp. 49–54.

[65] C. Oh, "Improving SAT solvers by exploiting empirical characteristics of CDCL," Ph.D. dissertation, New York University, 2016.

[66] A. Mirzaeian, H. Homayoun, and A. Sasan, "Tcd-npe: A re-configurable and efficient neural processing engine, powered by novel temporal-carry-deferring macs," in *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2019, pp. 1–8.

[67] ——, "Nesta: Hamming weight compression-based neural proc. engineali mirzaeian," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2020, pp. 530–537.

[68] A. Mirzaeian, S. Manoj, A. Vakil, H. Homayoun, and A. Sasan, "Conditional classification: A solution for computational energy reduction," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2021, pp. 325–330.

[69] A. Mirzaeian, J. Kosecka, H. Homayoun, T. Mohsenin, and A. Sasan, "Diverse knowledge distillation (dkd): A solution for improving the robustness of ensemble models against adversarial attacks," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2021, pp. 319–324.

[70] Z. Chen, L. Zhang, G. Kolhe, H. M. Kamali, S. Rafatirad, S. M. Pudukotai Dinakarrao, H. Homayoun, C.-T. Lu, and L. Zhao, "Deep graph learning for circuit deobfuscation," *Frontiers in Big Data*, vol. 4, p. 12, 2021.

[71] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill, "Learning a sat solver from single-bit supervision," *arXiv preprint arXiv:1802.03685*, 2018.

[72] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[73] H. M. Kamali and A. Sasan, "Much-swift: A high-throughput multi-core hw/sw co-design k-means clustering architecture."

[74] D. S. *et al.*, "Guiding high-performance SAT solvers with unsat-core predictions."

[75] D. P. Kingma *et al.*, "ADAM: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[76] 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*, vol. 29, no. 4, pp. 643–656, 2021.

[77] 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 Trans. on CHES (TCHES)*, pp. 97–122, 2019.

[78] K. Z. Azar *et al.*, "NNgSAT: Neural Network guided SAT Attackon Logic Locked Complex Structures," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2020, pp. 1–9.

[79] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security Analysis of Integrated Circuit Camouflaging," in *ACM SIGSAC Conference on Computer & Communications Security (CCS)*, 2013, pp. 709–720.

[80] S. M. Nowick and M. Singh, "Asynchronous Design—Part 1: Overview and Recent Advances," *IEEE Design & Test*, vol. 32, no. 3, pp. 5–18, 2015.

[81] V. Khomenko, M. Koutny, and A. Yakovlev, "Logic Synthesis for Asynchronous Circuits based on STG unfoldings and Incremental SAT," *Fundamenta Informaticae*, vol. 70, no. 1, 2, pp. 49–73, 2006.

[82] A. Moreno, D. Sokolov, and J. Cortadella, "Synthesis from Waveform Transition Graphs," in *International Symposium on Asynchronous Circuits and Systems (ASYNC)*. IEEE, 2019, pp. 60–67.

[83] S. Ataei and R. Manohar, "AMC: An asynchronous memory compiler," in *International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2019, pp. 1–8.

[84] M. Fiorentino, C. Thibeault, Y. Savaria, F. Gagnon, T. Awad, D. Morrissey, and M. Laurence, "AnARM: A 28nm Energy Efficient ARM Processor based on Octasic Asynchronous Technology," in *International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2019, pp. 58–59.

[85] E. Beigne, P. Vivet, Y. Thonnart, J.-F. Christmann, and F. Clermidy, "Asynchronous Circuit Designs for the Internet of Everything: A Methodology for Ultra Low-Power Circuits with GALS Architecture," *IEEE Solid-State Circuits Magazine*, vol. 8, no. 4, pp. 39–47, 2016.

[86] D. Sokolov, V. Dubikhin, V. Khomenko, D. Lloyd, A. Mokhov, and A. Yakovlev, Alex, "Benefits of Asynchronous Control for Analog Electronics: Multiphase Buck Case Study," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2017, pp. 1751–1756.

[87] J. Cortadella *et al.*, "A Tool for Manipulating Concurrent Specifications and Synthesis of Asynchronous Controllers," *IEICE*, vol. 80, no. 3, pp. 315–325, 1997.

[88] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[89] J. Cortadella, A. Kondratyev, L. Lavagno, C. P. Sotiriou, "Desynchronization: Synthesis of asynchronous circuits from synchronous specifications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 1904–1921, 2006.

[90] B. Van Antwerpen, M. D. Hutton, G. Baeckler, and R. Yuan, "Register Retiming Technique," Oct. 10 2006, uS Patent 7,120,883.

[91] S. Furber and P. Day, "Four-phase Micropipeline Latch Control Circuits," *IEEE Transactions on Very Large Scale Integration (VLSI)*, vol. 4, no. 2, pp. 247–253, 1996.

[92] H. Hulgaard, S. M. Burns, and G. Borriello, "Testing Asynchronous Circuits: A Survey," *Integration*, vol. 19, no. 3, pp. 111–131, 1995.

[93] A. Bouzafour, M. Renaudin, H. Garavel, R. Mateescu, and W. Serwe, Wendelin, "Model-checking Synthesizable Systemverilog Descriptions of Asynchronous Circuits," in *International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2018, pp. 34–42.

[94] G. Tarawneh and A. Mokhov, "Formal Verification of Mixed Synchronous Asynchronous Systems using Industrial Tools," in *International Symposium on Asynchronous Circuits and Systems (ASYNC)*. IEEE, 2018, pp. 43–50.

[95] 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.

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

[97] U. Guin, Q. Shi, D. Forte, and M. M. Tehranipoor, "FORTIS: A Comprehensive Solution for Establishing Forward Trust for Protecting IPs and ICs," *ACM Transactions on Design Automation of Electronic Systems*, vol. 21, no. 4, p. 63, 2016.

[98] G. Gimenez, A. Cherkaoui, G. Cogniard, and L. Fesquet, "Static timing analysis of asynchronous bundled-data circuits," in *International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2018, pp. 110–118.

[99] N. Limaye *et al.*, "Thwarting All Logic Locking Attacks: Dishonest Oracle with Truly Random Logic Locking," *IEEE TCAD*, 2020.

[100] M. K. Ganai *et al.*, "Efficient BMC for Multi-clock Systems with Clocked Specifications," in *ASP-DAC*, 2007, pp. 310–315.

[101] J. Guajardo, S. S. Kumar, G.-J. Schrijen , and P. Tuyls, "Physical Unclonable Functions and Public-Key Crypto for FPGA IP Protection," in *Int'l Conference on Field Programmable Logic and Applications (FPL)*, 2007, pp. 189–195.

[102] 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 *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2019, pp. 181–195.

[103] D. S. Green, "Leveraging the commercial sector and providing differentiation through functional disaggregation," 2013, https://www.darpa.mil/attachments/DisaggregatetheCircuit_Slides.pdf.

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

[105] J. B. Wendt and M. Potkonjak, "Hardware Obfuscation using PUF-based Logic," in *IEEE/ACM Int'l Conference on Computer-Aided Design (ICCAD)*, 2014, pp. 270–271.

[106] F. Koushanfar, "Provably Secure Active IC Metering Techniques for Piracy Avoidance and Digital Rights Management," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 1, pp. 51–63, 2012.

[107] G. Contreras, Md. T. Rahman, and M. Tehranipoor, "Secure Split-Test for Preventing IC Piracy by Untrusted Foundry and Assembly," in *IEEE Int'l Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, 2013, pp. 196–203.

[108] E. Barker and J. Kelsey, "Recommendation for the Entropy Sources used for Random Bit Generation," *Draft NIST Special Publication*, pp. 800–900, 2012.

[109] H. M. Kamali, K. Z. Azar, S. Roshanisefat, A. Vakil, and A. Sasan, "Extru: A lightweight, fast, and secure expirable trust for the internet of things," *arXiv preprint arXiv:2004.06235*, 2020.

[110] B. J. Gilbert Goodwill, J. Jaffe, and P. Rohatgi, "A Testing Methodology for Side-Channel Resistance Validation," in *NIST Non-Invasive Attack Testing Workshop*, vol. 7, 2011, pp. 115–136.

[111] H. Ahmadi and W. E. Denzel, "A Survey of Modern High-Performance Switching Techniques," *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 7, pp. 1091–1103, 1989.

[112] 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.

[113] M. J. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," Tech. Rep., 2007, nIST Technical Report.

[114] H. Wu, "ACORN: A Lightweight Authenticated Cipher (v3)," *Candidate for the CAESAR Competition*, 2016, https://competitions.cr.yp.to/round3/acornv3.pdf.

[115] W. Diehl, F. Farahmand, A. Abdulgadir, J.-P. Kaps, and K. Gaj, "Face-Off between the CAESAR Lightweight Finalists: ACORN vs. Ascon," in *Int'l Conference on Field Programmable Technology (ICFPT)*, 2018.

[116] E. Homsirikamol, W. Diehl, A. Ferozpuri, F. Farahmand, P. Yalla, J.-P. Kaps, and K. Gaj, "CAESAR Hardware API," *Cryptology ePrint Archive, Report 2016/626*, p. 669, 2016.

[117] S. Nikova, C. Rechberger, and V. Rijmen, "Threshold Implementations against Side-Channel Attacks and Glitches," in *Int'l Conference on Information and Communications Security*, 2006, pp. 529–545.

[118] O. Petura, U. Mureddu, N. Bochard, V. Fischer, and L. Bossuet, "A Survey of AIS-20/31 Compliant TRNG Cores Suitable for FPGA Devices," in *Int'l Conference on Field Programmable Logic and Applications (FPL)*, 2016, pp. 1–10.

[119] C. De Canniere and P. Bart, "Trivium Specifications," in *eSTREAM, ECRYPT Stream Cipher Project*, 2005.

[120] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan Detection using IC Fingerprinting," in *IEEE Symposium on Security and Privacy (SP)*, 2007, pp. 296–310.

[121] J. Delvaux and I. Verbauwhede, "Attacking PUF-based Pattern Matching Key Generators via Helper Data Manipulation," in *Cryptographers' Track at the RSA Conference*, 2014, pp. 106–131.

[122] W. Diehl, A. Abdulgadir, F. Farahmand, J.-P. Kaps, and K. Gaj, "Comparison of Cost of Protection against Differential Power Analysis of Selected Authenticated Ciphers," *Cryptography*, vol. 2, no. 3, p. 26, 2018.

[123] A. Vakil, H. Homayoun, and A. Sasan, "IR-ATA: IR annotated timing analysis, a flow for closing the loop between PDN design, IR analysis & timing closure," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2019, pp. 152–159.

[124] A. Vakil, F. Behnia, A. Mirzaeian, H. Homayoun, N. Karimi, and A. Sasan, "Lasca: Learning assisted side channel delay analysis for hardware trojan detection," *arXiv preprint arXiv:2001.06476*, 2020.

[125] A. Vakil, F. Niknia, A. Mirzaeian, A. Sasan, and N. Karimi, "Learning assisted side channel delay test for detection of recycled ics," *arXiv preprint arXiv:2010.12704*, 2020.

[126] L. Bhamidipati, B. Gunna, H. Homayoun, and A. Sasan, "A Power Delivery Network and Cell Placement Aware IR-Drop Mitigation Technique: Harvesting Unused Timing Slacks to Schedule Useful Skews," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2017, pp. 272–277.

[127] B. Gunna, L. Bhamidipati, H. Homayoun, and A. Sasan, "Spatial and Temporal Scheduling of Clock Arrival Times for IR Hot-Spot Mitigation, Eeformulation of Peak Current Reduction," in *IEEE/ACM Int'l Symposium on Low Power Electronics and Design (ISLPED)*, 2017, pp. 1–6.

[128] T. Machida, D. Yamamoto, M. Iwamoto, and K. Sakiyama, "Implementation of Double Arbiter PUF and its Performance Evaluation on FPGA," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2015, pp. 6–7.

Biography

Kimia Zamiri Azar has been a Ph.D. student in Department of Electrical & Computer Engineering at George Mason University, Fairfax, VA from 2017 to 2021. She was a member of the Green, Accelerated, and Trustworthy Engineering (GATE) Lab, working under the supervision of Dr. Avesta Sasan. Kimia's research interest mainly lie in the areas of hardware security and trust, security for supply chain, and VLSI design and test. She received her Master Degree in Computer Engineering from the Shahid Beheshti University, Tehran, Iran in 2015. Furthermore, she obtained her Bachelor Degree in Computer Engineering from K. N. Toosi University of Technology, Tehran, Iran in 2013.