ABSTRACT

Title of dissertation: CIRCUIT DESIGN OBFUSCATION FOR HARDWARE SECURITY

Yang Xie
Doctor of Philosophy, 2018

Dissertation directed by: Professor Ankur Srivastava
Department of Electrical and
Computer Engineering

Nowadays, chip design and chip fabrication are normally conducted separately by independent companies. Most integrated circuit (IC) design companies are now adopting a fab-less model: they outsource the chip fabrication to offshore foundries while concentrating their effort and resource on the chip design. Although it is cost-effective, the outsourced design faces various security threats since the offshore foundries might not be trustworthy. Attacks on the outsourced IC design can take on many forms, such as piracy, counterfeiting, overproduction and malicious modification, which are referred to as IC supply chain attacks. In this work, we investigate several circuit design obfuscation techniques to prevent the IC supply chain attacks by untrusted foundries.

Logic locking is a gate-level design obfuscation technique that's proposed to protect the outsourced IC designs from piracy and counterfeiting by untrusted foundries. A locked IC preserves the correct functionality only when a correct key is provided. Recently, the security of logic locking is threatened by a strong

attack called SAT attack, which can decipher the correct key of most logic locking techniques within a few hours even for a reasonably large key-size. In this dissertation, we investigate design techniques to improve the security of logic locking in three directions. Firstly, we propose a new locking technique called Anti-SAT to thwart the SAT attack. The Anti-SAT can make the complexity of SAT attack grow exponentially in key-size, hence making the attack computationally infeasible. Secondly, we consider an approximate version of SAT attack and investigate its application on fault-tolerant hardware such as neural network chips. Countermeasure to this approximate SAT attack is proposed and validated with rigorous proof and experiments. Lastly, we explore new opportunities in obfuscating the parametric characteristics of a circuit design (*e.g.,* timing) so that another layer of defense can be added to existing countermeasures.

Split fabrication based on 3D integration technology is another approach to obfuscate the outsourced IC designs. 3D integration is a technology that integrates multiple 2D dies to create a single high-performance chip, referred to as 3D IC. With 3D integration, a designer can choose a portion of IC design at his discretion and send them to a trusted foundry for secure fabrication while outsourcing the rest to untrusted foundries for advanced fabrication technology. In this dissertation, we propose a security-aware physical design flow for interposer-based 3D IC (also known as 2.5D IC). The design flow consists of security-aware partitioning and placement phases, which aim at obfuscating the circuit while preventing potential attacks such as proximity attack. Simulation results show that our proposed design flow is effective for producing secure chip layouts against the IC supply chain attacks.

The circuit design obfuscation techniques presented in this dissertation enable future chip designers to take security into consideration at an early phase while optimizing the chip's performance, power, and reliability.

CIRCUIT DESIGN OBFUSCATION FOR HARDWARE SECURITY

by

Yang Xie

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2018

Advisory Committee:
Professor Ankur Srivastava, Chair/Advisor
Professor Dana Dachman-Soled
Professor Manoj Franklin
Professor Gang Qu
Professor Jeffrey S. Foster

Dedication

To my parents, my sister, and my wife, for their love and support.

Acknowledgments

First and foremost, I would like to thank my advisor, Professor Ankur Srivastava, for his continuous guidance and support through my Ph.D. study. I'm extremely grateful for his patience, motivation and immense knowledge in all the time we spent on solving challenging research problems. He has always guided me to the correct path and supported my decision, which makes my Ph.D. experience productive and rewarding. The enthusiasm he has for his research will always be a great motivation for me. It's truly a pleasure to work with and learn from him.

I would also like to express my gratitude to Professor Dana Dachman-Soled, Professor Manoj Franklin, Professor Gang Qu, and Professor Jeffrey S. Foster for their time to serve on this committee and their valuable feedback on this dissertation.

I would like to extend my thanks to all my colleagues in Professor Srivastava's research group. My thanks first go to three senior colleagues Chongxi Bao, Tiantao Lu and Caleb Serafy for showing me how to conduct research, set up experiments and write papers in the early stage of my Ph.D. study. Without them, I would not be able to adapt to the Ph.D. life so quickly. Thanks also go to my current colleagues Zhiyuan Yang, Yuntao Liu, Ankit Mondal, Abhishek Chakraborty and Mike Zuzak for the inspiring research discussions, for the late nights we were working together before deadlines, and for all the fun we have had in our lab.

Finally, I owe the deepest gratitude to my family. I want to express my appreciation to my parents and my sister for their endless love and devotion and to my wife Miao Zhang who has been accompanying me during every hardship.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ATPG	Automatic Test Pattern Generation
BDD	Binary Decision Diagram
BEOL	Back End of Line
CNN	Convolutional Neural Network
DIO	Distinguishing input/output
DLL	Delay+Logic Locking
DPA	Differential Power Analysis
EDA	Electronic Design Automation
EM	Electromagnetic
FEOL	Front End of Line
FF	Flip-Flops
FIB	Focused Ion Beam
FSM	Finite State Machine
HD	Hamming Distance
HT	Hardware Trojan
HSM	Hardware Security Module
IC	Integrated Circuit
IP	Intellectual Property
LB	Lower Bound
LUT	Look Up Table
MILP	Mixed Integer Linear Programing
MLP	Multi-Layer Perceptron
MUX	Multiplexer
PCB	Printed Circuit Board
PE	Processing Element
PO	Primary Outputs
PUF	Physically Unclonable Function
RTL	Register Transfer Level

SA	Simulated Annealing
SAT	Satisfiability
SCA	Side Channel Attack
SLL	Secure Logic Locking
SoC	System on Chip
SPA	Simple Power Analysis
SPS	Signal Probability Skew
TDB	Tunable Delay Buffer
TDK	Tunable Delay Key-gate
TRNG	True Random Number Generator
TSV	Through-Silicon-Vias
TVR	Timing Violation Ratio
UB	Upper Bound

Chapter 1: Introduction

Traditionally, cyber-security studies focus mainly on software and information security, which aims at protecting the confidentiality and integrity of data that is computed and transmitted among multiple parties. Software-oriented security is developed in forms such as password, encryption and decryption, digital signature, anti-virus software, crypto-currency, *etc.* When developing these software-based security applications, the underlying hardware systems (*i.e.,* electronic systems, chips, ICs) that provide the computation and communication of the software are normally assumed to be secure and reliable. However, such security assumptions about hardware cannot be easily guaranteed, because hardware systems could also have vulnerabilities and could be attacked. In recent years, people have discovered more and more hardware-based security threats. Hardware security threats could be due to unintended human mistakes or oversight during design time. For example, recent attacks Meltdown [1] and Spectre [2] exploit critical vulnerabilities in modern processors to develop an effective attack scheme which allows attackers to steal data that is processed on the hardware. A software patch to fix this hardware flaw could take up to 25% performance overhead [3]. Besides unintended design flaws, hardware security threats could also be due to malicious tampering that is performed during

chip design and fabrication. Recent reports [4–6] have discovered malicious backdoors (also known as hardware Trojans) that are inserted into IC designs which can stealthily make the hardware system malfunction. In addition to hardware tampering, hardware design piracy and counterfeiting have become a significant issue in modern IC supply chain. According to a report by IHS Technology, the potential annual financial risks for the global electronics supply chain due to counterfeiting was estimated to be over $169 billion in 2011 [7]. These hardware attacks pose significant security threats to both consumer electronics and mission-critical systems.

1.1 Taxonomy of Hardware Attacks

Hardware attacks can take many forms and they can happen during different stages of an IC's life cycle. To begin with, attacks on hardware can happen during its design and fabrication, which are referred to as *supply chain attacks*. Modern chips go through a complicated supply chain of design, fabrication, packaging, and assembly, as shown in Fig. 1.1. Each stage of the IC supply chain involves many global suppliers which could be possibly untrustworthy. Hardware designs might be tampered or pirated by untrusted parties in the IC supply chain, resulting in a huge economic loss to most IC design companies. Furthermore, attacks can also happen after the hardware is deployed to end users, which are referred to as *post-deployment attacks*. For example, malicious end users might want to reverse-engineer the chips in order to obtain its implementation details. Besides, they might want to identify hardware vulnerabilities and use them to access valuable data that is processed on

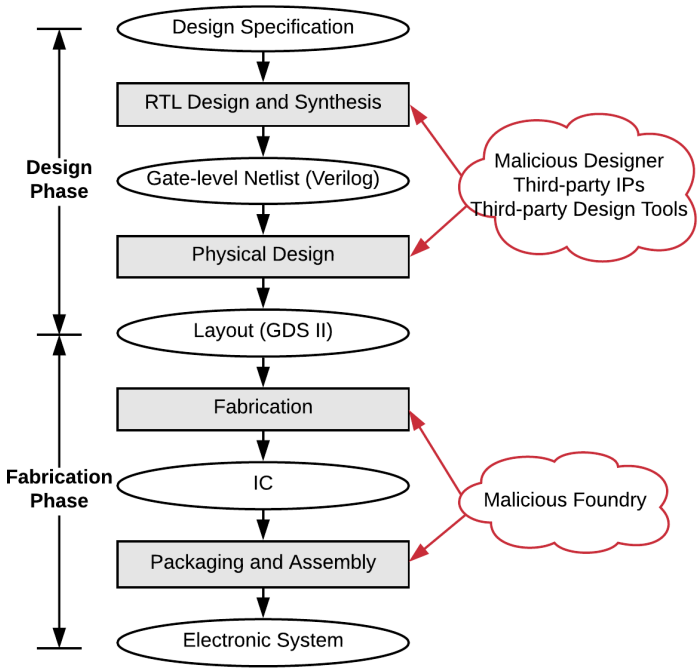


Figure 1.1: IC supply chain and its security threats.

the chips. In this section, we provide a detailed taxonomy of hardware attacks, including IC supply chain attacks and post-deployment attacks.

1.1.1 IC Supply Chain Attacks

Fig. 1.1 shows an overview of modern IC supply chain. In general, it can be divided into two phases:

- *Design Phase.* In the design phase, an abstract design specification is first described using register-transfer level (RTL) language such as Verilog and then synthesized into a gate-level netlist using commercial design and synthesis tools. After that, physical design tools are used to produce a layout for the

netlist. The layout file describes the physical shapes, locations, and routing of different gates in the netlist, which is used as a reference for chip fabrication. Nowadays, it's very common for designers to integrate hardware designs that are purchased from other companies, which are known as third-party Intellectual Property (IP).

- *Fabrication Phase.* In the fabrication phase, the layout is used to generate masks for chip fabrication. After that, the chips are packaged and assembled into a printed circuit board (PCB) to produce the final electronic system. To access advanced semiconductor technology at a low cost, most IC designs are now outsourced to an off-shore foundry for chip fabrication. Once the layouts are sent to the foundries, they are not under the direct control or monitoring of the designer.

Different attacks can happen in different phases of the IC supply chain. The following summarizes potential attacks in each phase.

1.1.1.1 Attacks in Design Phase

Security threats in the design phase come from three sources, malicious designer, untrusted third-party IPs, and untrusted third-party design tools. First of all, a rogue employee in the design team can steal or modify the chip design. This is the most dangerous threat in the design phase because the rogue employee has full access to the design. Besides the malicious insiders, third-party IPs which are provided by untrusted IP vendors can also secretly sabotage the hardware design.

Although integrating third-party IPs can expedite the design process, it allows attackers (the IP vendors) to insert malicious backdoor that could make the chip malfunction under certain circumstances. Lastly, third-party design tools also pose potential threats to hardware design. Often times, designers simply rely on the automatic design optimization and verification processes offered by the tools. However, the resulting design might be tampered or undermined during these automatic design processes.

To summarize, two attacks can happen during the design phase:

- *IP piracy.* The objective of IP piracy attack is to steal the hardware design, illegally claim the ownership and use them in an unauthorized way. IP piracy can be conducted by a rogue employee in the design team. The employee can steal the RTL or gate-level designs and sell them to other design companies to gain profits.

- *Hardware Trojan (HT).* HTs refer to the malicious modifications or backdoors that can 1) change or nullify some functionalities [8] and 2) undermine the IC's performance and reliability [9]. HTs can be inserted during the design phase by rogue employees in the design team, untrusted third-party IP vendors or untrusted third-party design tools.

1.1.1.2 Attacks in Fabrication Phase

Security threats in the fabrication phase mainly come from the malicious insider in the fabrication foundries. To access advanced semiconductor technology

at a lower cost, most IC design companies are now outsourcing their IC designs to an offshore foundry for fabrication. However, the offshore foundry might not be trustworthy. Without direct control and monitoring from the design company, a malicious insider in the foundry can pirate or tamper the IC designs. The following summarizes potential attacks that can happen during the outsourced fabrication phase.

- *IP Piracy.* IP piracy can be conducted by a malicious foundry. Since the foundry has access to the layout of the hardware design, it can analyze the layout and use state-of-the-art IC reverse engineering techniques [10] to gain knowledge of the design at different levels (*e.g.,* RTL or gate-level). Later, the malicious foundry can benefit from selling the extracted design knowledge and details to other design companies

- *IC Overproduction.* This attack is conducted by malicious fabrication foundries which overproduce and sell extra copies of chips for profit. To reduce cost, these overbuilt ICs might not be subject to a complete testing process. As a result, some unauthorized and low-quality ICs may end up being packaged and sold to the market, which renders both economic and reputation loss to the design company.

- *Counterfeiting.* This attack is related to the production and distribution of out-of-spec, fake, or recycled chips [11]. Out-of-spec ICs (which fail some quality tests) are normally supplied by malicious foundries. During testing, out-of-spec chips can be withheld by the foundry and marked as qualified

chips. Later, these chips can be sold to the market, without the designer being aware of it. Fake and recycled chips are other forms of counterfeited chips. For example, an old-generation chip can be relabeled into a new-generation one. These counterfeited chips could be integrated into a hardware system by attackers during chip packaging and assembly.

- *Hardware Trojan (HT).* HTs can be inserted during the fabrication phase by a malicious foundry. The attacker can modify the layout to change the chip's functionality or undermine its performance and reliability.

1.1.2 Post-deployment Attacks

After the hardware are deployed to the users, various attacks can be performed to tamper the hardware or the data that's processed on it. Following describes two common hardware-based attacks in the post-deployment phase.

- *Reverse Engineering.* In this attack, a malicious user wants to obtain implementation details of the hardware. He can use state-of-the-art reverse engineering technique [10] to extract valuable knowledge which might be used for further attacks such as producing counterfeited ICs. Compared to layout-level reverse engineering by malicious foundries (as discussed in Section 1.1.1.2), chip-level reverse engineering by end users is harder because it requires more steps such as chip decapsulation and delayering.

- *Side-channel attack (SCA).* In this attack, a malicious user exploits physical characteristics of a hardware system (*e.g.,* power [12, 13], run time [14, 15],

electromagnetic (EM) emission [16–18] *etc.*) to learn the secret data that is processed on the hardware. The aforementioned Meltdown [1] and Spectre [2] attacks are examples of SCAs.

The taxonomy of hardware-based attacks is illustrated in Fig. 1.2. These emerging hardware security threats motivate researchers to develop effective design techniques to enhance the security of modern chips.

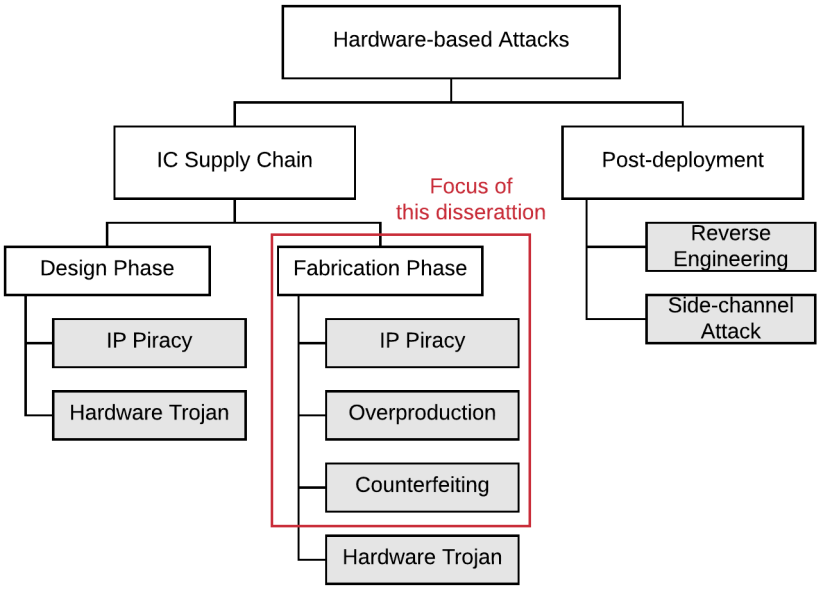


Figure 1.2: Taxonomy of hardware-based attack.

1.2 Contributions and Thesis Organization

1.2.1 Focus of This Dissertation

In this dissertation, we concentrate on the *IC supply chain security* for modern chips with a focus on security threats in the *fabrication phase*. The main theme

of this work is to develop novel design techniques to enhance the security of outsourced IC designs that are fabricated in possibly *untrustworthy foundries*. We investigate new attack strategies and propose security-aware design techniques that aim at thwarting the supply chain attacks including IP piracy, overproduction and counterfeiting by untrusted foundries. These design techniques can be utilized to enhance trust between design companies and fabrication foundries so that a mutually beneficial cooperation can be maintained.

1.2.2 Contributions

One set of techniques to protect the outsourced IC designs is to obfuscate the functionality and implementation detail of the outsourced circuit design so as to confuse the untrusted foundry. Without knowing the correct functionality or implementation, an attacker cannot obtain the original circuit design or overproduce any usable chips. Various circuit obfuscation techniques have been proposed, including IC metering [19], logic locking [20] and split fabrication [21]. These circuit obfuscation techniques, however, have been shown to have vulnerabilities and could be attacked in recent literature [22–30]. De-obfuscation attacks have been proposed to de-obfuscate the circuit and recover the original design. The vulnerabilities in existing circuit obfuscation techniques motivate this work to develop new and more secure countermeasures.

The first contribution of this work is to enhance the security of existing circuit obfuscation techniques (*e.g.,* logic locking) so that de-obfuscation attacks would be

computationally infeasible. We have investigated various attacks on logic locking and developed countermeasures that can thwart such attacks in a provably-secure manner. The second contribution of this work is to investigate the security of existing logic locking techniques when applied to emerging hardware design such as neural network chips. We proposed new attack and defense strategies to enhance the security of locking neural chips. The third contribution of this work is to develop completely new design techniques that exploit new obfuscation possibilities in hardware design. For example, conventional circuit obfuscation techniques aim at obfuscating the digital aspects (*e.g.,* the Boolean functionality) of a circuit while its counterpart, the analog aspects (*e.g.,* timing, power, *etc.*) are not fully exploited. We explore new opportunities in obfuscating the parametric characteristics so that another layer of defense can be added to existing countermeasures. In addition, we explore emerging fabrication technologies (*e.g.,* 3D integration) for obfuscation, which opens new opportunities in protecting the outsourced IC design. Overall, we develop the following design obfuscation techniques to enhance chip security against untrusted foundries.

- **Anti-SAT: Secure Logic Locking Against SAT Attack.** Logic locking is a circuit obfuscation technique that has been proposed to protect outsourced IC designs from piracy and counterfeiting by untrusted foundries. A locked IC preserves the correct functionality only when a correct key is provided. Recently, the security of logic locking is threatened by a new satisfiability (SAT) checking based attack, denoted as SAT attack [24]. The SAT attack

can decipher the correct key of most logic locking techniques within a few hours even for a reasonably large number of keys. In this work, we present a circuit block (referred to as Anti-SAT block) to thwart the SAT attack. We show that the number of SAT attack iterations required to reveal the correct key in a circuit comprising an Anti-SAT block is an exponential function of the key-size thereby making the SAT attack computationally infeasible.

- **Strong Anti-SAT: Secure Logic Locking for Neural Network Chips.** In this work, we investigate the security of logic locking when it's applied to a neural network chip. Locking neural chips is not the same as locking conventional chips in two aspects. Firstly, most neural network applications are inherently error-tolerant. The classification accuracy of a neural network would be acceptable even when some of its underlying computations are incorrect. This can be exploited by an attacker who can just find an approximate key instead of a correct key to approximately unlock the chip such that it can output correctly for most inputs. Secondly, most neural network models are tune-able (*e.g.,* by fine-tuning the weight values). An attacker can adjust his own neural models to accommodate the approximately-unlocked neural chips, hence further improving the classification accuracy. To address these new challenges, we propose a novel locking technique called Strong Anti-SAT to protect the neural chips against untrusted foundries.

- **Delay Locking: Security Enhancement of Logic Locking Against Overproduction and Counterfeiting.** In this work, we propose a new

obfuscation technique called delay locking. For delay locking, the key to a locked circuit not only determines its functionality but also its timing profile. A functionality-correct but timing-incorrect key will result in timing violations and thus make the circuit malfunction. With delay locking, functionality oriented attacks (*e.g.,* SAT attack) are thwarted because they cannot be utilized to decipher a timing-correct key.

- **Security-aware Design Flow for 2.5D IC Split Fabrication.** 3D integration is a technology that integrates multiple 2D dies to create a single high-performance chip, referred to as 3D IC. With 3D integration, a designer can choose a portion of layers at his discretion and fabricate them in a trusted foundry while outsourcing the rest to untrusted foundries for advanced fabrication technology. This split fabrication strategy of 3D ICs creates a new opportunity to obfuscate the outsourced designs. In this work, we propose a security-aware physical design flow for interposer-based 3D IC (also known as 2.5D IC) technology to prevent supply chain attacks by untrusted foundries.

1.2.3 Thesis Organization

The rest of this dissertation is organized as follows. In Chapter 2, we introduce the supply chain attacks by untrusted foundries and summarize various circuit obfuscation techniques that have been proposed to prevent IC designs from being pirated, overproduced or counterfeited. In Chapter 3, we discuss a new logic locking technique called Anti-SAT and validate its security against SAT attack. These

results have been published in [31,32]. In Chapter 4, we first investigate new attack strategies for a neural chip that's locked with Anti-SAT. Based on the attack results, we discuss a new defense called Strong Anti-SAT to enhance the security of logic locking for neural chips. In Chapter 5, we describe a new obfuscation technique called delay locking that obfuscates the timing profile of an IC design. The results have been published in [33]. In Chapter 6, we introduce a security-aware design flow for 2.5D IC split fabrication. Part of the results were published in [34–38]. Finally, Chapter 7 concludes this dissertation and discusses future work.

Chapter 2: Background

In this chapter, we give an overview of supply chain attacks in fabrication phase with a focus on hardware IP threats. We will describe the attack model, attack schemes and summarize various types of attack mitigation techniques.

2.1 Supply Chain Attacks for IP Piracy

As introduced in Section 1.1.1, IC designs are increasingly outsourced to an offshore foundry for fabrication. Since the foundry might not be trustworthy, the outsourced IC design might be pirated, overproduced, counterfeited or maliciously modified. In this dissertation, we focus on hardware IP threats. We assume that the attacker is an untrusted foundry and its primary objective is to steal the hardware IP or overproduce/counterfeit the chips to make profits. However, as design techniques such as circuit obfuscations (which will be discussed in Section 2.2) have been proposed to thwart the supply chain attacks, the attacker might have to circumvent these countermeasures before performing further attacks. Following sections will formally describe the attack model and attack schemes.

2.1.1 Attack Model

We assume that the attacker is an *untrusted foundry* which fabricates chips for an IC design company (designer). The attacker's objective is to 1) learn about the design implementation and steal the IC design and 2) overproduce or counterfeit the IC design. The untrusted foundry has access to two components:

1. *Layout files of an IC design, provided by the IC design company.* These outsourced layout files are assumed to be correctly and securely designed using trusted Electronic Design Automation (EDA) tools for logic synthesis and physical design. We assume that the attacker has the ability to reverse-engineer the layouts to obtain their gate-level netlists using state-of-the-art IC reverse-engineering techniques [10]. Such reverse-engineering capability enables the attacker to analyze and pirate the IC design.

2. *A functional chip, obtained from the open market.* This chip will function correctly as a black-box, *i.e.,* it will produce correct outputs but the implementation detail of the chip is not known. This component gives the attacker the ability to query correct input/output responses from the functional chip. As will be discussed later, such ability can help the attacker to circumvent certain defense techniques.

The aforementioned attack model has been widely used in most works on hardware IP protection against malicious foundries [22–24, 26, 36, 39–44]. In some research works, only the first component is assumed to be accessible to attackers [27–29].

2.1.2 Attack Schemes

We consider two attack schemes based on whether an outsourced design is obfuscated or not.

- *Design Without Obfuscation.* If the outsourced design is not obfuscated, the foundry can directly perform the supply chain attacks such as IP piracy, overproduction, and counterfeiting, as described in Section 1.1.1.2. It can reverse-engineer the layout and pirate the IP, overproduce extra unauthorized chip copies, and produce counterfeit products that utilize out-of-spec, fake or recycled chips.

- *Design With Obfuscation.* If the outsourced design is obfuscated, the foundry has to first de-obfuscate the circuit before performing the supply chain attacks. As described in Section 2.1.1, the attacker has access to two components which can be exploited for circuit de-obfuscation. The first component is the obfuscated layout files provided by the designer, which is accessible during the fabrication phase. These layout files can be reverse-engineered to gate-level netlists which can provide more information about the obfuscated design. The second component is a functional chip, which is accessible only after the designer has deployed the chips to the open market. This functional chip can be used to query correct input/output patterns which could assist the attacker in de-obfuscating the circuit.

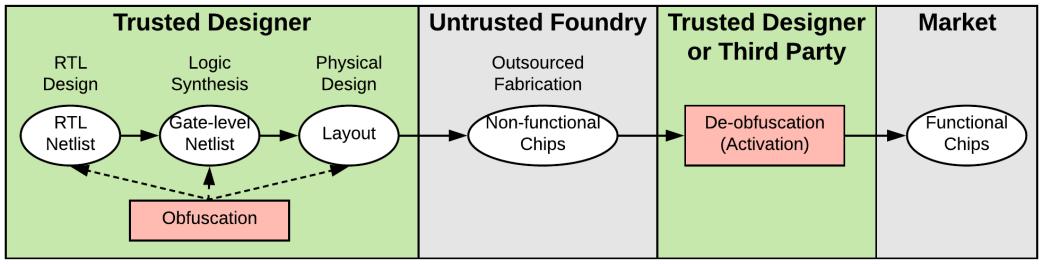


Figure 2.1: An IC design and fabrication flow enhanced with circuit obfuscation.

2.2 Circuit Obfuscation

The security threats in outsourced chip fabrication have heightened the need for effective countermeasures. One set of techniques to prevent the supply chain attacks is to obfuscate the functionality and implementation detail of the outsourced circuit design so as to confuse the untrusted foundry. In this section, we will first summarize various circuit obfuscation techniques. Then, we will introduce two obfuscation techniques, namely logic locking and split fabrication, in more detail.

2.2.1 Overview of Circuit Obfuscation

Fig. 2.1 shows an IC design and fabrication flow that's enhanced with circuit obfuscation. During design time, a circuit design can be obfuscated at different design stages to hide its functionality and implementation. After that, the layout of the obfuscated design is outsourced to an untrusted foundry for fabrication. Without knowing the correct functionality or implementation, an attacker cannot reverse-engineer the original circuit design or overproduce any functional chips. Af-

ter fabrication, the non-functional chips will be sent to the designer or trusted third-party, which can de-obfuscate these chips to regain the functional chips and sell them to market.

Circuit obfuscation can be implemented at different levels, namely finite-state-machine (FSM) level, gate level, and layout level.

- *FSM Level.* FSM is a representation of the Boolean function of sequential circuits. An FSM normally consists of inputs, outputs, states, and transitions. Active metering [19] is an FSM-level obfuscation technique. In this technique, an FSM is obfuscated by adding extra non-functional states and transitions. To enter a valid functional state, one would need to provide a sequence of inputs (called pass-key), which converts an initial non-functional state into a valid functional state. Obfuscated FSMs can be integrated with other RTL design to obfuscate its control and data flow, as proposed in [45, 46].

- *Gate Level.* Obfuscation can also happen at the gate level after the Boolean logic is synthesized into a gate-level netlist. Given a gate-level netlist, a designer can obfuscate its combinational portion by modifying the original design such that the functionality of obfuscated netlist are different from the original one. The key aspect is that these netlist modifications should be recoverable by the designer or other trustworthy parties after the chips are fabricated. Logic locking [1] [20] is a gate-level obfuscation technique that obfuscates a netlist by inserting a set of key-controlled logic gates (key-gates) and key-inputs. The

[1] Logic locking is also known as logic encryption [47] and logic obfuscation [22].

locked netlist preserves the original functionality only when a correct key is provided. On the other hand, a wrong key can cause faults inside the netlist and make it output incorrectly. Chip Editor [48] is another gate-level circuit obfuscation technique. During design time, a designer can obfuscate the gate-level netlist by adding extra gates and wires. After receiving the chips fabricated in untrusted foundries, the designer or trusted third parties can use focused ion beam (FIB) based chip edit technology to directly modify the chips to nullify the extra gates or wires, thereby making the chips function correctly [48].

- *Layout Level.* A circuit layout is typically composed of multiple metal layers and device layers. At the layout level, a circuit can be obfuscated by splitting its layout into two tiers: a trusted tier and an untrusted tier. The untrusted tier consists of layers which require advanced semiconductor technology, so it is outsourced to an untrusted foundry. On the other hand, the trusted tier which contains less advanced layers can be fabricated in a trusted foundry for security. The final integration of two tiers is also done securely in the trusted foundry. This security-oriented fabrication strategy is called split fabrication [21]. By having two tiers fabricated separately, the untrusted foundry could not have access to the trusted tier and thus cannot have a complete view of the IC design.

In the remaining of this section, we will discuss two obfuscation techniques logic locking and split fabrication in more detail.

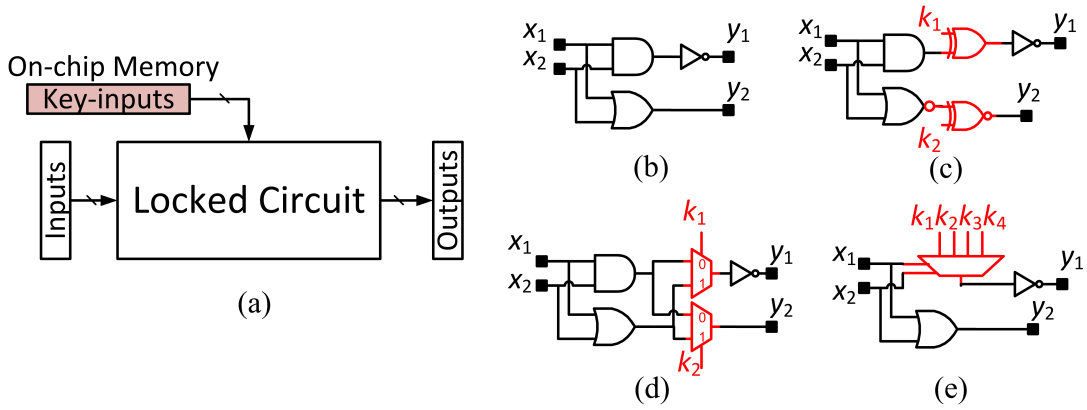


Figure 2.2: Logic locking techniques: (a) overiew; (b) an original netlist; (c) XOR/XNOR based; (d) MUX based; (e) LUT based.

2.2.2 Logic Locking

2.2.2.1 Basic Idea

Logic locking [20] is a gate-level obfuscation technique. It obfuscates the combinational part of a circuit design by inserting a set of key-controlled logic gates (key-gates) and key-inputs. Fig. 2.2 shows the overview of logic locking techniques. A circuit is locked by inserting a set of key-gates and key-inputs. The key-inputs are connected to an on-chip memory and the locked IC preserves the correct functionality only when a correct key is set to the on-chip memory. After inserting the key-gates, the obfuscated netlist will be re-synthesized so that the key-gates can blend into the original netlist, which prevents them from being identified and removed. To prevent the untrusted foundry from probing internal signals of a running chip, a tamper-proof chip protection shall be implemented.

Recent years have seen various logic locking techniques based on different key-

gate types and key-gate insertion algorithms. According to the key-gate types, they can be classified into three major categories: XOR/XNOR based logic locking [20, 22, 47], multiplexer (MUX) based logic locking [23, 39, 47, 49] and Look-Up-Table (LUT) based logic locking [50–52], as shown in Fig. 2.2(c-e). Besides these three major types of key-gates, previous works have also investigated the use of AND/OR gates [53] and other special logic blocks [45] as the key-gates. Among all, the XOR/XNOR based logic locking has received the most attention mainly due to its simple structure and low performance overhead.

Different key-gate insertion algorithms have also been proposed [20, 22, 23, 39, 47, 53], which aim at identifying the optimal locations for key-gates to improve the security of the logic locking technique. The simplest algorithm is the random insertion [20], where key-gates are distributed randomly to different locations of the netlist. To increase fault impact, a fault-analysis based insertion algorithm [47] proposed to insert the key-gates at locations that can maximally corrupt the primary outputs (POs) when an incorrect key is given. This algorithm ensures that the functionality of the locked circuit (with an incorrect key) would deviate substantially from the original one. Following these two prior works [20, 47], researchers have started to develop key-learning attacks and proposed new logic locking algorithms to thwart such attacks [22, 23, 39, 53].

2.2.2.2 Attacks on Logic Locking

The attacks on logic locking assume that the attacker has access to two components: 1) a locked netlist which can be obtained by reverse-engineering the layout files provided by the designer and 2) an unlocked functional chip obtained from open market, as mentioned earlier in Section 2.1.1. The attack objective is to obtain the correct key by utilizing these two components.

- *Brute-force Attack [39].* A circuit netlist normally has multiple outputs. The functionality of each output is determined by a slice of sub-netlist that's referred to as *logic cone*. In this attack, an attacker firstly segments a locked netlist into multiple logic cones, each containing a subset of keys. After that, the attacker targets the logic cones one by one, from the simplest (the one has the least keys) to the most complicated (the one has the most keys). For each logic cone, a brute-force search attack is used to find a correct key. To thwart this attack, a logic-cone analysis based insertion algorithm was proposed to increase the number of key-gates per logic cone using MUX based key-gates.

- *Hill-climbing Search Attack [23].* This attack is based on the assumption that the output correctness of a locked circuit is proportional to the correctness of a key. It uses a hill-climbing based searching algorithm that toggles a guessed key bit-by-bit to search a direction that can improve output correctness (for a set of test inputs). The attack terminates when the guessed key can make a circuit output correctly for the test set. To thwart this attack, a MUX-based

gradient-obfuscated locking method was proposed which aims at reducing the sensitivity of circuit outputs to the key toggle.

- *Key-sensitizing Attack [22].* This attack uses automatic test pattern generation (ATPG) tool to find a set of input patterns that can sensitize the correct key values to POs. The attacker first analyzes the locked netlist to find specific input pattern that can create a path for keys to be sensitized to POs. Then, he can apply that input on an unlocked functional chip and observe the correct key. To mitigate this attack, an interference-analysis based insertion algorithm called Strong Logic Locking (SLL) was proposed which places the key-gates at locations with strong interferences (*i.e.,* not dispersed or isolated). This increases the difficulty of finding an effective sensitizing path from key-inputs to POs.

- *Side-channel Attacks (SCAs) [26, 40].* In [26], a Differential Power Analysis (DPA) based SCA was proposed to find the correct key by statistically correlating the power trace and the key values. To counter this attack, the authors suggest to 1) increase the key-input to primary input ratio in a given logic cone and 2) insert key-gates at selected locations such that incorrect key biases the POs towards a constant value on average. In [40], a finer-grain power SCA based on Template Analysis (TA) was proposed. It monitors power trace samples with respect to different logic depths (determined by arrival time of each gate) from the functional chip. This information is used as a template to guide finding the correct keys. Potential direction to counter the TA is to

equalize the logic depths of all key-gates through specific layout design.

- *SAT-based Attacks [24, 41–44].* Recently, the security of logic locking is threatened by a new attack called SAT attack [24], which can decipher the correct key of most logic locking techniques [20, 22, 47, 50, 53] within a few hours even for a reasonably large key-size. This attack iteratively solves SAT formulas which can progressively eliminate the incorrect keys till the circuit is unlocked. Based on [24], various SAT attack variants have been proposed, including approximate SAT (AppSAT) [41], Double-DIP [42], CycSAT [43] and model checker [44]. These attacks extend the original SAT attack to enhance its effectiveness. Developing secure countermeasures against SAT-based attacks remains an active area of research.

2.2.3 Split Fabrication

2.2.3.1 Basic Idea

Split fabrication [21] is a layout-level circuit obfuscation technique. A layout normally consists of multiple metal layers and device layers. A circuit can be obfuscated by splitting its layout layers into two tiers. One tier is called the *untrusted tier* because it's outsourced to an untrusted offshore foundry for advanced semiconductor technology. The other tier is called the *trusted tier* because it is fabricated in a trusted foundry for security. After fabrication, two tiers are aligned and integrated together in the trusted foundry. By having two tiers fabricated separately, the untrusted foundry could not have access to the trusted tier and thus cannot have a

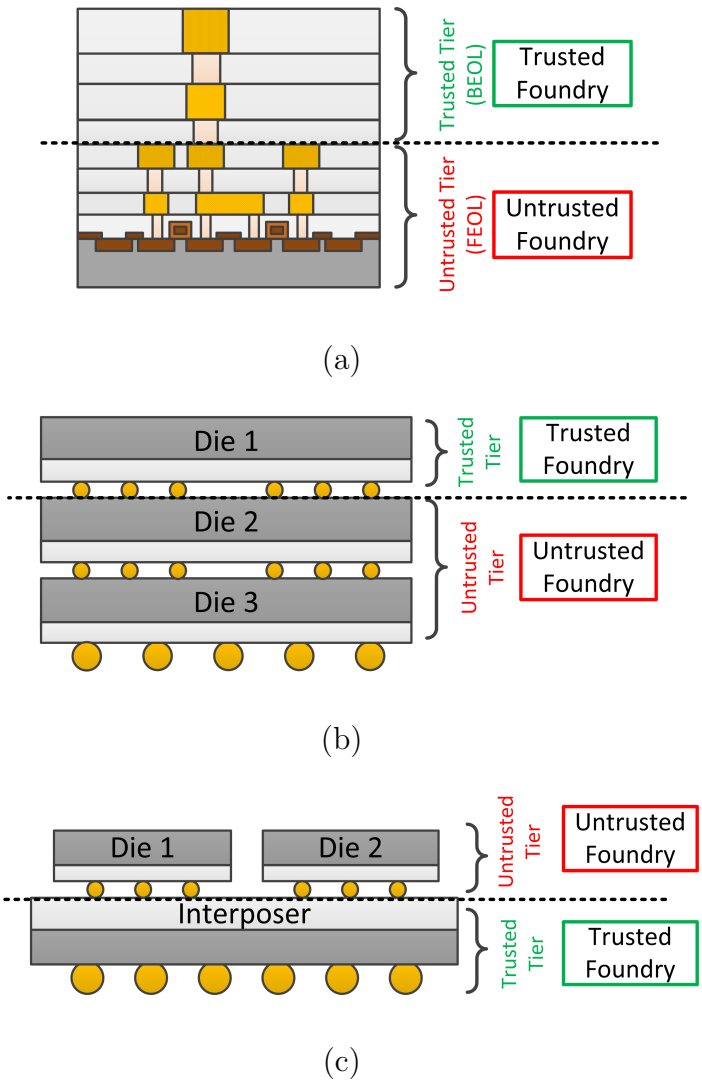


Figure 2.3: Split fabrication: (a) 2D IC; (b) 3D IC; (c) 2.5D IC.

complete view of the design.

The split fabrication strategy can be applied to conventional 2D IC technology and emerging 3D IC technologies.

- *2D IC based Split Fabrication.* Fig. 2.3 (a) shows a cross-section of a 2D IC layout, which consists of multiple metal layers and a device layer. For 2D IC split fabrication [27, 54–57], the layout layers are divided into a Front-End-

of-Line (FEOL) tier that contains the device layer and lower metal layers, and a Back-End-of-Line (BEOL) tier that contains only higher metal layers. The FEOL tier is outsourced to an untrusted foundry for advanced fabrication technology while the fabrication of the BEOL tier and the final integration are securely implemented in a trusted foundry. As a result, interconnect wires in the BEOL tier are kept secret from the untrusted foundry.

- *3D IC based Split Fabrication.* 3D integration is a technology that vertically integrates multiple 2D dies to create a single high-performance chip, referred to as 3D IC (as shown in Fig. 2.3 (b)). Split fabrication for 3D ICs can be done in two approaches [58–60]. In one approach, some 2D dies are fabricated in a trusted foundry as the trusted tier while others are fabricated in an untrusted foundry as the untrusted tier, as shown in Fig. 2.3 (b). With that, a portion of circuit design in the trusted tier are not directly accessible to the untrusted foundry. In another approach, all 2D dies are outsourced to offshore foundries, but they are securely aligned and integrated in a trusted foundry. By doing so, the vertical connections between dies are kept secret. Although the offshore foundry can reverse-engineer the layout of each die, the retrieved netlist is incomplete because it lacks the vertical inter-die connections. Such incomplete netlist would be incomprehensible if a circuit design is intelligently partitioned into different dies in an obfuscated manner.

- *2.5D IC based Split Fabrication.* Interposer-based 3D IC, also known as 2.5D IC, is a special form of 3D IC. As shown in Fig. 2.3(c), 2.5D IC places multiple

2D dies side-by-side and stacks them on a silicon interposer. The interposer contains both horizontal chip-scale interconnect wires between dies as well as vertical interconnect TSVs to external I/O pins. While commercial large-scale 3D IC is still being developed, the 2.5D product is already in the market, such as Xilinx Virtex-7 2000T [61]. For 2.5D IC split fabrication [34–38, 62, 63], the silicon interposer is fabricated in the trusted foundry as the trusted tier while the dies are outsourced to offshore foundries as the untrusted tier. The final integration is also implemented in the trusted foundry. Accordingly, the interconnection in the interposer is hidden from the untrusted foundry.

2.2.3.2 Attacks on Split Fabrication

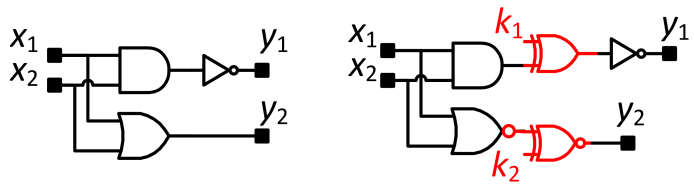
The security of split fabrication rests upon the assumption that the attacker does not know the hidden portion (the trusted tier) and cannot infer it based on the exposed portion of design (the untrusted tier). Otherwise, the attacker can reconstruct the complete design and continue to pirate or overproduce it. Attacks on 2D and 2.5D split fabrication have been proposed in recent literature [27–29, 34]. To infer the hidden wires in the trusted tier, Rajendran *et al.* [27] proposed an attack called *proximity attack*. The attack is based on the observation that modern floorplanning and placement tool will place two connected input/output pins closely so as to reduce the interconnect wire-length. However, the physical proximity of two connected pins leaks the information of the hidden connections. Since the layout information of the untrusted tier is known to the attacker, he can obtain the position

of all input/output pins. With this, he can iteratively connect an output pin to its closet input pin until a netlist is reconstructed. Extensions of the proximity attack have been proposed, which exploit extra layout information such as routing proximity [29], input/output capacitance and timing constraints [28].

Chapter 3: Anti-SAT: Secure Logic Locking Against SAT Attack

3.1 Introduction

In Section 2.2.2, we have introduced a circuit obfuscation technique called logic locking, which can be used to protect outsourced IC designs from being pirated or counterfeited by untrusted foundries. Fig. 3.1 shows a simple example of logic locking. A gate-level netlist is locked by inserting a set of key-gates and key-inputs. The locked netlist preserves the correct functionality only when a correct key is provided to the key-inputs. Recently, the security of logic locking is threatened by a strong attack called SAT attack, which can decipher the correct key of most logic locking techniques within a few hours [24] even for a reasonably large key-size. This attack iteratively solves SAT formulas which progressively eliminate the incorrect keys till the circuit is unlocked. In this chapter, we present a circuit block (referred



(a) (b)

Figure 3.1: Logic locking: (a) original circuit; (b) locked circuit.

to as Anti-SAT block) to enhance the security of existing logic locking techniques against the SAT attack. We show using a mathematical proof that the number of SAT attack iterations to reveal the correct key in a circuit comprising an Anti-SAT block is an exponential function of the key-size thereby making the SAT attack computationally infeasible. Besides, we address the vulnerability of the Anti-SAT block to various removal attacks and investigate obfuscation techniques to prevent these removal attacks. More importantly, we provide a proof showing that these obfuscation techniques for making Anti-SAT un-removable would not weaken the Anti-SAT block's resistance to SAT attack. The contributions of this work are as follows.

- We propose an Anti-SAT circuit block to mitigate the SAT attack on logic locking. We illustrate how to construct the functionality of the Anti-SAT block and use a mathematically rigorous approach to prove that if chosen correctly, the Anti-SAT block makes SAT attack computationally infeasible (exponential in key-size).

- The Anti-SAT block might be subject to attacks that intend to identify and nullify it, which are called removal attacks. We investigate a unified obfuscation technique to hide the functionality and structure of the Anti-SAT block. Also, we provide a proof showing that the obfuscation technique would not weaken the Anti-SAT block's resistance to the SAT attack.

- Rigorous analysis and experiments on 6 circuits from ISCAS85 and MCNC benchmark suites have been conducted to validate the effectiveness of our

proposed technique in improving the security of existing logic locking techniques.

3.2 Preliminary: SAT Attack

Recently, Subramanyan *et al.* [24] proposed a new attack called SAT attack that can effectively break many logic locking techniques including [20, 22, 47, 50, 53].

3.2.1 Attack Model

The SAT attack is based on the same attack model as discussed in Section 2.1.1. It assumes that the attacker is an untrusted foundry whose objective is to obtain the correct key of a locked circuit and then pirate or overprudce the unlocked IC design. The attacker has access to the following two components: 1) a gate-level locked netlist, which can be obtained by reverse-engineering the layout file provided by the designer and 2) an activated functional chip, which can be obtained from an open market. The followings discuss the insight and algorithm of the SAT attack.

3.2.2 Attack Insight

The key idea of the SAT attack is to reveal the correct key using a small number of carefully selected inputs and their correct outputs observed from an activated functional chip. These special input/output pairs are referred to as *distinguishing input/output (DIO) pairs*. Each DIO can be used to identify a subset of *wrong*

key combinations based on circuit satisfiability checking. Together, these DIOs guarantee that only the correct key can be consistent with these correct I/O pairs. This implies that a key that correctly matches the inputs to the outputs for all the DIOs must be the correct key. The crus of the SAT attack is to find this set of DIOs by iteratively building and solving a sequence of SAT formulas (which will be discussed later).

Definition 1 (**Wrong key combination**)**.** Consider the logic function $\vec{Y} = f_l(\vec{X}, \vec{K})$ and its SAT formula $C(\vec{X}, \vec{K}, \vec{Y})$ in conjunctive normal form (CNF). Let $(\vec{X}, \vec{Y}) = (\vec{X_i}, \vec{Y_i})$, where $(\vec{X_i}, \vec{Y_i})$ is a correct I/O pair. The set of key combinations WK_i which result in an incorrect output of the logic circuit (*i.e.,* $\vec{Y_i} \neq f_l(\vec{X_i}, \vec{K})$, $\forall \vec{K} \in WK_i$) is called the set of wrong key combinations identified by the I/O pair $(\vec{X_i}, \vec{Y_i})$. In terms of SAT formula, it can be represented as $C(\vec{X_i}, \vec{K}, \vec{Y_i}) = False$, $\forall \vec{K} \in WK_i$.

Definition 2 (**Distinguishing input/output (DIO) pair**)**.** As noted above, the SAT attack shall solve a set of SAT formulas iteratively. In each iteration, it shall find a correct I/O pair to identify a subset of wrong key combinations until none of these are left. An I/O pair at i-th iteration is a DIO, denoted as $(\vec{X_i^d}, \vec{Y_i^d})$, if it can identify a *unique* subset of wrong key combinations that cannot be identified by the previous $i-1$ DIOs, *i.e.,* $WK_i \nsubseteq (\cup_{j=1}^{j=i-1} WK_j)$, where WK_i is the set of wrong key combinations identified by the DIO at i-th iteration.

The crux of the SAT attack algorithm relies on finding the DIOs iteratively to identify *unique* wrong key combinations (see Definition 2) until no further ones can be found. At this point, the set of all DIOs *together* can identify all wrong key

combinations thereby revealing the correct one. An illustration of the SAT attack process is shown in Fig. 3.2. In each iteration, the SAT attack will find a new DIO that can rule out a subset of wrong key combinations WK_i. Notice that each iteration can identify unique wrong key combinations that are not belong to the ones discovered previously, *i.e.*, $WK_i \nsubseteq (\cup_{j=1}^{j=i-1} WK_j)$. The attack terminates when all wrong key combinations are identified.

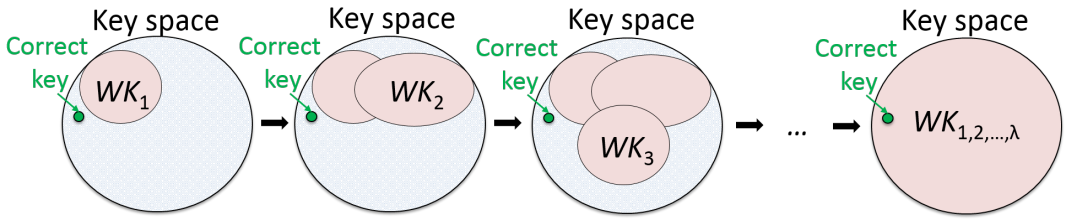


Figure 3.2: Illustration of the iterative SAT attack process. Wrong key combinations are iteratively identified by a set of DIOs till no new ones exist. WK_i is the set of wrong key combinations identified by i-th DIO.

Take the XOR/XNOR based locked circuit in Fig. 3.1 as an example. At first iteration, the I/O pair $(\vec{X}_1^d, \vec{Y}_1^d) = (00, 10)$ is a distinguishing I/O pair because it can rule out wrong key combinations $\vec{K} = (01)$, (10), and (11) as these key combinations will result in incorrect outputs $(y_1 y_2) = (11)$, (00) and (01), respectively. Since this single I/O observation has already ruled out all incorrect key combinations, we have revealed the correct key $\vec{K} = (00)$. In general, a small number of correct I/O pairs (compared to all possible I/O pairs) are usually enough to infer the correct key [24]. As a result, the SAT attack is efficient because it only requires a small number of iterations to find these distinguishing I/O pairs.

3.2.3 Attack Algorithm

As noted above, the central theme of SAT attack algorithm is to iteratively find distinguishing I/O pairs till no new ones can be found. To find such distinguishing I/O pairs, the SAT attack algorithm iteratively formulates a SAT formula that can be solved by SAT solvers. The SAT formula F_i at i-th iteration is:

$$\begin{aligned} F_i :=& C(\vec{X}, \vec{K}_1, \vec{Y}_1) \wedge C(\vec{X}, \vec{K}_2, \vec{Y}_2) \wedge (\vec{Y}_1 \neq \vec{Y}_2) \\ & (\bigwedge_{j=1}^{j=i-1} C(\vec{X}_j^d, \vec{K}_1, \vec{Y}_j^d)) \wedge (\bigwedge_{j=1}^{j=i-1} C(\vec{X}_j^d, \vec{K}_2, \vec{Y}_j^d)) \end{aligned} \tag{3.1}$$

Let's look at this SAT formula F_i line by line. Recall that $C(\vec{X}, \vec{K}, \vec{Y})$ is the SAT formula of a locked circuit with PIs $\vec{X}$, key-inputs $\vec{K}$, and POs $\vec{Y}$. Besides, $\vec{X}_j^d$ is the distinguishing input identified in the previous j-th iteration and $\vec{Y}_j^d$ is the corresponding correct output. This correct output is know from the activated functional chip obtained from the open market.

- The first line of Eq. (3.1) can be interpreted as a Miter-like circuit [64] as shown in Fig. 3.3. Specifically, $C(\vec{X}, \vec{K}_1, \vec{Y}_1)$ can be viewed as the first copy of the locked circuit and $C(\vec{X}, \vec{K}_2, \vec{Y}_2)$ is the second copy of the locked circuit, where $\vec{X}$, $\vec{K}_1$, $\vec{K}_2$, $\vec{Y}_1$, $\vec{Y}_2$ are all unknown variables. As seen, these two circuit copies share the same PIs $\vec{X}$ but have different key-inputs and different POs. The clauses $(\vec{Y}_1 \neq \vec{Y}_2)$ enforce that two POs must be different in order to satisfy this SAT formula.

- In the second line of Eq. (3.1), $C(\vec{X}_j^d, \vec{K}_1, \vec{Y}_j^d)$ can be viewed as another copy of the locked circuit, where its PIs are fixed to known values $\vec{X}_j^d$, POs are

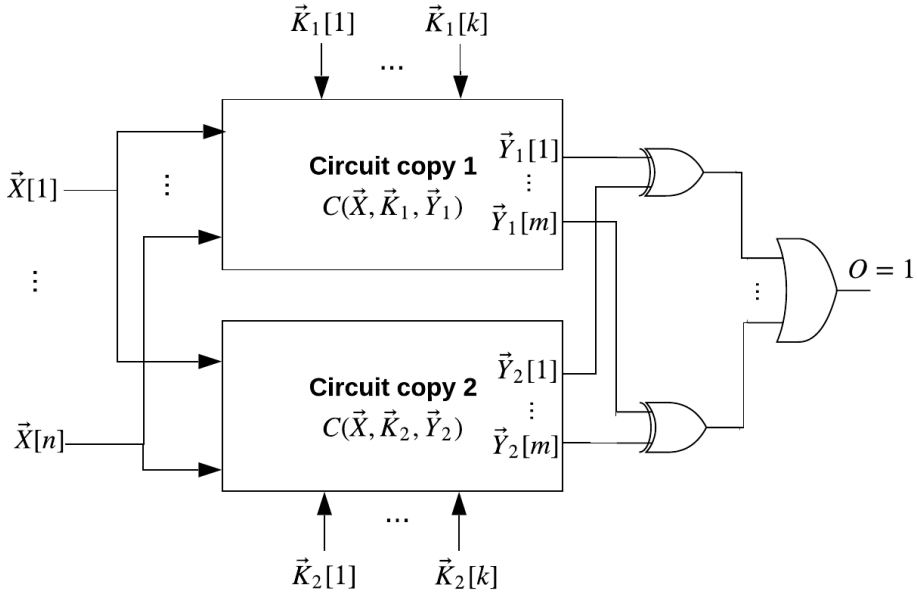


Figure 3.3: Miter-like circuit for finding distinguishing inputs.

fixed to known values $\vec{Y}_j^d$, and the key-inputs are connected to $\vec{K}_1$. Similarly, $C(\vec{X}_j^d, \vec{K}_2, \vec{Y}_j^d)$ can be viewed in the same way but the key-inputs are connected to $\vec{K}_2$.

If Eq. (3.1) is satisfiable, an assignment for variables $\vec{X}$, $\vec{K}_1$, $\vec{K}_2$, $\vec{Y}_1$, $\vec{Y}_2$ will be generated. Let's denote the values assigned to $\vec{X}$ as $\vec{X}_i^d$. The first line in the formula Eq. (3.1) guarantees that the input $\vec{X} = \vec{X}_i^d$ is capable of identifying two keys $\vec{K}_1$, $\vec{K}_2$ which produce different outputs (see $\vec{Y}_1 \neq \vec{Y}_2$). In other words, at least one of the key assignments is incorrect. This in itself is not enough to call $\vec{X} = \vec{X}_i^d$ as a distinguishing input. According to Definition 2, a distinguishing input in the i-th iteration must find *unique* wrong key combinations that have not been identified by previous $i-1$ DIOs. This condition is checked by the second line of Eq. (3.1). The clauses in line 2 guarantee that the keys $\vec{K}_1$ and $\vec{K}_2$ which result in *different* outputs in line 1 of this formula produce the *correct* outputs for all previous DIOs. Hence, in

this iteration we could identify at least one incorrect key combination which previous iterations could not. Therefore, by Definition 2 the input $\vec{X}_i^d$ (obtained from the SAT solver) and the corresponding correct output $\vec{Y}_i^d$ (obtained from the activated chip) represent the i-th DIO pair. The processing of finding DIOs is continued till no new ones can be found (assuming after λ iterations). At this point, a correct key can be obtained by solving the following SAT formula G:

$$G := \bigwedge_{i=1}^{\lambda} C(\vec{X}_i^d, \vec{K}, \vec{Y}_i^d) \tag{3.2}$$

Basically it finds a key $\vec{K}$ which satisfies the correct functionality for all the DIOs. This must be the correct key since no other DIOs exist at this point (see Definition 2).

The SAT attack algorithm is shown in Algorithm 1. It starts by first solving the line 1 of the SAT formula Eq. (3.1) and as iterations progress it adds the clauses comprised in line 2 of the formula Eq. (3.1). It stops when the resulting SAT formula is unsatisfiable indicating no further DIOs exist. The correct key is obtained by finding a key value which satisfies the correct I/O behavior of all the DIOs (Eq. (3.2)). This algorithm is guaranteed to find the correct key. Please refer to [24] for any further theoretical details.

3.3 Motivation and Problem Statement

As discussed in Section 3.2, the SAT attack [24] has created a new security concern on the logic locking technique. Note that the SAT attack is an iterative

Algorithm 1 SAT Attack Algorithm [24]

Input: C and $eval$

Output: $\vec{K}_C$

1: $i := 1$;

2: $G_i := True$;

3: $F_i := C(\vec{X}, \vec{K}_1, \vec{Y}_1) \wedge C(\vec{X}, \vec{K}_2, \vec{Y}_2) \wedge (\vec{Y}_1 \neq \vec{Y}_2)$;

4: **while** sat$[F_i]$ **do**

5: $\quad \vec{X}_i^d := \text{sat\_assignment}_{\vec{X}}[F_i]$;

6: $\quad \vec{Y}_i^d := eval(\vec{X}_i^d)$;

7: $\quad G_{i+1} := G_i \wedge C(\vec{X}_i^d, \vec{K}, \vec{Y}_i^d)$;

8: $\quad F_{i+1} := F_i \wedge C(\vec{X}_i^d, \vec{K}_1, \vec{Y}_i^d) \wedge C(\vec{X}_i^d, \vec{K}_2, \vec{Y}_i^d)$;

9: $\quad i := i + 1$;

10: **end while**

11: $\vec{K}_C := \text{sat\_assignment}_{\vec{K}}(G_i)$;

process, the efficiency of SAT attack can be evaluated by the total execution time:

$$T = \sum_{i=1}^{\lambda} t_i \tag{3.3}$$

where λ is the total number of SAT attack iterations and t_i is the SAT solving time for i-th iteration. Consequently, the SAT attack can be mitigated if t_i is large and/or λ is large. λ depends on the key-size and key location in the locked circuit. However, simply increasing the key-size or trying different key locations may not effectively thwart the SAT attack. As shown in the SAT attack results [24], even with large number of keys (50% area overhead), for six previously proposed key-gate insertion algorithms [20, 22, 47, 50, 53], 86% benchmarks on average can still be unlocked in

10 hours.

This serious security threat motivates us to investigate a SAT attack resistant logic locking design. We want to develop a light-weight SAT-attack resistant circuit block (denoted as the *Anti-SAT block*) to enhance the security of conventional logic locking against the SAT attack. The Anti-SAT block will be integrated into the original circuit to thwart the SAT attack. The objective is to construct the Anti-SAT block and use a mathematically rigorous approach to prove that if chosen correctly, the Anti-SAT block makes the SAT attack iterations grow exponentially in key-size, thereby making SAT attack computationally infeasible. Besides, to prevent the Anti-SAT block from being identified (and removed by an attacker), we shall develop obfuscation techniques to hide the functionality and structure of the Anti-SAT block.

3.4 Anti-SAT Based Logic Locking

To mitigate the SAT attack, we propose to insert a relatively light-weight circuit block (referred to as Anti-SAT block) that can efficiently increase the number of iterations λ so as to increase the total execution time T.

3.4.1 Anti-SAT Configurations

Fig. 3.4(a) and Fig. 3.4(b) illustrate two configurations of the proposed Anti-SAT block, referred to as **type-0 Anti-SAT** and **type-1 Anti-SAT**. They consist of two logic blocks g and $\overline{g}$, which share the same set of inputs $\vec{X} = (X_1...X_n)$.

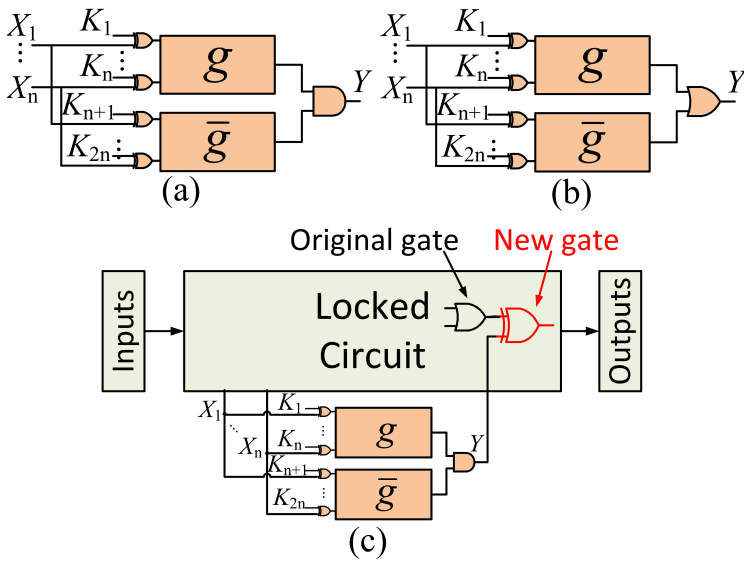


Figure 3.4: Anti-SAT block configuration: (a) type-0 Anti-SAT: always outputs 0 if key values are correct; (b) type-1 Anti-SAT: always outputs 1 if key values are correct; (c) integrating the Type-0 Anti-SAT block into a circuit.

The functionalities of g and $\overline{g}$ are complementary. A set of key-gates (XORs [1]) are inserted at the inputs of two logic blocks, denoted as $\vec{K}_{l1} = (K_1...K_n)$ and $\vec{K}_{l2} = (K_{n+1}...K_{2n})$. Hence the key-size is $2n$. The output of g and $\overline{g}$ are fed into an AND2 gate (for Fig. 3.4(a)) or an OR2 gate (for Fig. 3.4(b)) to form the final single-bit output Y. As a result, we have $Y = g(\vec{X} \oplus \vec{K}_{l1}) \wedge \overline{g(\vec{X} \oplus \vec{K}_{l2})}$ for type-0

[1]Note that here we are using only XOR gates as key-gates for the sake of ease of explanation. The key-gates used could be either XOR or XNOR gates (+ inverters) based on a user-defined key [47]. The usage of inverters can remove the association between key-gate types and key-values (*e.g.,* the correct key into an XOR gate can be either 0 or 1). Besides, the synthesis tools can "bubble push" the inverters to their fan-out gates and an attacker cannot easily identify which inverters are part of the key-gates [47]. Therefore, the attacker cannot obtain the correct key-values by simply inspecting the key-gate types.

Anti-SAT and $Y = g(\vec{X} \oplus \vec{K}_{l1}) \vee \overline{g(\vec{X} \oplus \vec{K}_{l2})}$ for type-1 Anti-SAT.

Constant-output Property: One basic property of Anti-SAT block is that when the key vector is correctly set, the output Y is a constant. Specifically, given a correct key, Y always outputs value 0 for type-0 Anti-SAT (Fig. 3.4(a)) and always outputs value 1 for type-1 Anti-SAT (Fig. 3.4(b)). On the other hand, when a wrong key is given, Y can output either 1 or 0 depending on the inputs $\vec{X}$. This property enables it to be integrated into the original circuit. Fig. 3.4(c) shows an example of integrating a type-0 Anti-SAT into a circuit. As seen, the inputs of Anti-SAT block $\vec{X}$ are from the wires in the original circuit. The output Y is connected into the original circuit using an XOR gate. When a correct key is provided, the output Y always equals to 0 (so the XOR gate behaves as a buffer) and thus will not affect the functionality of the original circuit. If a wrong key is provided, Y can be 1 for some inputs (so the XOR gate behaves as an inverter) and thus can produce a fault in the original circuit. Similarly, the type-1 Anti-SAT block can be integrated into the original circuit using an XNOR gate.

Correct Keys: Since the Anti-SAT block has $2n$ keys, the total number of wrong key combinations is $2^{2n} - c$, assuming there exists c correct key combinations. To ensure the constant-output property, the correct keys for the Anti-SAT block would be the ones that make type-0 Anti-SAT always output 0 and type-1 Anti-SAT always output 1. We can design g such that this happens when i-th key-bit from $\vec{K}_{l1}$ and i-th key-bit from $\vec{K}_{l2}$ have the same value. So the number of correct key combinations $c = 2^n$ for both types of Anti-SAT blocks and the number of wrong

key combinations is $2^{2n} - 2^n$.

In the subsequent sections, we provide details on constructing the Anti-SAT block (*i.e.,* the functionality of g) and its impact on SAT attack complexity. We provide a rigorous mathematical analysis which gives a provable lower bound to the number of SAT attack iterations. For some constructions of g, this lower bound is exponential in the key-size thereby making the SAT-attack complexity very high.

3.4.2 SAT Attack Resistance Analysis

Here we analyze the complexity of SAT attack on the Anti-SAT block (assuming this is the circuit being attacked to decipher the $2n$ key bits).

Terminology Given a Boolean function $g(\vec{L})$ with n inputs, assuming there exists p input vectors that make g equal to one (denote p as *on-set size*, $1 \leq p \leq 2^n - 1$), we can classify the input vectors $\vec{L}$ into two groups L^T and L^F, where one group makes $g = 1$ and another makes $g = 0$:

$$\begin{aligned} L^T &= \{\vec{L} | g(\vec{L}) = 1\}, \quad (|L^T| = p) \\ L^F &= \{\vec{L} | g(\vec{L}) = 0\}, \quad (|L^F| = 2^n - p) \end{aligned} \tag{3.4}$$

The function g and its complementary function $\overline{g}$ are used to construct the Anti-SAT block as shown in Fig. 3.4.

Theorem 3.4.1. Assuming the on-set size p of function g is sufficiently close to 1 or sufficiently close to $2^n - 1$, the number of iterations needed by the SAT attack to decipher the correct key is lower bounded by 2^n.

Proof for Type-0 Anti-SAT. As shown in Section 3.2, the SAT attack algorithm will

iteratively find a DIO ($\vec{X}_i^d$, Y_i^d) to identify wrong key combinations in the Anti-SAT block until all wrong key combinations are identified. In the i-th iteration, the corresponding DIO can identify a subset of wrong key combinations, denoted as WK_i. Notice that for any input combinations (including the distinguishing inputs $\vec{X}_i^d$), the correct output (when provided the correct key) is 0 for type-0 Anti-SAT. Therefore, a wrong key combination $\vec{K} = (\vec{K}_{l1}, \vec{K}_{l2}) \in WK_i$ which was identified by ($\vec{X}_i^d$, Y_i^d) must produce the Anti-SAT block output incorrectly as 1. This condition is described below.

$$\begin{aligned} & Y_i^d = g(\vec{X}_i^d \oplus \vec{K}_{l1}) \wedge \overline{g(\vec{X}_i^d \oplus \vec{K}_{l2})} = 1 \\ \Leftrightarrow\ & (g(\vec{X}_i^d \oplus \vec{K}_{l1}) = 1) \wedge (g(\vec{X}_i^d \oplus \vec{K}_{l2}) = 0) \\ \Leftrightarrow\ & ((\vec{X}_i^d \oplus \vec{K}_{l1}) \in L^T) \wedge ((\vec{X}_i^d \oplus \vec{K}_{l2}) \in L^F) \end{aligned} \tag{3.5}$$

Basically Eq. (3.5) states that the wrong key identified in the i-th iteration must be such that its output Y_i^d should be 1. This implies that both g and $\overline{g}$ must evaluate to 1. This means that the input to g, which is $\vec{X}_i^d \oplus \vec{K}_{l1}$, should be in L^T and the input to $\overline{g}$, which is $\vec{X}_i^d \oplus \vec{K}_{l2}$, should be in L^F.

Since $\vec{X}_i^d \oplus \vec{K}_{l1}$ is the input vector to g, for any given $\vec{X}_i^d$, we can always find a key $\vec{K}_{l1}$ such that $\vec{X}_i^d \oplus \vec{K}_{l1} \in L^T$. Basically $\vec{X}_i^d \oplus \vec{K}_{l1}$ flips some of the bits of $\vec{X}_i^d$ (for which corresponding $\vec{K}_{l1}$ bits are 1) while keeping other bits the same (for which corresponding $\vec{K}_{l1}$ bits are 0). Hence for a given $\vec{X}_i^d$, we can always choose $\vec{K}_{l1}$ such that the resulting input to g is in L^T. However note that $|L^T| = p$ in Eq. (3.4). Hence for any given $\vec{X}_i^d$, we can select $\vec{K}_{l1}$ in p different ways such that $\vec{X}_i^d \oplus \vec{K}_{l1} \in L^T$.

Similarly, for any given $\vec{X}_i^d$, we can always find a key $\vec{K}_{l2}$ such that $\vec{X}_i^d \oplus \vec{K}_{l2} \in L^F$. Note that $|L^F| = 2^n - p$ in Eq. (3.4). Hence for any given $\vec{X}_i^d$, we can select $\vec{K}_{l2}$ in $2^n - p$ different ways such that $\vec{X}_i^d \oplus \vec{K}_{l2} \in L^F$.

Now, as noted above, for a given $\vec{X}_i^d$, a wrong key $\vec{K} = (\vec{K}_{l1}, \vec{K}_{l2})$ should be such that $\vec{X}_i^d \oplus \vec{K}_{l1} \in L^T$ and $\vec{X}_i^d \oplus \vec{K}_{l2} \in L^F$. The total number of ways in which we can select such a wrong key is $p \cdot (2^n - p)$.

Now in any given iteration i, for a given X_i^d, the *maximum* number of incorrect keys identified is $p \cdot (2^n - p)$. This follows naturally from the discussion above. This is the *maximum* number because it is very much possible that some of these keys were identified in *previous* iterations. Hence the total number of *unique* wrong keys UK_i identified in iteration i is upper-bounded by $p \cdot (2^n - p)$. This is noted in the equation below.

$$p \cdot (2^n - p) \geq UK_i \tag{3.6}$$

The SAT attack works by iteratively removing all incorrect keys till only the correct ones are left (assuming after λ iterations). Hence the following holds true.

$$\lambda(p \cdot (2^n - p)) \geq \sum_{i=1}^{\lambda} UK_i \tag{3.7}$$

Since $\sum_{i=1}^{\lambda} UK_i$ is the total number of wrong key combinations, its value is $2^{2n} - 2^n$ as discussed in Section 3.4.1. Eq. (3.7) can be rewritten as follows.

$$\lambda \geq \frac{2^{2n} - 2^n}{p(2^n - p)} \tag{3.8}$$

We denote this lower bound on λ as λ_0. When $p \to 1$ or $p \to 2^n - 1$, we have

$$\lambda_0 = \frac{2^{2n} - 2^n}{p(2^n - p)} \to \frac{2^{2n} - 2^n}{1 \times (2^n - 1)} = 2^n \tag{3.9}$$

Hence proved.

■

Proof for Type-1 Anti-SAT. For type-1 Anti-SAT, the correct output (when provided the correct key) is always 1. Therefore, a wrong key combination $\vec{K} = (\vec{K}_{l1}, \vec{K}_{l2}) \in WK_i$ which was identified by $(\vec{X}_i^d, Y_i^d)$ must produce the incorrect output as 0. This condition is described below.

$$\begin{aligned} & Y_i^d = g(\vec{X}_i^d \oplus \vec{K}_{l1}) \vee \overline{g(\vec{X}_i^d \oplus \vec{K}_{l2})} = 0. \\ \Leftrightarrow & (g(\vec{X}_i^d \oplus \vec{K}_{l1}) = 0) \wedge (g(\vec{X}_i^d \oplus \vec{K}_{l2}) = 1) \\ \Leftrightarrow & ((\vec{X}_i^d \oplus \vec{K}_{l1}) \in L^F) \wedge ((\vec{X}_i^d \oplus \vec{K}_{l2}) \in L^T) \end{aligned} \tag{3.10}$$

Based on the discussion in the proof for type-0 Anti-SAT, we know that for any given $\vec{X}_i^d$, we can select $\vec{K}_{l1}$ in $2^n - p$ different ways such that $\vec{X}_i^d \oplus \vec{K}_{l1} \in L^F$. Also, for any given $\vec{X}_i^d$, we can select $\vec{K}_{l2}$ in p different ways such that $\vec{X}_i^d \oplus \vec{K}_{l2} \in L^T$. As noted in Eq. (3.10), for a given $\vec{X}_i^d$, a wrong key $\vec{K} = (\vec{K}_{l1}, \vec{K}_{l2})$ should be such that $\vec{X}_i^d \oplus \vec{K}_{l1} \in L^F$ and $\vec{X}_i^d \oplus \vec{K}_{l2} \in L^T$. The total number of ways in which we can select such a wrong key is $p \cdot (2^n - p)$, which is exactly the same as the one for type-0 Anti-SAT. Therefore, the subsequent analysis would be the same as the analysis for type-0 Anti-SAT and we can obtain the same lower bound λ_0 as shown in Eq. (3.9). Hence proved.

■

As seen in Eq. (3.9), if we choose a g function such that p is either very low or very high then the SAT attack would at least require an exponential number of

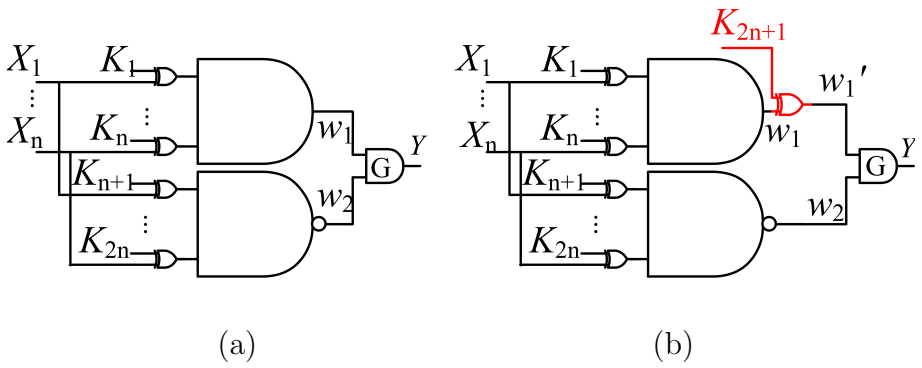


Figure 3.5: Anti-SAT block design and obfuscation: (a) one possible construction of function g to ensure large number of SAT attack iterations; (b) an additional key-gate is inserted for functional obfuscation.

iterations in n. Since the key-size of Anti-SAT is $2n$, the number of SAT attack iterations is also an exponential number in the key-size of Anit-SAT when g is correctly configured. One possible choice of g is indicated in Fig. 3.5(a) where g is chosen to be a simple n-input AND gate. For AND gates $p = 1$ which clearly results in exponential complexity of SAT attack in n. Experimental results to indicate that shall be shown later. Moreover, we can see that the lower bound λ_0 is tight when $p = 1$ or $p = 2^n - 1$. This is because that for a n-input Anti-SAT block, the total number of input combinations is 2^n so the number of iterations to find distinguishing inputs is upper-bounded, *i.e.,* $\lambda \leq 2^n$. This combined with the Eq. (3.9) shows that the lower bound is tight when $p = 1$ or $p = 2^n - 1$.

3.4.3 Integrating Anti-SAT into a Circuit

When the Anti-SAT block is integrated into a circuit, a set of wires in the original circuit are connected to the inputs $\vec{X}$ of the Anti-SAT block and the output

Y of the Anti-SAT block is integrated to a wire in the original circuit (as shown in Fig. 3.4(c)). If $\vec{X}$ are connected to wires that are highly correlated (*e.g.,* two wires with identical logic), then the overall security of the block shall be reduced because less possible input combinations can occur at the inputs of the Anti-SAT block. The location for Y is also important. An incorrect key causes $Y = 1$ for some inputs (for type-0 Anti-SAT). This incorrect Anti-SAT output must impact the overall functionality of the original circuit. Otherwise the logic will continue to function correctly despite of wrong key inputs. In conclusion, the best location of the Anti-SAT block is such that the inputs $\vec{X}$ are highly independent and Y has high observability at the POs (*i.e.,* changes in Y can be observed by the POs of the original circuit). Here we propose a *secure integration* method: n inputs of the Anti-SAT block $\vec{X}$ are connected to n PIs of the original circuit. The output Y is connected to a wire which is randomly selected from wires that have the top 30% observability. The randomness of the location of Y can assist in hiding the output wire of the Anti-SAT block and preventing it from being identified and nullified. The impact of the Anti-SAT integration location on the overall security shall be evaluated in the experiments.

3.4.4 A Combined Locking Approach

As noted before, conventional logic locking techniques such as [22, 47] try to avoid an unauthorized user who does not have a key from accessing the chip's functionality. They attempt to insert key gates in a way to achieve high output-

corruptibility, *i.e.,* forcing the chip to deviate substantially from the actual functionality whenever a wrong key is provided. These techniques are not immune to SAT attack (as noted in [24] and also indicated in our simulations). While our Anti-SAT block can provide provable measures to increasing the SAT attack complexity, they may not necessarily cause substantial deviation in the chip functionality for incorrect keys. Hence an unauthorized end user may still be able to use the chip correctly for "many" inputs (but not all). Therefore, conventional logic locking techniques need to be combined with our Anti-SAT block designs for achieving foolproof logic locking. Moreover, the key-gates inserted at the original circuit can make the Anti-SAT block less distinguishable with the original circuit. Without these key-gates in the original circuit, an attacker has less difficulty to locate the Anti-SAT block by inspecting the only key-inputs into the Anti-SAT block.

In this work, the original circuit is locked using the secure logic locking (SLL), an interference-based logic locking algorithm [22]. This technique has been shown to be secure against key-sensitizing attack [22] (see Section 2.2.2.2) while obfuscating the original functionality.

3.5 Anti-SAT Block Obfuscation

In this section, we first analyze the security of Anti-SAT against a new type of attack called *removal attack*, which aims at identifying and nullifying the Anti-SAT block. Then, we investigate a unified obfuscation based on [51] which hides the functional and structural traces of the Anti-SAT block.

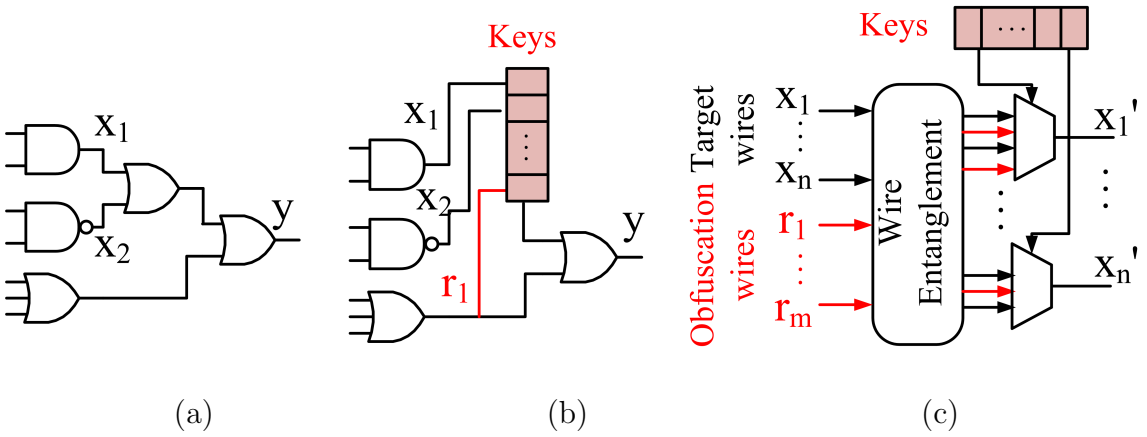
3.5.1 Removal Attacks on Anti-SAT

3.5.1.1 Functional Attributes

In Anti-SAT, the logic blocks g and $\overline{g}$ have complementary functionality. An attacker can simulate the circuit and find potential complementary pairs of signals leading to potential identification of the Anti-SAT block. Moreover, in order to guarantee exponential number of SAT attack iterations, the function g shall be configured to have very small on-set size p. Assuming $p = 1$, the outputs of $g/\overline{g}$ would be 0/1 for most of the time even when wrong keys are provided for the Anti-SAT block. In other words, the outputs of g and $\overline{g}$ will have very high signal skews of opposite polarities. This functional attribute is exploited by *Signal Probability Skew (SPS) attack* [65] to identify the Anti-SAT block. Another functionality attribute of Anti-SAT block is its low functionality corruptibility. Due to the construction of g and $\overline{g}$, the Anti-SAT keys normally have lower output corruptibility than the conventional keys in the original circuit. Recently, an approximate de-obfuscation technique called AppSAT [41] was proposed to learn the high-corruptibility conventional keys in the original circuit. An *AppSAT + netlist analysis based removal attack* [66] has also been proposed to identify the Anti-SAT block.

3.5.1.2 Structural Attributes

In the Anti-SAT block, the internal wires in g and $\overline{g}$ do not have connections with the locked circuit. This makes the Anti-SAT block a relatively isolated and



(a) (b) (c)

Figure 3.6: Design withholding and entanglement technique [51]: (a) original circuit; (b) design withholding and (c) wire entanglement.

separable structure. When the size of the Anti-SAT block is roughly known, it's possible for an attacker to utilize a partitioning algorithm to partition the whole circuit into two parts while ensuring that small partition has about the same size as the Anti-SAT block. If a large portion of gates of the Anti-SAT block is moved to the small partition, then the attacker will have less difficulty to identify the Anti-SAT block using this *partitioning based removal attack*.

3.5.2 Mitigating Removal Attacks

To mitigate the removal attacks, we propose a unified obfuscation technique that obfuscates the functionality and structural attributes of the Anti-SAT block using design withholding and wire entanglement [51]. Fig. 3.6 illustrate the basic idea of design withholding and wire entanglement. In design withholding (Fig. 3.6(a)), a portion of design is replaced with a set of LUT's to ensure that the original design detail is not available to the untrusted foundry. Hence, design withholding technique

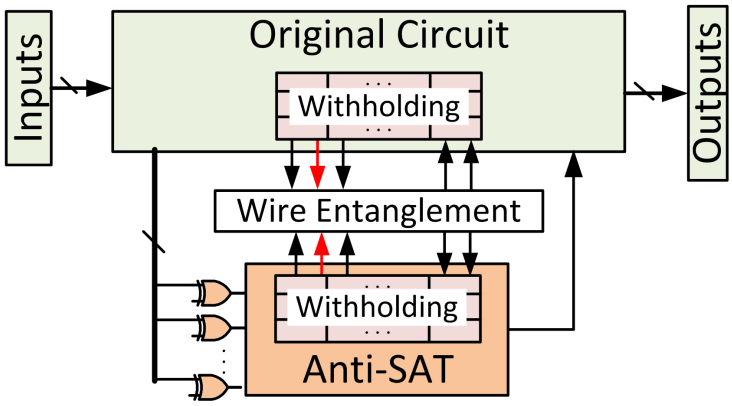


Figure 3.7: Anti-SAT obfuscation based on design withholding and wire entanglement.

can be used to hide both the functionality and implementation detail of the Anti-SAT. Design entanglement is another obfuscation technique that aims at obfuscating the interconnect structure of an IC design by using a wire-entanglement module, as shown in Fig. 3.6(b). The basic idea of wire-entanglement module is to entangle a set of target wires with another set of obfuscation wires using MUX-based interconnect network. When the selection bits of the MUXes are correctly configured, the wire-entanglement module will represent the original interconnection. The wire-entanglement module is useful for obfuscating the interconnect structure between the Anti-SAT block and the original netlist.

Fig. 3.7 illustrates the overall obfuscation for Anti-SAT based on design withholding and wire entanglement. The design withholding technique is used to hide the functionality of the Anti-SAT block and part of the original netlist. When obfuscated using LUTs, the signal skews of g and $\overline{g}$ would be significantly reduced, so the SPS attack cannot effectively identify the Anti-SAT block. Note that the

obfuscation for g and $\overline{g}$ does not necessarily need to be balanced (*i.e.,* the key-size of g and $\overline{g}$ can be different) as long as the outputs of g and $\overline{g}$ do not have a high signal skew. This can help mitigating the AppSAT + netlist analysis attack which assumes both g and $\overline{g}$ would have roughly the same key-size and use it as a hint to locate the Anti-SAT.

On the other hand, wire-entanglement technique is used to obfuscate the interconnection between the original circuit and the Anti-SAT block to prevent the partitioning-based attack. With the wire-entanglement module, the interconnections between the Anti-SAT block and the locked circuit will be increased and it's difficult for an attacker to partition and isolate the Anti-SAT block from the locked circuit. Besides, the design withholding and entanglement technique can be designed to mitigate the AppSAT + netlist analysis attack. This is because that after wire entanglement, new signal paths would be created which fanout many low-corruptibility keys to the original netlist. Therefore, gates in the original netlist can have many low-corruptibility keys in their fan-in cones. It increases the difficulty of the netlist analysis phase which tries to identify the Anti-SAT by counting the number of low-corruptibility keys in a gate's fan-in.

With regard to SAT attack, since these techniques are based on LUT-based and MUX-based logic locking, they can be modeled in SAT formula and attacked by the SAT attack. However, in Section 3.5.3, we will use a proof to show that the number of SAT attack iterations for unlocking a circuit with an obfuscated Anti-SAT will not be reduced. Experimental results in Section 3.6.3 also validate this analysis.

3.5.3 SAT-attack Resistance of Anti-SAT After Obfuscation

In Section 3.5.2, a unified obfuscation technique for Anti-SAT block based on withholding and entanglement [51] is discussed. It basically obfuscates the Anti-SAT block by adding additional logic gates and key-inputs. Here we use a rigorous proof to show that the resistance of Anti-SAT block would not be weakened when obfuscation technique is applied. In other words, adding addition key-gates and key-inputs will not reduce the number of SAT attack iterations required to decipher the Anti-SAT block.

We first show that adding one additional key-gate for obfuscation will not reduce the number of SAT attack iterations required for unlocking the Anti-SAT. Without loss of generality, we use XOR/XNOR-based key-gates instead of LUT-based key-gates to obfuscate the Anti-SAT in order to simply the proof. Also, we insert the extra key-gate as shown in Fig. 3.5(b).

Theorem 3.5.1. Assuming a new key-gate K_{2n+1} is inserted into the Anti-SAT block (with $p = 1$) for obfuscation (as shown in Fig. 3.5(b)), the number of SAT attack iterations needed by the SAT attack to decipher the correct key will remain to be 2^n.

Proof: Let's first derive the equation which represents the wrong key combinations that can be identified by a DIO $(\vec{X}_i^d, \vec{Y}_i^d)$. Notice that for any input combinations (including the distinguishing inputs $\vec{X}_i^d$), the correct output (when provided a correct key) is 0 for type-0 Anti-SAT. Therefore, a wrong key combination which is identified by $(\vec{X}_i^d, \vec{Y}_i^d)$ must result in incorrect output as 1. This condition is de-

scribed as

$$\begin{aligned}&[(K_{2n+1}=0)\wedge(g(\vec{X}_i^d\oplus\vec{K}_{l1})=1)\wedge(\overline{g(\vec{X}_i^d\oplus\vec{K}_{l2})}=1)]\\ \vee&[(K_{2n+1}=1)\wedge(g(\vec{X}_i^d\oplus\vec{K}_{l1})=0)\wedge(\overline{g(\vec{X}_i^d\oplus\vec{K}_{l2})}=1)]\end{aligned}\tag{3.11}$$

This is equivalent to

$$\begin{aligned}&[(K_{2n+1}=0)\wedge(\vec{X}_i^d\oplus\vec{K}_{l1}\in L^T)\wedge(\vec{X}_i^d\oplus\vec{K}_{l2}\in L^F)]\\ \vee&[(K_{2n+1}=1)\wedge(\vec{X}_i^d\oplus\vec{K}_{l1}\in L^F)\wedge(\vec{X}_i^d\oplus\vec{K}_{l2}\in L^F)]\end{aligned}\tag{3.12}$$

Note that the function g in Fig. 3.5(b) is an n-input AND gate,

$$L^T=\{(11...11)\},\quad L^F=\mathbb{B}^n\setminus L^T\tag{3.13}$$

where $\mathbb{B}^n$ is the set of all n-bit boolean vectors, and $\mathbb{B}^n\setminus L^T$ means every n-bit Boolean vector except (11...11).

We can see that when $K_{2n+1}=0$, to satisfy Eq. (3.12), we need to ensure $\vec{X}_i^d\oplus\vec{K}_{l1}\in L^T$ and $\vec{X}_i^d\oplus\vec{K}_{l2}\in L^F$. Since L^T has only one vector (11...11), for any $\vec{X}_i^d$, we only have one way of selecting $\vec{K}_{l1}$ to make $\vec{X}_i^d\oplus\vec{K}_{l1}=(11...11)$, that is $\vec{K}_{l1}=\neg\vec{X}_i^d$ (bit-wise negation), *i.e.,*

$$\vec{K}_{l1}[j]=\neg\vec{X}_i^d[j],\quad j=1...n\tag{3.14}$$

On the other hand, since $L^F=\mathbb{B}^n\setminus L^T$, for any $\vec{X}_i^d$, we have 2^n-1 ways to select $\vec{K}_{l2}$ such that $\vec{X}_i^d\oplus\vec{K}_{l2}\in L^F$, those are $\vec{K}_{l2}\in\mathbb{B}^n\setminus\neg\vec{X}_i^d$.

Therefore, the wrong key combinations (with $K_{2n+1}=0$) identified by $(\vec{X}_i^d,Y_i^d)$ has the following form:

$$(\vec{K}_{l1}=\neg\vec{X}_i^d,\vec{K}_{l2}\in(\mathbb{B}^n\setminus\neg\vec{X}_i^d),K_{2n+1}=0)\tag{3.15}$$

We can see that since $\vec{K}_{l1} = \neg\vec{X}_i^d$, there exists an one-to-one matching between each pair of $\vec{X}_i^d$ and $\vec{K}_{l1}$. In other words, any $\vec{X}_i^d$ value (from 0 to $2^n - 1$) can identify a unique set of wrong key combinations in a form of Eq. (3.15). It's unique because that $\vec{K}_{l1} = \neg\vec{X}_i^d$ and different $\vec{X}_i^d$ would result in different $\vec{K}_{l1}$. Therefore, every input combination (from 0 to $2^n - 1$) is a distinguishing input because it can identify a unique set of wrong key combinations that can only be identified by it. Thus, the SAT attack requires 2^n DIOs (*i.e.,* 2^n iterations) to identify all wrong key combinations.

■

We now show that adding more than one additional key-gates for obfuscation still does not reduce the number of SAT attack iterations required for unlocking the Anti-SAT.

Theorem 3.5.2. Assuming n_{obf} new key-gates are inserted into the Anti-SAT block (with $p = 1$) for obfuscation, the number of SAT attack iterations needed by the SAT attack to decipher the correct key will be 2^n.

Proof: The proof for Theorem 2 shows that after adding a new key-gate to the Anti-SAT block for obfuscation (Fig. 3.5(b)), the number of SAT attack iterations remains to be 2^n. Notice that this conclusion is also true when n_{obf} additional key-gates are inserted to the Anti-SAT for obfuscation. Let's denote the extra keys for obfuscation as $\vec{K}_{obf}$ and its correct key is $\vec{K}_{obf}^C$. Based on Eq. (3.15), we can conclude that any Anti-SAT input combination is a distinguishing input $\vec{X}_i^d$ because it can

identify a unique set of wrong keys which is:

$$(\vec{K}_{l1} = \neg\vec{X}_i^d, \vec{K}_{l2} \in (\mathbb{B}^n \setminus \neg\vec{X}_i^d), \vec{K}_{obf} = \vec{K}_{obf}^C) \tag{3.16}$$

and this set of wrong keys can only be identified by this input $\vec{X}_i^d$. Hence the SAT attack requires at least 2^n DIOs (*i.e.,* 2^n iterations) to identify all wrong key combinations. Therefore, adding additional key-gates at different locations for obfuscation will not weaken the Anti-SAT's resistance to the SAT attack.

■

3.6 Experiments and Results

In this section, we evaluate the security level of our proposed Anti-SAT blocks. The security level is evaluated by the number of *SAT attack iterations* as well as the *execution time* to infer the correct key. SAT attack tools and benchmarks used are from [24]. The SAT attack tool uses the Lingeling [67] SAT solver. The CPU time limit is set to 10 hours as [24]. The experiments are running on an Intel Core i5-2400 CPU with 16GB RAM.

3.6.1 Anti-SAT Block Design

3.6.1.1 On-set Size p

Table 3.1 illustrates the impact of p on the security level of 16-bit Anti-SAT blocks (type-0 and type-1). For both types of Anti-SAT, when $p \to 1$ and $p \to 2^{16} - 1 = 65535$, the SAT attack algorithm fails to unlock the Anti-SAT block in

Table 3.1: Impact of p on the security level of Anti-SAT (When $n = 16$).

	p	1	81	243	2187	30375	63349	65293	65455	65535
Type-0	# Iterations	-	10675	4760	901	273	898	4647	-	-
Anti-SAT	Time (s)	timeout	16555.8	8746.12	174.743	3.24	307.104	12932.3	timeout	timeout
Type-1	# Iterations	-	-	4853	877	285	881	4691	-	-
Anti-SAT	Time (s)	timeout	timeout	3559.96	55.108	3.148	187.896	1048.19	timeout	timeout

10 hours. This is because that it requires a large number of iterations to rule out all the incorrect key combinations. As $p \to 2^{16}/2$ (the worst case), the SAT attack begins to succeed using less and less iterations and execution time for both types of Anti-SAT. This result validates out analysis in Eq. (3.9), which shows that for a fixed n, when p is close to 1 or $2^n - 1$, λ will be large and the SAT attack will fail within a practical time limit.

3.6.1.2 Input-size n

As shown in Eq. (3.9), λ_0 is an exponential function of n when p is very low ($p \to 1$) or very high ($p \to 2^n - 1$). Table 3.2 shows the exponential relationship between λ and n when $p = 1$ for type-0 and type-1 Anti-SAT block. It can be seen that as n increases, the simulated SAT iterations and execution time grows exponentially.

In the following experiments, we focus on the type-0 Anti-SAT and construct an *n-bit baseline Anti-SAT block (n-bit BA)* using an n-input AND gate ($p = 1$) as the logic block g to ensure large number of iterations. However notice that this is not the only possible choice for g.

Table 3.2: Impact of n on the security level of Anti-SAT (When $p = 1$).

	n	8	10	12	14	16
Type-0 Anti-SAT	# Iterations	255	1023	4095	16383	-
	Time (s)	1.14569	20.024	324.727	4498.03	timout
Type-1 Anti-SAT	# Iterations	255	1023	4095	16383	-
	Time (s)	1.06	14.612	273.1	3658.76	timeout

Table 3.3: Comparison between secure and random integration.

	n	8	12	16
Random	Avg. # Iteration	151	1748	11461
	Avg. Time (s)	1.4296	162.529	10272.4
Secure	# Iteration	255	4095	-
	Time (s)	3.452	759.924	timeout

3.6.1.3 Secure Integration of Anti-SAT

Here we compare two approaches of integrating the Anti-SAT block with the original circuit, namely *secure integration* and *random integration*. For the *secure integration*, n inputs of the Anti-SAT block $\vec{X}$ are connected to n PIs of the original circuit. The output Y is connected to a wire which is randomly selected from wires that have the top 30% observability. For the *random integration*, the inputs $\vec{X}$ are connected to random wires of the original circuit, and the output Y is connected to a random wire. For both cases, the wire for Y has a later topological order than that of the wires for $\vec{X}$ to prevent combinational loop. Table 3.3 shows the results for two integration approaches when three n-bit BA ($n = 8, 12, 16, p = 1$)

Table 3.4: Benchmark information of 6 circuits from ISCAS85 and MCNC.

Circuit	#PI	#PO	#Gates	Key-size	
				SLL	n-bit BA
c1355	41	32	546	29	2n
c1908	33	25	880	46	
c3540	50	22	1669	86	
dalu	75	16	2298	119	
des	256	245	6473	336	
i8	133	81	2464	130	

are integrated into the c1355 circuit from ISCAS85. It can be seen that secure integration is better than random integration as the former requires more iterations ($\sim 2\times$) and execution time ($\sim 3\times$) for the SAT attack algorithm to reveal the key. Therefore, in the following experiments, we adopt the secure integration as the way to integrate the Anti-SAT block into a circuit.

3.6.2 Anti-SAT Block Application

We evaluate the security level of the Anti-SAT block when it's applied to 6 circuits from ISCAS85 and MCNC benchmark suites. The benchmark information is shown in Table 3.4. We compare two logic locking configurations as follows:

- **SLL**: The original circuit is locked only using the secure logic locking (SLL), an interference-based logic locking algorithm [22]. This technique has been shown to be secure against key-sensitizing attack [22] while obfuscating the original functionality.

- **SLL (5%) + n-bit BA**: The original circuit is locked with SLL technique. The number of key-gates inserted in the original circuit equals to 5% of the gate-size of the original circuit. Besides, an n-bit BA is integrated into the locked circuit using the secure integration (described in Section 3.4.3).

We compare the security level of two configurations when the same number of keys are used in each configuration [2]. The SAT attack results of two configurations w.r.t increasing key-size are shown in Fig. 3.8. It can be seen that for SLL, increasing the key-size cannot effectively increase SAT attack complexity. For all benchmarks locked with SLL, they can be easily unlocked using at most 67 iterations and 1070.85 seconds. On the other hand, when the Anti-SAT blocks are integrated, the SAT attack complexity increases exponentially with the key-size in the Anti-SAT block. The SAT attack fails to unlock the circuits within 10 hours when a 16-bit BA is integrated (as shown by the fifth data point w.r.t. x-axis).

3.6.3 Anti-SAT Block Obfuscation

In Section 3.5.3, we have shown that after obfuscation, the security of Anti-SAT against SAT attack would not be undermined. To validate this proof, we obfuscate the Anti-SAT block using LUTs and MUXes. Fig. 3.9 shows the SAT

[2]For SLL, the extra key-gates are inserted to the original circuit. For SLL(5%) + n-bit BA/OA, the extra key-gates are used in the Anti-SAT block and increasing the key-size also indicates increasing the input-size n because we construct the n-bit BA with key-size $k_{BA} = 2n$. In this experiment, we experiment the n-bit BA with $n = 8, 10, 12, 14, 16, 18, 20$. The key-sizes are shown in Table 3.4.

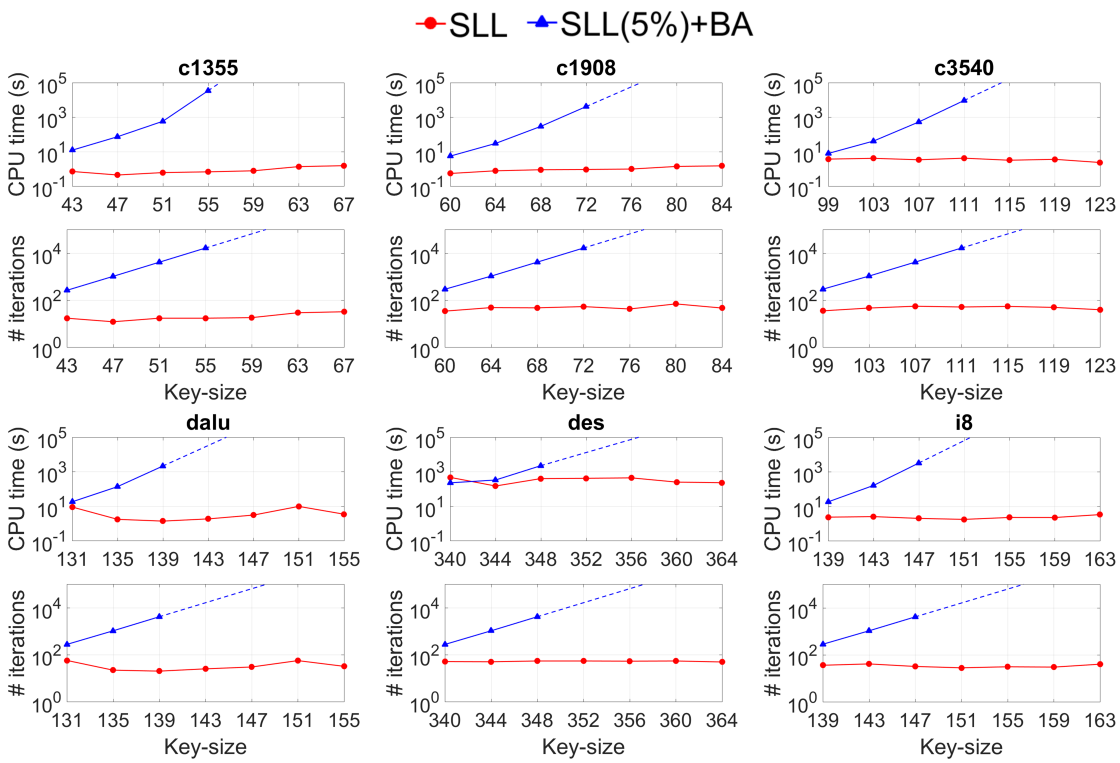
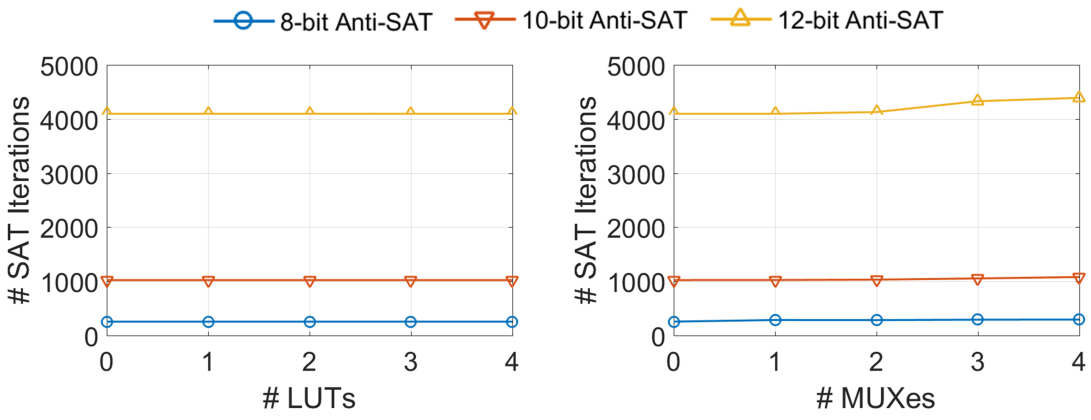


Figure 3.8: SAT attack results on 6 benchmarks with three logic locking configurations: SLL and SLL(5%) + n-bit BA. Timeout is 10 hours (3.6×10^4 s). The dashed lines are the curve fitting results when the SAT attack has time-outed after certain key-size.

attack results of c1355 circuit when obfuscation techniques are applied. Fig. 3.9(a) shows the result for LUT based design withholding technique. Here we increasingly replace the 2-input AND gates in g and $\overline{g}$ with LUTs and evaluate its impact on the SAT attack iteration. As seen, when the number of LUTs is increased, the SAT attack iteration remains to be 2^n, where n is the input-size of Anti-SAT ($n = 8, 10, 12$). Fig. 3.9(b) shows the results for MUX based wire entanglement technique. Here we use 2-input MUXes, where one input of MUX comes from the original



(a) (b)

Figure 3.9: SAT attack results of c1355 circuit when obfuscation techniques are applied: (a) design withholding and (b) wire entanglement. For both techniques, the number of SAT attack iterations required are $\geq 2^n$ after obfuscation, where n is the input-size of Anti-SAT.

circuit and the other input comes from the Anti-SAT block. As seen, the SAT attack iteration is $\geq 2^n$ for different choices of n. The number of iterations could be larger than 2^n because the MUXes enlarge the fan-in cones of the Anti-SAT block. These results confirm that the proposed obfuscation technique will not hamper the SAT attack resistance of Anti-SAT.

3.6.4 Performance Overhead

Different implementation of g and $\overline{g}$ will result in different overhead. In our experiments, we utilize an n-bit AND gate and an n-bit NAND gate to implement the function g and $\overline{g}$, each consists of $n-1$ AND2 gates. The estimated area for a n-bit BA is $4n$ additional gates. Since the number of SAT attack iteration

required is 2^n, a slight increase in area overhead of the Anti-SAT block can result in exponential increase in SAT attack complexity. To counter removal attacks, we investigate both the design withholding and entanglement techniques. These two obfuscation techniques will inevitably increase the performance overhead. For example, an n-input, m-output LUT would require $O(m \times 2^n)$ gates. The key-size for such LUT is $m \times 2^n$. An n-input, m-output wire entanglement module would require m number of n-input MUXes. The key-size for such module is $m \times \log(n)$. To reduce overhead, we can use multiple small LUTs/MUXes (with less inputs and outputs) to form large LUTs/MUXes, as suggested in [51]. However, we present it as the first unified obfuscation technique to make various removal attacks harder. A more light-weight solution may be explored in future research.

3.7 Related Work

3.7.1 SAT-attack Resilient Logic Locking

Recent years have seen an increasing number of research work on mitigating the SAT attack on logic locking. In [68], Yasin *et al.* proposed to add an AES circuit into a locked circuit which aims at increasing the SAT solving time. Although this approach is effective, the AES circuit leads to a significant performance and area overhead since a standard AES circuit implementation requires a large number of gates [69]. In [70], a technique called SARLock was proposed which can make the number of SAT attack iterations grow exponentially in key-size. SARLock is similar to Anti-SAT, however, it has been shown to be vulnerable to some variants of SAT

attack called double-DIP [42] or bypass attack [71]. On the contrary, these attacks cannot break Anti-SAT (when the combination of SLL and Anti-SAT locking is used), as analyzed in [42]. In [71], Xu *et al.* proposed a Binary Decision Diagram (BDD) based design technique to achieve exponential number of SAT attack iterations. However, the disadvantage of the BDD based technique is that it will result in a very significant area overhead, because the size of the BDD is almost always exponential in the key-size as shown in [71]. To increase the difficulty of SAT formulation, Shamsi [72] *et al.* proposed a cyclic logic locking technique which introduces non-reducible combinational loops to the locked circuit. However, the cyclic logic locking technique was shown vulnerable to a variant of SAT attack called Cyclic-SAT [43]. Besides conventional logic locking, a new set of locking techniques called parametric locking is proposed [33, 73]. The parametric locking techniques aim at obfuscating the parametric behavior of the circuit such as power, delay and reliability *etc.* For incorrect keys, the locked circuit will malfunction or have degraded performance.

3.7.2 SAT Attack on IC Camouflaging

IC camouflaging is a reverse-engineering prevention technique that hides a circuit's functionality with camouflaging cells. It has been shown that SAT attack can also be applied to recover the functionality of the camouflaging cells [74, 75]. To counter the SAT attack, various countermeasures have been proposed [76, 77], which aim at making the de-camouflaging effort exponentially harder in the number

of camouflaged gates.

3.8 Conclusion

In this chapter, we present a circuit block called Anti-SAT to mitigate the SAT attack on logic locking. We show that the iterations required by the SAT attack to reveal the correct key in the Anti-SAT block is an exponential function of the key-size in the Anti-SAT block. The Anti-SAT block is integrated to a locked circuit to increase its resistance to the SAT attack. A unified obfuscation technique has been proposed to protect the Anti-SAT block from removal attacks such as the SPS attack and the partitioning based attack. Overall, our proposed Anti-SAT based logic locking can effectively thwart the SAT attack and various removal attacks.

Chapter 4: Strong Anti-SAT: Secure Logic Locking for Neural Network Chips

4.1 Introduction

In recent years, neural network has made a significant impact on various fields such as computer vision, speech recognition, and natural language processing. As neural network models (neural models) are getting deeper and more complex, its computation is becoming time-consuming and resource-intensive. To address these problems, researchers have started to develop fast and low-power neural network chips that can support a range of neural models. Examples of such neural chips include DianNao [78], EIE [79], Eyeriss [80], TPU [81], *etc.*

Neural chips, following the trend of fabrication outsourcing, are inevitably subject to supply chain attacks by untrusted foundries. Logic locking techniques, as discussed in Chapter 2 and Chapter 3, can be applied to protect the neural network chip design from being pirated or overproduced. However, locking neural chips is not the same as locking conventional chips in two aspects. *Firstly, most neural network applications are inherently error-tolerant.* The classification accuracy of a neural network would be acceptable even when some of its underlying computations are in-

correct [82]. This can be exploited by an attacker who can just find an *approximate key (approx-key)* instead of a *correct key* to approximately unlock the chip such that it can output correctly for most inputs. The relaxed requirement makes attacks such as Approximate SAT (AppSAT) attack [41] applicable. *Secondly, most neural models are tune-able (e.g., by fine-tuning the weight values).* An attacker can adjust his own neural models to accommodate the approximately-unlocked (approx-unlocked) neural chips, hence further improving the classification accuracy.

In this work, we address these new challenges and propose a novel locking technique to protect neural chips against untrusted foundries. The neural chip can be loaded with a wide range of neural models. *The objective of locking the neural chip is to ensure that given any wrong key, the locked neural chip cannot function correctly, so any neural model running on such chip would have very low classification accuracy.* Neural chips are normally composed of a control unit, an arithmetic unit, a memory interface and an interconnect unit. These components can be locked to protect the neural chips. For simplicity, we target the locking of the *arithmetic unit (i.e., adders and multipliers)* in this work. However, some of our proposed techniques can be extended and applied to other components. The contributions of this work are as follows.

- We propose an attack methodology to investigate the vulnerability of state-of-the-art logic locking techniques [31] in securing neural chips. The proposed attack firstly utilizes the AppSAT attack [41] to find an approx-key which results in neural chips with very low error rate. Deploying neural models on

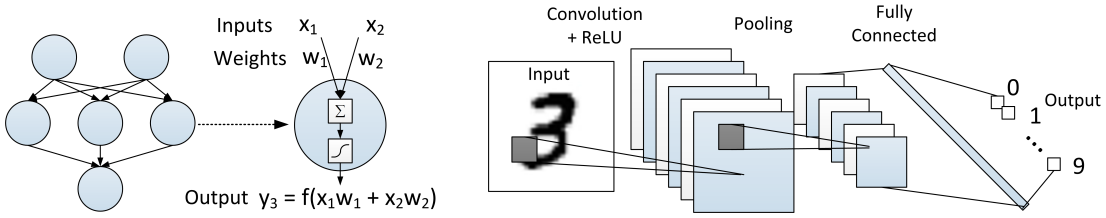
such hardware is shown to have only 7.2% reduction in classification accuracy. To further improve accuracy, we propose a neural-network fine-tuning technique which exploits the error characteristics of the approx-unlocked neural chips. Experiment results show that after fine-tuning, the accuracy loss for the deployed neural model is 0%.

- To counter this attack, we propose a secure locking scheme for neural chips which is based on a co-design of locking infrastructure and functional modules. We first propose an improved locking infrastructure (called *Strong Anti-SAT block*) based on Anti-SAT. We use a rigorous proof to derive a lower-bound of error rate for the Strong Anti-SAT block. Note that this error rate lower-bound holds for any wrong key. Thus, with correct configuration of Strong Anti-SAT, we can guarantee a high error rate for any key that's obtained by the AppSAT attack [41], hence making the attack ineffective. Furthermore, we investigate functional modules (*e.g.,* multipliers) which can be designed to be very hard for SAT solving. Hence, by appropriate design of the functional modules, we can ensure extremely long time for *exact* SAT attack [24] to find a correct key, thereby making it computationally impractical to correctly unlock the neural chips.

- Experimental results show that the proposed locking scheme can result in 1) 80% accuracy loss for neural models deployed on any approx-unlocked chip, and 2) a long time (*e.g.,* $\geq$ 1 year) for exact SAT attack to find the correct key.

4.2 Preliminary

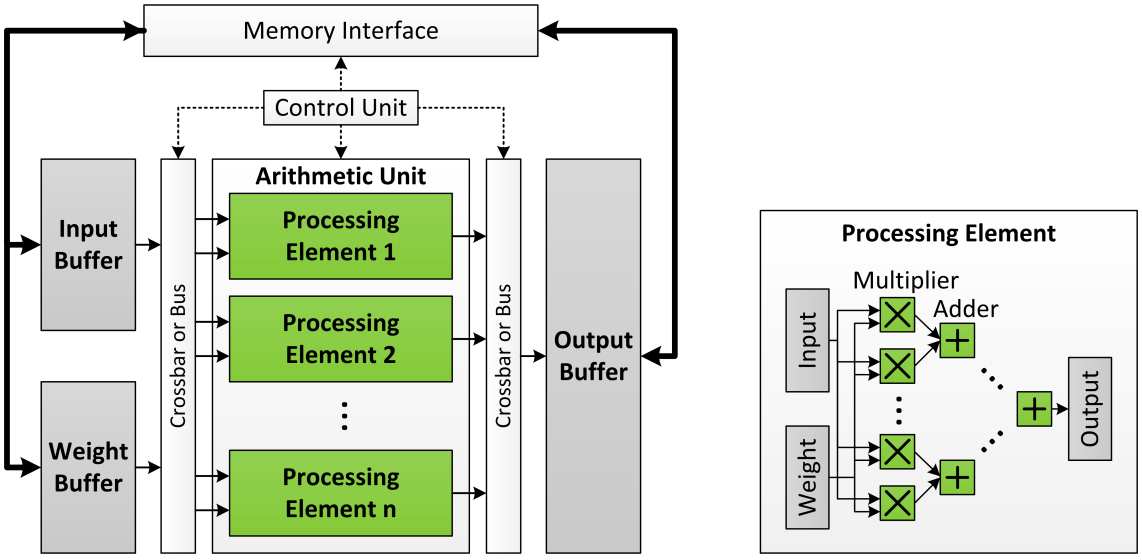
4.2.1 Neural Network Models



(a) (b)

Figure 4.1: Neural networks: (a) multi-layer perceptron; (b) convolutional neural network

Fig. 4.1(a) illustrates an example of an artificial neural network based on *multi-layer perceptron (MLP)*. A set of neurons are arranged in multiple layers and neurons in subsequent layers are fully connected. Each neuron takes the outputs of its previous layer as inputs, perform an inner-product between the inputs and a weight vector and pass the result into a nonlinear activation function to produce an output for this neuron. During *training*, a set of data-label pairs is used to tune the weights (using back-propagation algorithm) such that the prediction error for the training data is minimized. After the weights are learned, a forward-propagation of the neural network will output a predicted label given an input data. This forward-propagation is also known as *testing* or *inference*. A variant of MLP is called *Convolutional neural network (CNN)* [83]. CNN is a promising neural network model that has shown its effectiveness in various classification applications such as computer vision [83, 84].



(a) (b)

Figure 4.2: Neural chip: (a) core components; (b) processing element

Fig. 4.1(b) shows a simple CNN example. It normally consists of a few common building blocks, namely convolutional layers, pooling layers and fully-connected layers.

4.2.2 Neurnal Network Chips

In order to improve the flexibility to support more neural network models, many neural chip designs have been proposed, such as DianNao [78], EIE [79], and TPU [81]. These neural chips have emerged as a perfect platform for achieving fast and low-power computation for a wide range of neural network models. Fig. 4.2(a) shows some core components that are commonly shared among most neural chip designs, including a control unit, an arithmetic unit, buffers (for input, weight, and output), interconnect components, and a memory interface to external memory.

The arithmetic unit is an important component because it performs vector multiplication, the most fundamental operation in neural networks. For example, in AlexNet [84], more than 96% of weights are used in the fully-connected layers and these weights require a massive amount of vector multiplication operations. The arithmetic unit is formed by a group of processing elements (PEs), which is composed of an array of multipliers and an adder tree [78, 85] as shown in Fig. 4.2(b). To map a neural network design into the chip, a compiler will translate network specifications (*e.g.,* number of layers, layer types and layer sizes) into a set of instructions and store them into the control unit. Besides, pre-trained weights will normally be stored in an external memory.

4.2.3 Anti-SAT Based Logic Locking

To access advanced semiconductor technology, modern chips are increasingly outsourced to an offshore foundry for fabrication. The outsourced chip designs, however, are subject to attacks such as piracy, overproduction, and counterfeiting by the untrusted foundry. Neural chips, following this trend of fabrication outsourcing, are inevitably subject to these threats. One class of prevention technique is logic locking, which was introduced in Section 2.2.2. During design time, a circuit is locked by inserting a set of key-gates and key-inputs. The locked chip preserves the correct functionality only when a correct key is provided. Anti-SAT based logic locking is a sophisticated logic locking techniques, as discussed in Chapter 3. Such a logic locking would render the SAT attack [24] which attempts to learn

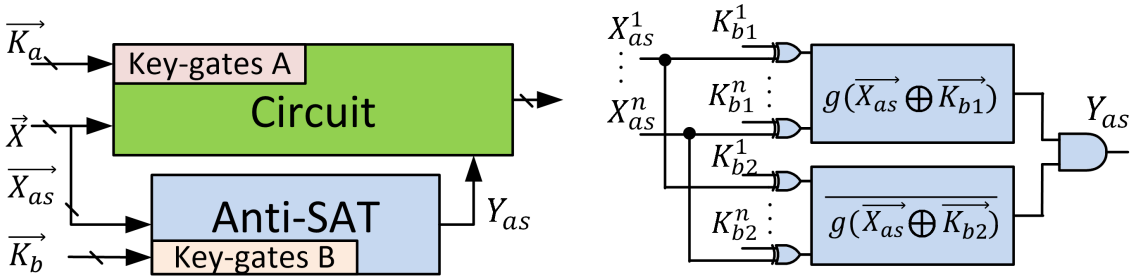
Table 4.1: Terminology list

Symbol	Definiton	Symbol	Definition
n	Input-size of original circuit	λ	SAT attack iterations
n_{as}	Input-size of Anti-SAT	ϵ	Error rate
$\vec{X}$	Inputs of original circuit	λ_0, ϵ_0	Lower-bounds of λ and ϵ
$\vec{X}_{as}$	Inputs of Anti-SAT	g_0	Mini-blocks of Strong Anti-SAT
Y_{as}	Output of Anti-SAT	n_0	Input-size of mini-block g_0
$\vec{K}_a$	Conventional keys	p_0	On-set size of mini-block g_0
$\vec{K}_b$	Anti-SAT keys	t	SAT solving time per iteration
$g, \overline{g}$	Logic blocks of Anti-SAT	T	Total SAT solving time
p	On-set size of logic block g	T_0	Lower-bound of T

the perfectly-correct key ineffective. Here we review the design of Anti-SAT and analyze its SAT-attack resistance and output corruptibility. The terminologies used are listed in Table 4.1.

4.2.3.1 Anti-SAT Configuration

Fig. 4.3(a) shows the overview of Anti-SAT based logic locking. The original circuit is locked with $\vec{K}_a$ (referred to as *conventional keys*) using conventional locking techniques such as [22, 47]. Besides, an Anti-SAT block is attached to the locked circuit. The Anti-SAT has inputs $\vec{X}_{as}$ which are connected to some primary inputs $\vec{X}$ of the original circuit. The Anti-SAT output Y_{as} is connected to an internal wire of the original circuit using an XOR gate. A set of keys $\vec{K}_b$ (referred to as *Anti-SAT*



(a) (b)

Figure 4.3: Anti-SAT based logic locking: (a) overview; (b) Anti-SAT block

keys) are inserted at the Anti-SAT block. Fig. 4.3(b) shows the detail of Anti-SAT block. It is composed of two logic blocks g and $\overline{g}$ which have complementary functionalities. The Anti-SAT keys $\vec{K}_b = (\vec{K}_{b1}, \vec{K}_{b2})$ are inserted at each input of g and $\overline{g}$. The outputs of g and $\overline{g}$ are fed into a 2-input AND gate to produce the Anti-SAT output Y_{as}, which makes $Y_{as} = g(\vec{X}_{as} \oplus \vec{K}_{b1}) \wedge \overline{g(\vec{X}_{as} \oplus \vec{K}_{b2})}$. Given a wrong key, Y_{as} will output 1 and inject faults into the circuit. The logic block g in Anti-SAT has input-size n_{as} and on-set size p, where on-size size is the number of input patterns that can make function g output one.

4.2.3.2 SAT-Attack Resilience

According to Theorem 3.4.1 in Chapter 3, the number of SAT attack iterations λ required to unlock the Anti-SAT block is bounded by λ_0:

$$\lambda \geq \lambda_0 = \frac{2^{2n_{as}} - 2^{n_{as}}}{p \times (2^{n_{as}} - p)} \tag{4.1}$$

As seen in Eq. (4.1), when $p \to 1$ or $p \to 2^{n_{as}} - 1$, we have

$$\lambda \geq \lambda_0 = 2^{n_{as}} \tag{4.2}$$

which shows the exponential increase of SAT attack complexity. When $p = 1$, g could simply be an n_{as}-bit AND gate. In the remaining of this paper, we by default assume Anti-SAT blocks have $p = 1$ as it can ensure exponential increase of λ_0.

4.2.3.3 Output Corruptibility (Error Rate)

Here we analyze the output corruptibility of the Anti-SAT block. According to [31], when $p = 1$, for any wrong key into $\vec{K}_b$, only 1 out of $2^{n_{as}}$ input patterns (w.r.t $\vec{X}_{as}$) can make incorrect output $Y_{as} = 1$. Let's denote such corrupted input pattern as $\vec{X}^w_{as}$. As a result, the error rate of the Anti-SAT block is $\frac{1}{2^{n_{as}}}$ for any wrong key. Now let us analyze the error rate of a circuit that's comprised of the Anti-SAT block. Assume the circuit has n inputs and n_{as} out of them are used for the Anti-SAT inputs. The corrupted input patterns would be such that 1) the n_{as} primary inputs that are connected to the Anti-SAT have values as $\vec{X}^w_{as}$; and 2) the other $n - n_{as}$ primary inputs can take any values. Thus, we have $2^{(n-n_{as})}$ corrupted input patterns (w.r.t $\vec{X}$) that can make incorrect output $Y_{as} = 1$, so the error rate of the locked circuit is $\epsilon = \frac{2^{(n-n_{as})}}{2^n} = \frac{1}{2^{n_{as}}}$. As seen, although reducing n_{as} can increase ϵ, it inevitably reduces λ as illustrated in Eq. (4.2).

4.2.4 AppSAT Attack

In [41], Shamsi *et al.* proposed an approximate SAT (AppSAT) attack which targets locking techniques such as Anti-SAT [31] and SARLock [70]. The AppSAT attack assumes the same attack model as the one used for SAT attack (see Sec-

tion 3.2). It extends the SAT attack by adding an early termination condition to avoid taking an exponential number of iterations to find a correct key. When the early termination condition is satisfied, the AppSAT terminates and outputs the approx-key which can match all already found distinguishing inputs to their correct outputs. As shown in [41], after a few iterations, the AppSAT attack can decipher an approx-key which has correct conventional keys $\vec{K}_a$ but incorrect Anti-SAT keys $\vec{K}_b$ (see Fig. 4.3(a)). This is because that the keys $\vec{K}_a$ inserted at the original netlist have high output-corruptibility and each distinguishing input/output pair can eliminate a large number of wrong keys w.r.t. the $\vec{K}_a$. As a result, the $\vec{K}_a$ are gradually learned in the first few iterations, hence leaving the circuit that's only locked with $\vec{K}_b$. Since the Anti-SAT keys $\vec{K}_b$ have very low corruptibility as discussed in Section 4.2.3.3, the approx-key can effectively de-obfuscate most of the correct functionality.

4.3 Attack on Locked Neural Chips

In this section, we investigate the security of neural chips when its arithmetic units (adders and multipliers) are locked with Anti-SAT based logic locking [31]. As discussed in Section 4.2.3, this locking technique utilizes a combination of conventional logic locking [22, 47] (with conventional keys $\vec{K}_a$) and Anti-SAT block (with Anti-SAT keys $\vec{K}_b$) which represents among the strongest defenses to SAT attack. Such locking technique would render the SAT attack which attempts to learn the correct key ineffective. However, we propose an attack methodology to show that

such sophisticated locking scheme is not secure for neural chips.

4.3.1 Attack Model

The proposed attack methodology consists of two steps, which exploits 1) the AppSAT attacks [41] and 2) neural model fine-tuning in effective ways. In the AppSAT attack step, we assume that the attacker is an untrusted foundry and its objective is to obtain an approx-key to approximately unlock the chip. In the neural-model fine-tuning step, the attacker can be the untrusted foundry or an end-user who is in collusion with the untrusted foundry. The attacker's objective is to deploy a neural model to the approx-unlocked neural chips. To achieve a high classification accuracy, the attacker wants to tune his neural model to adapt to the approx-unlocked neural chip.

Neural chips are normally composed of a control unit, an arithmetic unit, a memory interface and an interconnect unit. These components can be locked to protect the neural chips. For simplicity, we target the locking of the *arithmetic unit (i.e., adders and multipliers)* in this work. The adders and multipliers perform the most fundamental and frequent operation of the neural network, *i.e.,* vector multiplication. Hence, as a natural choice, we assume that the attacker inserts the key-gates into the adders/multipliers to lock the core functionality of the neural chip. However, some of our proposed techniques can be extended and applied to other components.

4.3.2 Step 1: Approx-unlocking Neural Chips

Different from conventional chips, the neural chips are designed for neural network applications that are normally error-tolerant. By exploiting the error-tolerant nature, an attacker only needs to obtain an approx-key which can unlock most (but not all) of the correct functionality for the neural chips. This relaxed requirement makes attacks such as AppSAT [41] applicable. As shown in [41] and in this work, after a few iterations, the AppSAT attack can decipher the conventional keys $\vec{K}_a$ but not the Anti-SAT keys $\vec{K}_b$. This is because that the output corruptibility of $\vec{K}_a$ is much higher than that of $\vec{K}_b$. Therefore, as iteration progresses, $\vec{K}_a$ is gradually learned, thereby leaving a circuit that's only locked using the Anti-SAT block with keys $\vec{K}_b$. As analyzed in Section 4.2.3.3, the error rate of such locking scheme is $\epsilon = \frac{1}{2^{n_{as}}}$. For typical n_{as}, this error rate may be very small thereby resulting in an approx-unlocked circuit which is correct for most input patterns. The AppSAT attack results on locked adders/multipliers will be shown in Section 4.3.4.2.

4.3.3 Step 2: Neural-network Fine-tuning

Now let us suppose on such approx-unlocked neural chips, an attacker wishes to deploy a neural model. The attacker may just deploy his model on the approx-unlocked chips directly and tolerate a rather humble degradation in quality. Or he may exploit knowledge of the error characteristics of the approx-unlocked chips and tune the neural model to avoid high error scenarios. Here we assume the attacker has the ability to fine-tune his own neural model and schedule its computation

to specific arithmetic modules in the neural chips. The objective of the attack is to reduce the number of incorrect computations based on the observation that the classification accuracy would increase as the computation error decreases. To achieve this objective, we propose a *neural network fine-tuning technique* which consists of three steps: *error profiling*, *weight tuning*, and *adder-input shifting*.

4.3.3.1 Error Profiling

After acquiring the approx-unlocked neural chip, the attacker can first profile the error distribution for both the approx-unlocked multipliers and adders. The error profile represents the numerical distance between incorrect outputs and correct outputs for given input operands. If enumerating all possible I/Os is impractical, an attacker can randomly sample a large subset of I/Os to estimate the error profile.

4.3.3.2 Weight Tuning

In neural-network models, inputs of neurons are multiplied with neural weights, which are computed by the multipliers in the arithmetic unit. The pre-trained weight values are known and tunable by the attackers. Recent studies have found that minor weight changes (*e.g.,* weight quantization or fine-tuning) would not affect the classification accuracy of the neural model, which has been widely exploited for various hardware optimization [86]. In this work, we fine-tune the weights for another purpose, which is to reduce the number of computation error thereby improving classification accuracy. To reduce the error, an attacker can tune a weight to an-

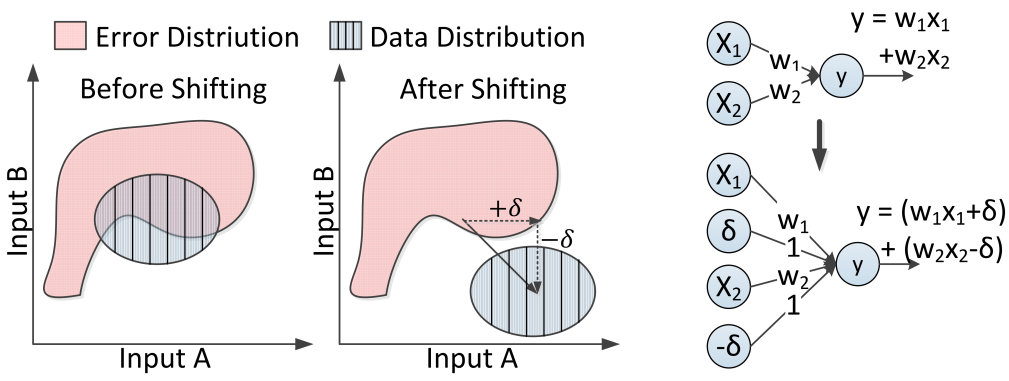
other value in its vicinity such that the new weight will have less multiplication error. We formulate the problem as follows.

$$\begin{aligned} \underset{w^m_{i,j}}{\text{minimize}} \quad & \sum_{d \in D_{i,j}} error(w^m_{i,j} \times d) \\ \text{subject to} \quad & |w^m_{i,j} - w_{i,j}| \leq \sigma \end{aligned} \tag{4.3}$$

Here $w_{i,j}$ is the j-th pre-trained weight at i-th layer, $w^m_{i,j}$ is the modified weight, σ is the limit for weight tuning, $error(w^m_{i,j} \times d)$ is the absolute numerical error for multiplying the weight $w^m_{i,j}$ with data d, and $D_{i,j}$ is the typical set of values that the weight $w_{i,j}$ is multiplied with. For each weight, $D_{i,j}$ is either the set of training data values or the intermediate output values of neurons, so it can be easily estimated based on a set of training data. In our implementation, the weights are fine-tuned layer by layer. After updating all the weights at layer i based on $D_{i,j}$, we compute the neuron outputs which are used in the next layer as $D_{i+1,j}$. We repeat the weight tuning process until every layer is tuned. Since weight tuning deviates the weights from the ones learned from training data, a large change in weights might instead decrease accuracy. Therefore, in this work, σ is set to be a very small value. For example, for a weight which is represented as a fixed-point number with q bits for its fraction part, we set $\sigma = 2^{-q}$.

4.3.3.3 Adder-input Shifting

Unlike multipliers where one input is a tunable weight, the adders accept intermediate values as inputs that are not directly tunable. To reduce the error, we propose an adder-input shifting technique. The basic idea is shown in Fig. 4.4(a).



(a) (b)

Figure 4.4: Adder-input shifting: (a) illustration; (b) implementation

Recall that the attacker has an error profile of the approx-unlocked adder which represents the error distribution over two adder inputs (red region in Fig. 4.4(a)). Besides, because the attacker has the ability to schedule the computation to the arithmetic operator, he can obtain an input profile for adder based on a set of training data, which represents the data distribution over two adder inputs (blue region in Fig. 4.4(a)). Based on these two distributions, the attacker can shift two adder inputs by a constant value δ in opposite directions:

$$A + B \rightarrow (A + \delta) + (B - \delta) \tag{4.4}$$

such that the new data distribution for $(A + \delta)$ and $(B - \delta)$ will be away from the error distribution. To implement the adder-input shifting, the attacker can modify his neural network as shown in Fig. 4.4(b). Each add operation is now realized by three add operations as shown in Eq. (4.4). Besides, he needs to modify the data flow scheduling to ensure correct order of three add operations.

4.3.4 Attack Results

4.3.4.1 Experiment Setup

In this experiment, the neural chip under attack uses 16-bit fixed-point adders and multipliers (7 bits for fractional part). Such arithmetic operators can preserve high classification accuracy while saving power and area, which is widely used in modern neural chip designs [78–80]. The adders and multipliers are locked using Anti-SAT based logic locking. The original circuit is locked with keys $\vec{K}_a$ which has key-size $|\vec{K}_a| = 5\% \times$ #Gates of the original circuit. Besides, a 16-input obfuscated Anti-SAT block (with $p = 1$) was attached to the locked circuit, which has key-size $|\vec{K}_B| = 4 \times 16 = 64$. The 16 inputs of Anti-SAT are randomly connected to 16 out of 32 primary inputs of the adder/multiplier. Such locking scheme can cost the SAT attack more than 1 year to find the correct key [31].

4.3.4.2 Attack Result 1: Approx-unlocking

As discussed in Section 4.3.2, in the first step of our proposed attack methodology, we utilize the AppSAT attack to find an approx-key to de-obfuscate most of the correct functionality. Fig. 4.5 shows the AppSAT attack progress. As seen, for both the adder and the multiplier, the error rate ϵ starts at 100% for a random key. However, ϵ drops dramatically during the first 30 iterations and it continues to decrease gradually as the attack proceeds. These results show the efficiency of AppSAT attack on finding an approx-key which can achieve low error rate. In this

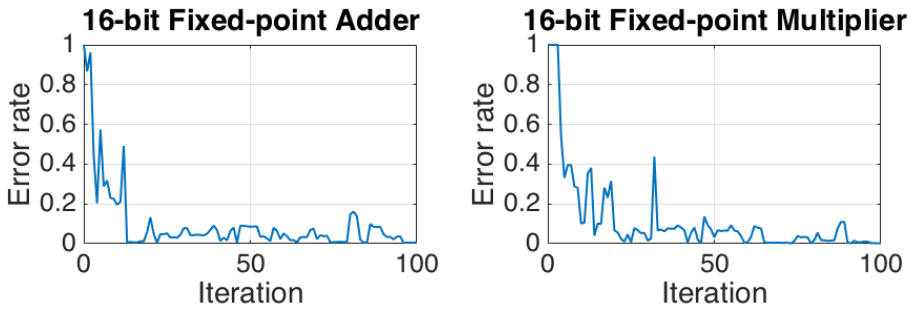


Figure 4.5: Error rate v.s. SAT attack iteration. The error rate is estimated using 10000 random input patterns.

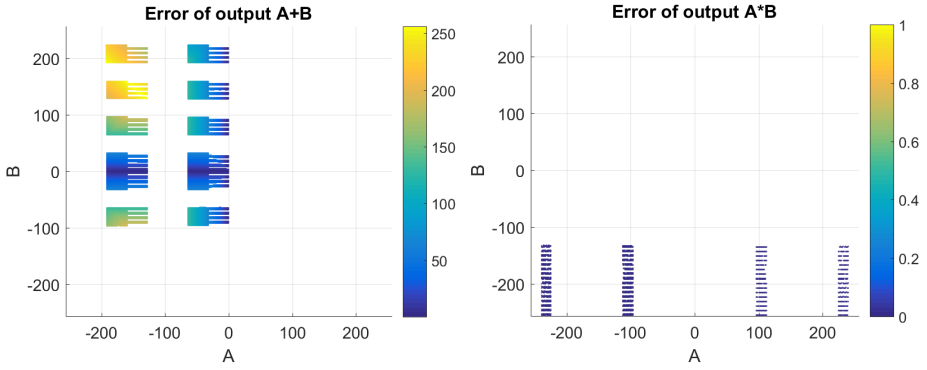


Figure 4.6: Error profiles of approx-unlocked adder/multiplier

experiment, the AppSAT attack is terminated at 5000 iterations and it outputs an approx-key denoted as $\vec{K}_{App}$. Fig. 4.6 illustrates the error distribution for the approx-unlocked adder/multiplier with $\vec{K}_{App}$. The colored region shows the error distribution over the input space (blank space means no error). Also, the color bar on the right of each figure shows the absolute numerical distance between correct and incorrect outputs.

Five neural-network models (as listed in Table 4.2) are used to evaluate the performance of the approx-unlocked chip. The neural network is simulated using Ristretto [86], a neural network framework based on Caffe. To simulate the arith-

Table 4.2: Neural network benchmarks

Benchmark	**#Label**	**#Test Data**	**Model**
MNIST	10	10000	LeNet
SVHN	10	10000	CIFAR10_Full
CIFAR10	10	10000	CIFAR10_Quick
ILSVRC-2012	1000	500	CaffeNet
Oxford102	102	1000	CaffeNet

metic error, error profiles of the approx-unlocked adder and multiplier are embedded into the simulation tool. Table 4.3 shows the accuracy of 5 models running on a neural chip that's unlocked with a correct key $\vec{K}_C$ and an approx-key $\vec{K}_{App}$. We can see that the relative accuracy loss of an approx-unlocked neural chip is only 7.20% on average. This is due to the low error rate of the approx-unlocked circuits as well as the inherent error-tolerant nature of neural networks.

4.3.4.3 Attack Result 2: Neural-network Fine-tuning

From Table 4.3, we can see that the accuracy decreases more for large benchmark such as ILSVRC-2012. Here we evaluate the effectiveness of our neural-network fine-tuning technique in further improving the accuracy. For weight tuning, we set $\sigma = 2^{-q} = 2^{-7}$. For adder input shifting, we first plot the adder input distribution for 5 benchmarks based on 100 training data, as shown in Fig. 4.7 . Based on Fig. 4.6 (left) and Fig. 4.7, we select the shift distance δ to be 50 as it can shift the input distribution away from the error region. As shown in the last column of Table 4.3,

Table 4.3: Accuracy of neural models deployed on a neural chip that's unlocked with a correct key $\vec{K}_C$ and an approx-key $\vec{K}_{App}$ (without/with neural network fine-tuning)

Benchmark	**With $\vec{K}_C$**	**With $\vec{K}_{App}$**		**With $\vec{K}_{App}$ + Fine-tuning**	
	Accuracy	**Accuracy**	**Accuracy Loss**	**Accuracy**	**Accuracy Loss**
MNIST	99%	98.51%	0.49%	99%	0%
SVHN	93.51%	92.64%	0.93%	93.51%	0%
CIFAR10	75.37%	69.75%	7.46%	75.37%	0%
ILSVRC-2012	41.40%	33.20%	19.81%	41.40%	0%
Oxford102	87.60%	81.20%	7.31%	87.60%	0%
Average	-	-	7.20%	-	0%

fine-tuning improves the accuracy of all 5 benchmarks. The accuracy loss is reduced to 0% for all benchmarks.

These results together demonstrate that an attacker can use the proposed attack methodology to approx-unlock a neural chip and fine-tune his neural network models to accommodate the approx-unlocked neural chips so as to achieve better accuracy.

4.4 Secure Locking for Neural Chips

The attack results in Section 4.3.4 illustrate that AppSAT can easily decipher an approx-key to obtain approx-unlocked adders/multipliers with low error rate. The low error rate of approx-unlocked adders/multipliers leads to a humble degra-

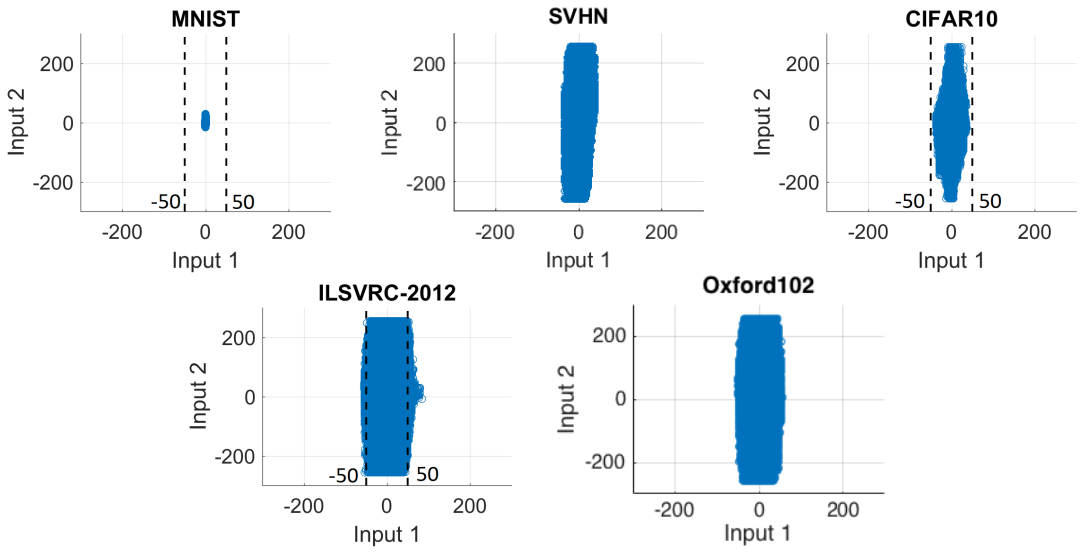


Figure 4.7: Adder input distribution for 5 benchmarks

dation in classification accuracy for neural models that are deployed on the approx-unlocked chips. In addition, the low error rate also facilitates the fine-tuning step, which makes the approx-unlocked neural chip work better for fine-tuned models. To thwart the proposed attack scheme, we need to ensure that the error rate of an approx-unlocked chip is sufficiently high. One possible approach is to increase the output corruptibility of the Anti-SAT block (*e.g.,* by reducing n_{as}). However, if n_{as} is reduced, the complexity of exact SAT attack for finding a correct key would surely come down, as discussed in Section 4.2.3.3. Hence there are two competing objectives:

1. AppSAT type attacks should have high error rate to ensure a sufficient quality degradation in application level;

2. The complexity of exact SAT attack to determine the correct key should still be very high.

In this section, we propose a secure locking scheme which aims at achieving these two objectives simultaneously. The proposed locking technique is based on a co-design of the locking infrastructure (*i.e.,* a modified Anti-SAT block) and the functional modules (*i.e.,* the multipliers).

4.4.1 Strong Anti-SAT: Increasing Error Rate

4.4.1.1 Strong Anti-SAT Configuration

To ensure that AppSAT type attacks should have a high error rate, we propose a modified Anti-SAT block (referred to as *Strong Anti-SAT*) which makes two modifications to existing Anti-SAT block. Firstly, the Strong Anti-SAT block decomposes the logic block g of Anti-SAT (shown in Fig. 4.3(b)) into m mini-blocks g_0, as shown in Fig. 4.8. Each mini-block g_0 has n_0 inputs and on-set size p_0 ($1 \leq p_0 \leq 2^{n_0} - 1$). We use $\vec{K}_{b1}^{j}$ and $\vec{K}_{b2}^{j}$ to denote the portion of key-inputs into the j-th mini-block of logic block g and $\overline{g}$, where $1 \leq j \leq m$. Also, we use $\vec{X}^{j}$ to denote the portion of Anti-SAT inputs into the j-th mini-block of logic block g and $\overline{g}$. The outputs of the mini-blocks in g and $\overline{g}$ are denoted as Y_{b1}^{j} and Y_{b2}^{j}, respectively. We enforce the g_0 to be a function that satisfy the following condition: if $\vec{K}_{b1}^{j} \neq \vec{K}_{b2}^{j}$, then there must exist an input $\vec{X}^{j}$ such that $g_0(\vec{X}^{j} \oplus \vec{K}_{b1}^{j}) \neq g_0(\vec{X}^{j} \oplus \vec{K}_{b2}^{j})$. This means that a mismatch between two keys must make the function g_0 output differently for some input patterns. This condition ensures that a key for Strong Anti-SAT $\vec{K} = (\vec{K}_{b1}, \vec{K}_{b2})$ is a wrong key if $\vec{K}_{b1} \neq \vec{K}_{b2}$.

The second modification in the Strong Anti-SAT is that, when locking a circuit

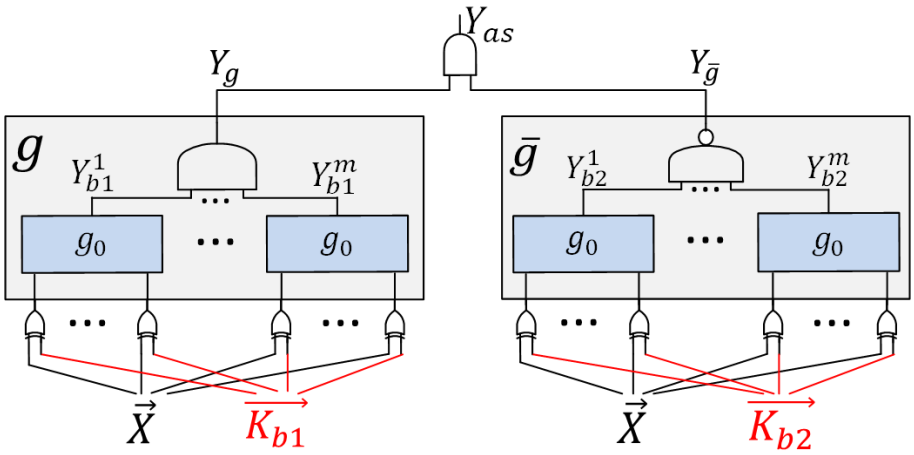


Figure 4.8: An n-input Strong Anti-SAT block. Each mini-block g_0 has n_0 inputs and on-set size p_0.

with n primary inputs, we enforce the Strong Anti-SAT block to have $n_{as} = n$ inputs. Such enforcement ensures that every primary input can affect the Anti-SAT block, thereby making the error more uniformly distributed across the input space. In the remaining of this chapter, we assume $n_{as} = n$ and use $\vec{X}$ and n to denote the inputs and input-size of both the original circuit and the Strong Anti-SAT block. So for n-input Strong Anti-SAT, we have $m = n/n_0$ mini-blocks in g.

4.4.1.2 Error Rate Analysis

In this section, we will derive an error rate lower-bound ϵ_0 of the Strong Anti-SAT block. Such ϵ_0 holds for any wrong key and it can be tuned to a large value, so AppSAT type attacks will not be able to decipher a "good" enough approx-key.

Theorem 4.4.1. The error rate of an n-input Strong Anti-SAT block (in Fig. 4.8) is $\epsilon \geq \epsilon_0 = \frac{p_0^{(n/n_0-1)}}{2^n}$ for any wrong key.

Proof. We prove this theorem by first deriving the form of wrong keys and corrupted input patterns. Then, for any given wrong key, we analyze the number of ways to construct a corrupted input pattern and then compute the error rate.

1. Firstly, we recall that the Strong Anti-SAT preserves the constant-output property as the Anti-SAT. In other words, given a correct key, the output Y_{as} in Fig. 4.8 should always output 0 for all inputs $\vec{X}$. On the other hand, given a wrong key, Y_{as} can output 1 for some input patterns. Note that a key for Strong Anti-SAT $\vec{K}_b = (\vec{K}_{b1}, \vec{K}_{b2})$ is a wrong key if $\vec{K}_{b1} \neq \vec{K}_{b2}$, as discussed earlier.

2. Now, we fix $\vec{K}_b = (\vec{K}_{b1}, \vec{K}_{b2})$ to some wrong keys which satisfy $\vec{K}_{b1} \neq \vec{K}_{b2}$, and analyze the number of possible assignments to inputs $\vec{X}$ that can produce incorrect output $Y_{as} = 1$. Without loss of generality, let's assume that these two key-input vectors $\vec{K}_{b1}$ and $\vec{K}_{b2}$ differ in the portion that's connected to the first mini-block, *i.e.,* the wrong key has $\vec{K}^1_{b1} \neq \vec{K}^1_{b2}$ and all other key-bits can be either 0 or 1 [1].

3. As shown in Fig. 4.8, to make $Y_{as} = 1$, the Anti-SAT block should have $Y_g = 1$ and $Y_{\overline{g}} = 1$, which requires that

$$\forall j \in [1, m], Y^j_{b1} = g_0(\vec{X}^j \oplus \vec{K}^j_{b1}) = 1 \tag{4.5}$$

[1]Note that we can repeat the analysis for m groups of wrong keys, where the j-th group satisfies that $\vec{K}^j_{b1} \neq \vec{K}^j_{b2}$ while all other key-bits can be 0 or 1. Here we only focus on the first group of wrong key for the simplicity of explanation.

and

$$\exists j \in [1, m], Y_{b2}^{j} = g_0(\vec{X}^j \oplus \vec{K}_{b2}^{j}) = 0 \tag{4.6}$$

The corrupted input patterns for $\vec{X} = (\vec{X}^1, ..., \vec{X}^m)$ must be those that satisfy both Eq. (4.5) and Eq. (4.6).

4. We now show that for the fixed wrong key as assumed in step 3, the number of ways to construct a corrupted input pattern based on conditions Eq. (4.5) and Eq. (4.6) would be at least $p_0^{m-1} = p_0^{n/n_0-1}$. Since we have assumed a wrong key with $\vec{K}_{b1}^{1} \neq \vec{K}_{b2}^{1}$, we can know that the inputs to the first mini-block in g (which is $\vec{X}^1 \oplus \vec{K}_{b1}^{1}$) and the inputs to first mini-block in $\overline{g}$ (which is $\vec{X}^1 \oplus \vec{K}_{b2}^{1}$) are different. Given $\vec{K}_{b1}^{1} \neq \vec{K}_{b2}^{1}$ and $\vec{X}^1 \oplus \vec{K}_{b1}^{1} \neq \vec{X}^1 \oplus \vec{K}_{b2}^{1}$, there must exist at least one way to construct $\vec{X}^1$ such that the $Y_{b1}^{1} = g_0(\vec{X}^1 \oplus \vec{K}_{b1}^{1}) = 1$ and $Y_{b2}^{1} = g_0(\vec{X}^1 \oplus \vec{K}_{b2}^{1}) = 0$ [2]. With this, the condition in Eq. (4.6) is satisfied. The only remaining conditions that need to satisfied is $Y_{b1}^{j} = 1,\ j \in [2, m]$ in Eq. (4.5). We want to compute the number of ways to construct the other inputs $\vec{X}^j,\ j \in [2, m]$ to satisfy the remaining conditions in Eq. (4.5). Since we assume the mini-block g_0 has on-set size p_0, for any key into g_0, there exists p_0 ways to select an input assignment for $\vec{X}^i$ to ensure $Y_{b1}^{i} = 1$. Therefore, each portion of inputs $\vec{X}^j$ $(j \in [2, m])$ can be selected in p_0 ways. Since $\vec{X}^1$ can be selected in at least one way and each $\vec{X}^j$ $(j \in [2, m])$ can be selected in p_0 ways, the total number of ways to construct a corrupted input pattern

[2] Note that in Section 4.4.1.1, we have enforced that, if $\vec{K}_{b1}^{j} \neq \vec{K}_{b2}^{j}$, then there must exist an input $\vec{X}^j$ such that $g_0(\vec{X}^j \oplus \vec{K}_{b1}^{j}) \neq g_0(\vec{X}^j \oplus \vec{K}_{b2}^{j})$.

would be at least $p_0^{m-1} = p_0^{n/n_0-1}$.

5. We can repeat above analysis for any other $m-1$ group of wrong keys, where the j-th group satisfies $\vec{K}_{b1}^{j} \neq \vec{K}_{b2}^{j}$, $j \in [2, m]$. Each group of wrong keys can corrupt p_0^{n/n_0-1} input patterns, however these corrupted input patterns might not be mutually exclusive. Therefore, we conclude that for any wrong key, the number of corrupted input patterns is $\geq p_0^{(n/n_0-1)}$.

6. Since the total number of input patterns is 2^n, the error rate for any wrong key is

$$\epsilon \geq \epsilon_0 = \frac{p_0^{(n/n_0-1)}}{2^n} \tag{4.7}$$

Hence proved. ■

Theorem 4.4.1 provides a rigorous error rate lower-bound ϵ_0 of the Strong Anti-SAT block for any wrong key. Based on Theorem 4.4.1, we can design an n-input Strong Anti-SAT block with a guarantee of high error rate by tuning n_0 and p_0. Hence AppSAT will never be able to find a key whose error rate is smaller.

4.4.2 Multiplier Design: Increasing SAT Solving Time Per Iteration

Since the logic block g in the Strong Anti-SAT block has input-size $n_{as} = n$ and on-set size $p = p_0^m = p_0^{(n/n_0)}$, the number of SAT attack iterations for finding a correct key becomes

$$\lambda \geq \lambda_0 = \frac{2^{2n} - 2^n}{p_0^{(n/n_0)} \times (2^n - p_0^{(n/n_0)})} = \Theta\Big(\frac{2^n}{p_0^{(n/n_0)}}\Big) \tag{4.8}$$

As can be seen in Eq. (4.7) and Eq. (4.8), for $p_0 = 1$ the error rate and the SAT iterations are exactly those for Anti-SAT. Now tuning n_0 and p_0 can help increase the error rate but as illustrated this will impact the number of SAT iterations. One way to counter this degradation in the number of SAT iterations is to investigate the design of functional units which are inherently very slow in completing each SAT iteration. Note that SAT is an NP-Complete problem. Hence by appropriate design of the functional units, we may be able to ensure that the each SAT iteration takes a long time to solve. Hence running $\Theta(\frac{2^n}{p_0^{(n/n_0)}})$ iterations, even though smaller than 2^n, is actually practically impossible. To maintain a large total SAT solving time, we propose to increase the size of multipliers in the arithmetic units. Modern neural chips [78–80] use 16-bit (or even smaller) fixed-point multiplier to save power and area. However, such a small multiplier is easily solvable by SAT solvers. A large multiplier, on the contrary, has been considered as a circuit that's hard for SAT solvers based on the conjecture that factoring large integers is difficult [87]. Experiments in [88] showed that the SAT solving time for factoring integers could increase exponentially in the operand width w. As will be shown in Section 4.5, the SAT solving time per iteration will also increase exponentially in w. Thus, we can maintain a high SAT attack complexity by using a larger multiplier [3].

[3]Since we enforce that the Strong Anti-SAT has the same number of inputs as the locked circuit ($n_{as} = n$), the size of the multiplier and the Strong Anti-SAT are both determined by n. Increasing n will increase the size of multipliers and Strong Anti-SAT.

4.4.3 Summary of Attack Mitigation

In summary, our attack mitigation is based on a co-design of the Strong Anti-SAT block and the multiplier modules. Firstly, appropriate choice of n_0 and p_0 of the Strong Anti-SAT can be designed to achieve a desired error rate lower-bound. Also, by enforcing the Strong Anti-SAT inputs to connect to all primary inputs $\vec{X}_{as} = \vec{X}$, we ensure that every primary input can affect the Anti-SAT block, thereby making the error more uniformly distributed across the input space. Secondly, a large multiplier design is utilized to increase the SAT solving time per iteration. Together these would 1) counter the AppSAT attack and the fine-tuning attack because the error rate of locked arithmetic units is sufficiently high for any approx-key, and 2) counter the exact SAT attack because the total SAT solving time for finding the correct key is extremely long (*e.g.*, $\geq$ 1 year).

4.5 Experiments and Results

This section shows the experimental results of our proposed secure locking technique for neural chips.

4.5.1 Validation of Analytical Lower Bounds

In Section 4.4.1, we discuss the configuration of a Strong Anti-SAT block and analyze the lower-bounds for error rate ϵ_0 and the number of required SAT iteration λ_0 as shown in Eq. (4.7) and Eq. (4.8). To validate the correctness of two analytical lower-bounds, we designed a 16-input Strong Anti-SAT block with different (n_0, p_0)

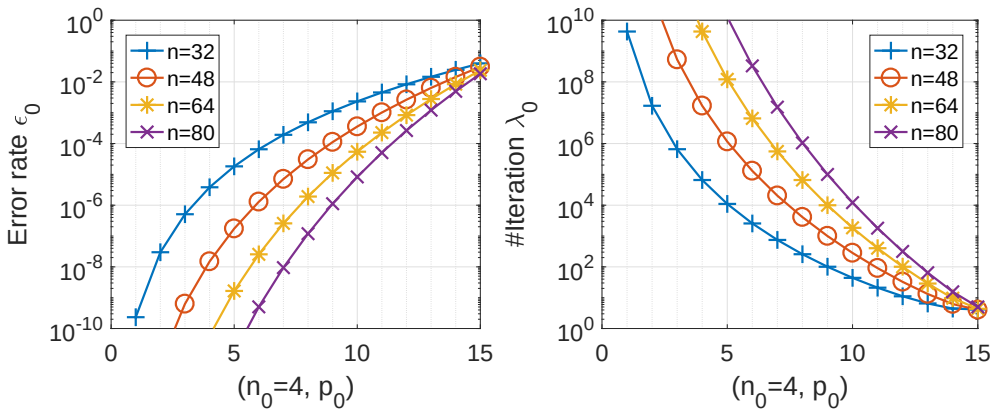
Table 4.4: Error rate ϵ and the number of SAT iterations λ of a 16-input Strong Anti-SAT block with different (n_0, p_0). ϵ_0 and λ_0 are the analytical lower-bounds. ϵ is the experimental error rate obtained by simulating all 2^{16} input patterns. λ is the experimental number of iteration required by SAT attack.

n=16						
(n_0, p_0)	(2,1)	(2,3)	(4,7)	(4,8)	(4,13)	(4,15)
ϵ_0	1.53E-05	3.34E-02	5.23E-03	7.81E-03	3.35E-02	5.15E-02
ϵ	1.53E-05	9.13E-02	3.57E-02	3.66E-02	2.16E-01	1.86E-01
λ_0	65536	12	29	18	5	6
λ	65536	364	656	1334	344	187

and test their actual error rate and SAT iterations. The result is shown in Table 4.4. As seen, for different (n_0, p_0), we always have $\epsilon \geq \epsilon_0$ and $\lambda \geq \lambda_0$, which validates the correctness of our analysis for the lower-bounds ϵ_0 and λ_0.

4.5.2 Error Rate and Accuracy Loss

In Fig. 4.9, we plot ϵ_0 and λ_0 for an n-input Strong Anti-SAT block with different (n_0, p_0). For each $n \in (32, 48, 64, 80)$, we set $n_0 = 4$ and increase p_0 from 1 to 15. As seen in Fig. 4.9(a), by increasing p_0, the ϵ_0 will increases substantially. For different n, we can always find a configuration (n_0, p_0) such that the ϵ_0 is larger than a desired error rate. A desired ϵ_0 can be estimated to a value such that neural applications of interest are guaranteed to have a high accuracy loss. To estimate a desired ϵ_0, we simulate the relationship between accuracy loss of neural models and error rate of multipliers for 5 benchmarks, as illustrated in Fig. 4.10. Based



(a) (b)

Figure 4.9: Lower-bounds of (a) error rate ϵ_0; (b) SAT iterations λ_0 for different Strong Anti-SAT configurations (n, n_0, p_0).

on this figure, we can estimate a desired ϵ_0 for the multiplier to achieve a sufficient application-level accuracy loss. For example, if we want to achieve 50% averaged accuracy loss, a desired ϵ_0 for the multiplier is estimated to be 10^{-6}. Using such an analysis we can estimate the desired ϵ_0 for the multipliers which can then be used to design the Strong Anti-SAT block (by tuning n_0 and p_0).

4.5.3 SAT Solving Iterations and Execution Time

Increasing ϵ_0, however, will inevitably decreases λ_0, as discussed in Section 4.4.2. This is validated in Fig. 4.9(b), which shows that λ_0 decreases as p_0 increases. To counter this degradation in the number of SAT iterations, we proposed to use a larger multiplier (which multiplies operands with larger bit-width w) [4]. Let's denote the SAT solving time per iteration as t. Fig. 4.11 shows the t for different

[4]The input-size of the multiplier is $n = 2w$.

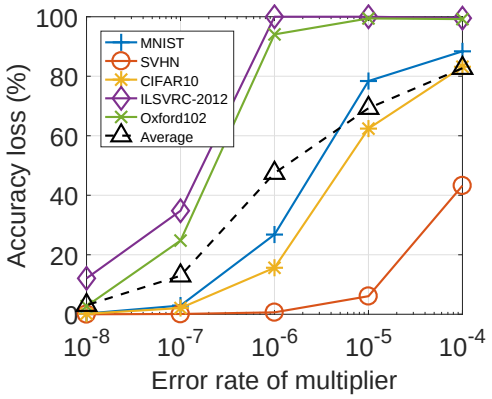


Figure 4.10: Accuracy loss v.s. error rate of multiplier for 5 benchmarks.

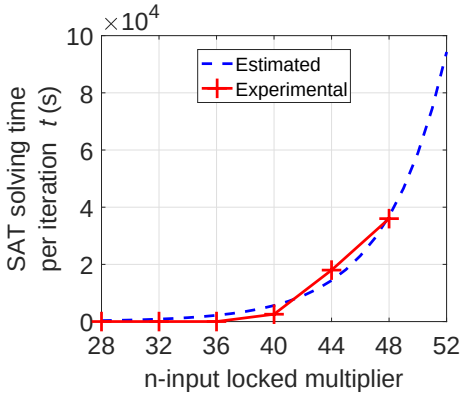


Figure 4.11: SAT solving time per iteration for n-input locked multiplier

n-input fixed-point multipliers locked with Anti-SAT. It's computed by running the SAT attack in 10 hours and then dividing it by the number of iterations the attack can process. As seen, as the input-size n increases, t increases exponentially. To validate the extrapolated exponential increase, we run the SAT attack on a 56-input locked multiplier and find that it can only process 1 iteration in 2.41E+05 seconds (about 67 hours), which is the estimated value based on the exponential curve. Hence such a predictive approach can be used to estimate an appropriate multiplier size for achieving certain t to ensure sufficient total SAT solving time. Based on λ_0 in Fig. 4.9(b) and t in Fig. 4.11, we can compute the lower-bound of total SAT solving time $T_0 = t \times \lambda_0$ for each tuple (n, n_0, p_0) and see if it can satisfy a pre-defined requirement on the total SAT solving time.

In Fig. 4.12, we plot the relationship between T_0 and ϵ_0 for different configurations of (n, n_0, p_0). This is useful for selecting a locking configuration that can satisfy both the SAT solving time requirement $T \geq T_0$ and the error rate require-

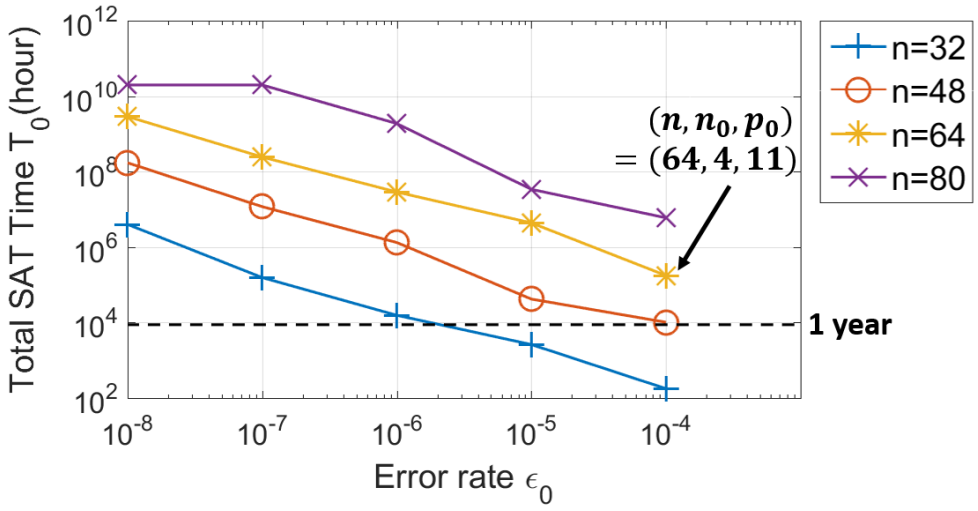


Figure 4.12: Total SAT solving time v.s. error rate

ment $\epsilon \geq \epsilon_0$. To obtain such plot, we first determine the (n, n_0, p_0) which can just achieve certain ϵ_0 based on Fig. 4.9(a). Then, we compute λ_0 and T_0 for each tuple and show the relationship between T_0 and ϵ_0 in Fig. 4.12. As seen, we can always select a configuration (n, n_0, p_0) so as to satisfy a desired ϵ_0 and T_0 simultaneously. As an example, when $(n, n_0, p_0) = (64, 4, 11)$, we have $\epsilon_0 > 10^{-4}$ and $T_0 > 1$ year. Such configuration would result in 80% accuracy loss for neural models running on approx-unlocked chips, as shown in Fig. 4.10.

4.6 Conclusion

In this chapter, we investigate both attack and defense methodologies for locked neural chips. Our proposed attack methodology exploits existing SAT-based attacks and neural model fine-tuning in effective ways. To counter this attack, we propose a secure locking scheme based on a co-design of the locking infrastructure

and the functional modules. Experimental results show that our proposed locking scheme can effectively secure neural chips against the AppSAT attack as well as the exact SAT attack.

Chapter 5: Delay Locking: Security Enhancement of Logic Locking Against Overproduction and Counterfeiting

5.1 Introduction

In Chapter 3, we discussed the vulnerability of many existing logic locking techniques to a strong attack called SAT attack [24]. Countermeasures have been proposed to mitigate the SAT attack [31, 68, 70], including our proposed Anti-SAT. Basically, these countermeasures proposed to insert additional SAT-attack resistant logic blocks such as the Anti-SAT block [31], the SARLock [70], or an AES block with a fixed AES key [68] into the locked circuit to increase the SAT attack iterations and execution time. Although effective, one limitation of all above countermeasures is that these SAT-attack resistant logic blocks have a special and separable structure. They may be removed or nullified by an attacker if they are identified. Then, the SAT attack can be launched to unlock the circuit without these SAT-resistant logic blocks. In this chapter, we propose a new technique called delay locking to enhance the security of existing logic locking techniques. For delay locking, the key to a locked circuit not only determines its functionality, but also its timing profile. A *functionality-correct but timing-incorrect* key will result in timing violations and

thus make the circuit malfunction. The SAT attack is thwarted because it cannot be utilized to decipher a timing-correct key. The contributions of this work are as follows.

- A delay+logic locking (DLL) technique is proposed to enhance the security of existing logic locking techniques to prevent IC counterfeiting and overproduction. It obfuscates the timing profile of a circuit design such that an incorrect key will violate timing constraints and thus make the circuit malfunction.
- A new type of key-gate called tunable delay key-gate (TDK) is introduced, which has two types of keys: functional-key and delay-key. The functional-key controls the TDK's functionality while the delay-key determines its gate delay.
- An overall DLL design flow is proposed, which allocates the new TDK gates and designs the timing constraints for simultaneous functional and delay obfuscation.
- Our proposed approach is fundamentally immune to previous attacks such as the SAT attack because these attacks only focus on deciphering the correct functional-key. Finding the correct delay-key can be formulated to be an instance of mixed-integer-linear-programming (MILP). However, necessary constraint relaxations to satisfy the linear formulation make it fail to find the correct delay-key (as discussed in Sec. 5.4.3). Hence our approach of simultaneous functional and delay obfuscation results in substantial security enhancements.

5.2 Attack Model

This work assumes the same attack model as discussed in Section 2.1.1. The attacker is an untrusted foundry whose objective is to obtain the correct key of a locked circuit and use it to unlock overproduced chips or out-of-spec counterfeit chips. The malicious foundry has access to the following two components:

1. A locked gate-level netlist, which can be obtained by reverse-engineering the layout file of the locked circuit provided by the designer.

2. An activated functional chip, which can be obtained from an open market. This chip can be used to observe a set of correct I/O pairs as a black box.

5.3 Delay+Logic Locking (DLL)

To enhance the security of existing logic locking techniques, we propose a new technique called delay locking that can thwart existing attacks on logic locking. The basic idea of delay locking is to make the circuit's delay dependent on the key value. When logic locking is enhanced with delay locking, the key into a locked circuit not only determines its functionality but also its timing profile. A correct key value should recover the original combinational functionality as well as the correct timing profile that can satisfy a set of pre-defined timing constraints. On the other hand, a key is incorrect if it fails to recover a) the original functionality or b) the correct timing profile. Since previous attack algorithms on logic locking (described in Section 2.2.2.2) only focus on retrieving the correct combinational functionality,

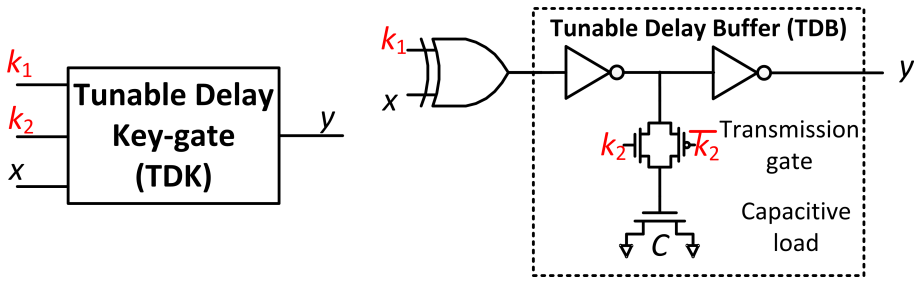
they are not guaranteed to recover the timing-correct key. A functionality-correct but timing-incorrect key will result in timing violations and thus make the circuit malfunction.

In the remaining of this section, we discuss how the delay locking is implemented and introduce the design objectives, design techniques, and the overall design flow of the DLL design.

5.3.1 Tunable Delay Key-gate (TDK)

To make the delay of a key-gate dependent on its key value, we propose a tunable delay key-gate (TDK). Fig. 5.1 illustrates the structure of the TDK, which combines a conventional key-gate (XOR/XNOR) with a tunable delay buffer (TDB) [89]. TDB is a widely used solution for post-silicon adjustment of gate/circuit delays. One typical application of TDBs is to correct timing violations that are induced by the process, temperature and other variances. Various implementations of TDBs have been proposed in the previous literature. One low-power TDB design is proposed by Tsai *et al.* [89], which is based on two inverters with a set of NMOS-based capacitive loads in between, as shown in Fig. 5.1(b). Each capacitive load is controlled by a transmission gate. When the transmission gate is activated by its control signal, the corresponding capacitive load is added into the path between the pair of inverters, thus obtaining tunable delays.

As shown in Fig. 5.1, each TDK has two key-inputs, one feeding into the XOR/XNOR gate (referred to as the *functional-key*) and the other feeding into the



(a) (b)

Figure 5.1: Tunable delay key-gate (TDK): (a) overview; (b) implementation

Table 5.1: Functionality and delay of the TDK

Key (k_1k_2)	Functionality	Delay
00	$y = x$	d_0
01	$y = x$	d_1
10	$y = \bar{x}$	d_0
11	$y = \bar{x}$	d_1

control signal of the TDB (referred to as the *delay-key*). The impact of the keys on the functionality and delay of the TDK is shown in Table 5.1. The functional-key k_1 determines whether the TDK behaves as a buffer or an inverter while the delay-key k_2 determines whether the TDK gate delay is d_0 or d_1. The TDK delay ratio

$$r = d_1/d_0 \tag{5.1}$$

can be set at design time by tuning the capacitive load. This delay ratio has a great impact on the circuit delay distribution across different key values as well as the timing violation sensitivity, which will be discussed in Section 5.3.3.2.

In the remaining of the paper, we refer a circuit that's locked with the TDKs to be a *delay-logic-locked circuit (DLL circuit)*.

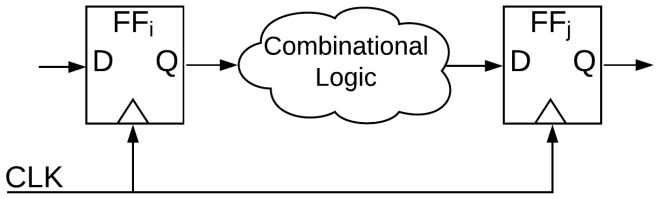


Figure 5.2: A simple sequential circuit.

5.3.2 Timing Constraints of DLL Circuit

Now we describe how the allocation of TDKs impacts the overall timing constraints of the design. We first describe the timing constraints for a conventional sequential circuit, as shown in Fig. 5.2. Given a sequential design, we can represent it as a directed graph $G = (V, E)$, where V is a set of flip-flops (FFs) and E is a set of edges representing the combinational logic paths between the FFs. We want to analyze the timing constraints for the path delay between any two FFs i and j. Two types of timing constraints are considered in a sequential circuit, namely *longest-path timing constraint* and *shortest-path timing constraint*. In a nutshell, the longest-path timing constraint ensures that the combinational netlist shall propagate the logic computation in time to the destination FF. On the other hand, the shortest-path timing constraint ensures that the combinational netlist shall not propagate too fast such that it contaminates the correct value that needs to be stored in the destination FF. The following formally describes these two timing constraints.

Let us assume that T_i and T_j are the clock arrival time at FFs i and j. T_i and T_j may not be the same due to clock skews (the spatial variation in arrival time of different FFs). Let D_{ij}^{long} be the longest-path delay between FFs i and j, *i.e.*,

the maximum delay among all combinational logic paths between two FFs i and j. Let T_{set}^{j} be the setup time for FF j and T_{clk} be the clock period. To meet the longest-path timing constraint, the circuit needs to satisfy:

$$D_{ij}^{long} + T_{set}^{j} \leq T_{clk} + T_j - T_i, \forall i, j \tag{5.2}$$

This longest-path timing constraint indicates that the clock period should be large enough for the data to propagate through the combinational logic paths and to be set up at the destination FF before the next trigging edge of the clock arrives.

Besides, the circuit should also satisfy the shortest-path timing constraint (hold time constraint) between two FFs. Let D_{ij}^{short} be the shortest-path delay between FFs i and j and T_{hold}^{j} be the hold time for FF j. The hold time of the destination FF j must be shorter than the shortest-path delay through the combinational logic network, considering the clock skew phenomenon:

$$D_{ij}^{short} \geq T_{hold}^{j} + T_j - T_i, \forall i, j \tag{5.3}$$

Based on Eq. (5.2) and Eq. (5.3), we can represent the timing constraints for a DLL circuit as follows. Let $D_{ij}^{short}(\vec{K})$ and $D_{ij}^{long}(\vec{K})$ be the shortest/longest path delay between FFs i and j of a DLL circuit with a key $\vec{K}$. The timing constraints for the DLL circuit can be represented as:

$$\begin{aligned} D_{ij}^{short}(\vec{K}) &\geq T_{hold}^{j} + T_j - T_i \equiv LB_{ij} \\ D_{ij}^{long}(\vec{K}) &\leq T_{clk} + T_j - T_i - T_{set}^{j} \equiv UB_{ij} \end{aligned} \tag{5.4}$$

Eq. (5.4) enforces that the combinational path delays (for a correct key $\vec{K} = \vec{K}_C$) between two FFs should satisfy both the shortest and the longest timing constraints.

Allocation of keys to the fastest or slowest corner may not be the correct timing solution because a key that increases delay may violate the upper bound (UB) and a key that decreases delay may violate the lower bound (LB). This makes the delay-key determination problem very hard from an attacker's standpoint.

5.3.3 DLL Design Flow

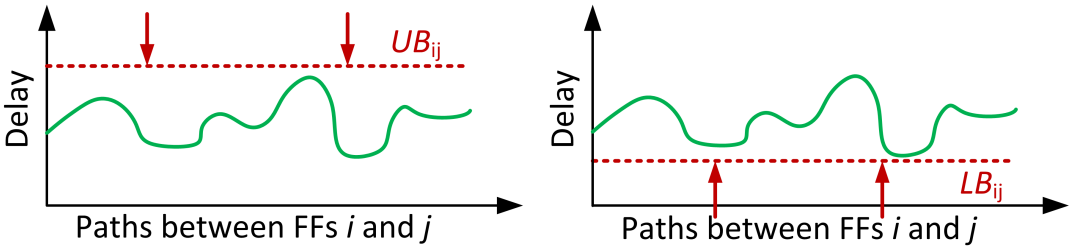
In this section, we introduce the design objectives, design techniques, and the overall design flow of the DLL design. In general, the design problem is formulated as follows: we want to achieve simultaneous functionality and delay obfuscation by allocating the TDK gates and designing the timing constraints as shown in Eq. (5.4).

5.3.3.1 Design Objective

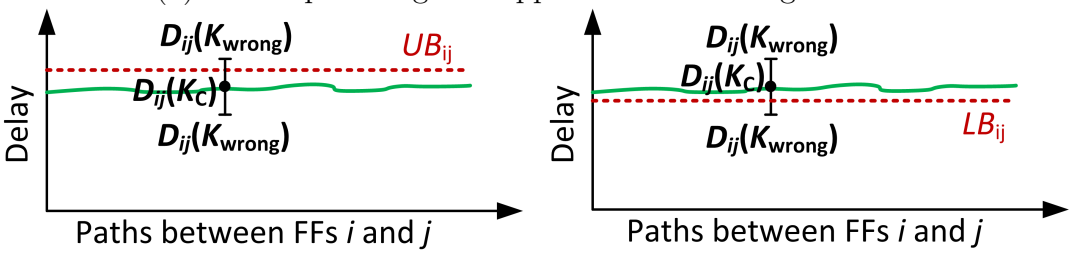
The objective of DLL design consists of two aspects:

1. *Functionality obfuscation.* The functionality of a circuit shall be obfuscated in order to conceal the correct functionality when it passes through the untrusted foundry and other potentially untrusted phases of the supply chain. This requires the TDKs to be located in *functionality-critical* spots.

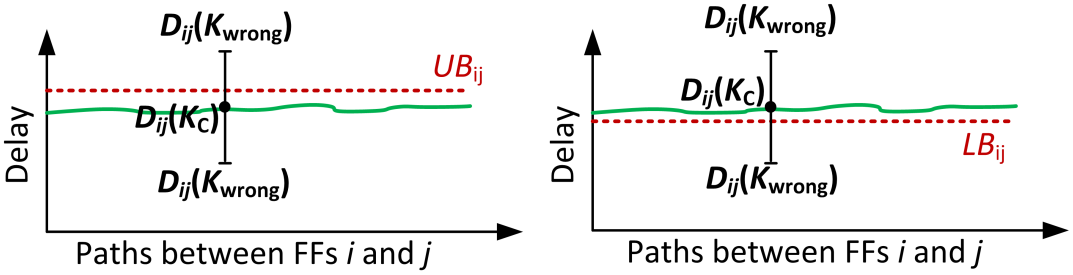
2. *Delay obfuscation.* The timing profile of the locked circuit should be obfuscated in order to defend the functionality-oriented attacks such as SAT attack. For an incorrect delay-key, at least one path will violate either the longest or the shortest-path timing constraint as described in Eq. (5.4). This requires the paths which comprise the TDKs to be *timing-critical*.



(a) Technique 1: tighten upper or lower timing bounds



(b) Technique 2: path delay balancing



(c) Technique 3: increase TDK gate delay ratio r

Figure 5.3: Illustrative examples of three design techniques for delay locking

5.3.3.2 Design Techniques

The first design objective (functionality obfuscation) can be achieved by previously proposed key-gate insertion algorithms [20, 22, 47], which insert the key-gates to internal wires of a netlist to obfuscate the original functionality. However, such locations might not belong to the timing-critical paths. Therefore, an incorrect key might not result in sufficient timing violations to cause an error. To achieve better

delay obfuscation, we need to make the paths which comprise the TDKs become timing-critical. We propose the following three design techniques to achieve this.

1) Timing Bound Design. The timing bounds UB_{ij} or LB_{ij} shall be sufficiently tightened to the longest or shortest path delay, respectively. In other words, either UB_{ij} is set to be larger than but sufficiently close to $D_{ij}^{long}(\vec{K}_C)$, or LB_{ij} is set to be less than but sufficiently close to $D_{ij}^{short}(\vec{K}_C)$, as illustrated in Fig. 5.3 (a). When the bounds are sufficiently tight, it can ensure that the paths with delay values that are very close to LB_{ij} or UB_{ij} can violate the timing bounds if the delay-keys are incorrect. As shown in Eq. (5.4), LB_{ij} and UB_{ij} can be tuned by changing the clock arrival time T_i and T_j, which can be controlled by the clock tree design [90]. Clock tree design which exploits the tuning of T_i and T_j is also called useful-skew based optimization. In this case, we exploit useful skews for security purpose.

2) Path Delay Balancing. The path delays shall be balanced such that every path delay is close to the longest-path delay of the whole combinational block, denoted as $D^{balanced}$. Imbalanced path delays mean that TDKs on the non-critical paths will not result in desired timing sensitivity. On the contrary, making all paths almost equally critical would make the timing profiles sensitive to all TDKs. Hence, path delay balancing makes it easier for the circuit to violate the timing constraints when the delay-key is incorrect. Path delay balancing is a widely used technique for eliminating glitches. There exist many approaches for path delay balancing. In this work, we exploit gate sizing and buffer insertion [91] to achieve the path delay balancing.

3) Increase TDK Delay Ratio r. The TDK delay ratio r shall be large enough

to ensure a desired timing violation magnitude, *i.e.,* the difference between an incorrect delay and the violated timing bound. A larger r indicates that when a key bit flips, the delays of the paths that comprise this TDK will increase or decrease with a larger magnitude. Therefore, these paths would have more severe timing violations, as shown in Fig. 5.3(c). A larger timing violation magnitude is preferred because it indicates a higher probability of fault occurrences.

The above mentioned three techniques will be used to achieve a high timing violation sensitivity to incorrect keys leading to maximum chances of delay locking to be effective. To quantify the timing violation level of a DLL circuit with an incorrect key, we define a security metric call timing violation ratio (TVR). For FFs i and j, the TVR of a key $\vec{K}$ can be calculated as:

$$TVR_{ij}(\vec{K}) = \frac{\max\{0, D_{ij}^{long}(\vec{K}) - UB_{ij}, LB_{ij} - D_{ij}^{short}(\vec{K})\}}{D^{balanced}} \tag{5.5}$$

A larger TVR indicates more timing violation and a higher probability of fault occurrences. On the other hand, for a correct key, there is not timing violation and TVR is 0. Based on Eq.(5.5), we can define the TVR for the whole circuit as follows:

$$TVR(\vec{K}) = \max_{\text{FFs } i,j}\{TVR_{ij}(\vec{K})\} \tag{5.6}$$

This metric captures the maximum timing violations (if any) among all pairs of FFs.

5.3.3.3 Design Flow

The overall delay locking design flow is shown in Fig. 5.4. It consists of two design phases: logic locking and delay locking.

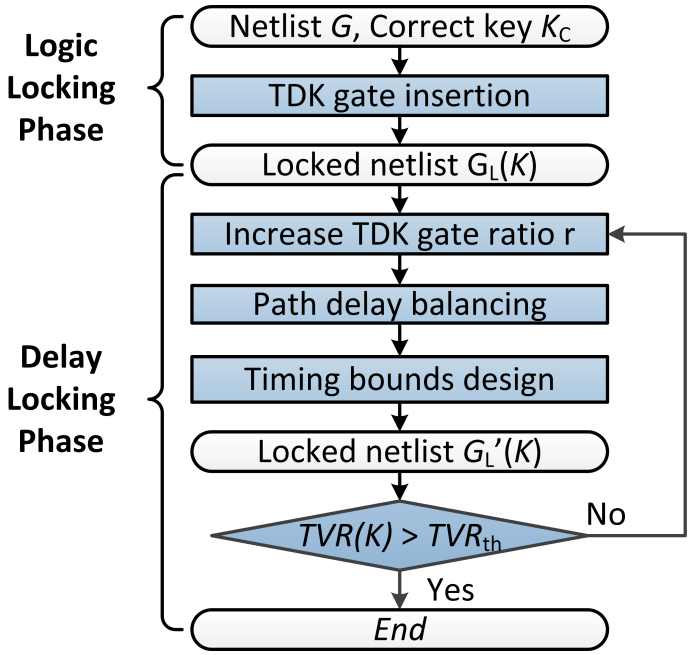


Figure 5.4: DLL design flow

Logic Locking Phase: Given a netlist G and a (randomly generated) correct key $\vec{K}_C$, we first integrate the TDK gates into the combinational block of the original netlist and produce a locked netlist $G_L(\vec{K})$. The locations for the TDK gates can be determined using previously proposed key-gate insertion algorithms, such as random insertion [20], fault-analysis based insertion [47] and interference-analysis based insertion [22], which is not the focus of this work.

Delay Locking Phase: In the delay locking phase, we apply three design techniques as discussed in Section 5.3.3.2 to improve the timing violation sensitivity. The TDK gate delay ratio r is gradually increased until the $TVR(\vec{K})$ (defined in Eq.(5.6)) for a random key is larger than a pre-defined security threshold TVR_{th}. After the delay locking phase, we obtain the final DLL design $G'_L(\vec{K})$.

5.4 Security Analysis of DLL

In this section, we analyze the security of DLL technique against different attacks.

5.4.1 TDK Removal Attack

The attacker might attempt to nullify or bypass the TDK gates (either the XOR/XNOR gate, the TDB, or both) in the DLL circuit by replacing it with a normal wire. However, such attempt cannot recover the original functionality and delay if he does not know the correct functional-key and delay-key. On one hand, a TDK can function as a buffer or inverter depending on the functional-key. Replacing it with a wire might flip the correct functionality. On the other hand, nullifying the TDK gates (or just the TDB) is simply equivalent to decreasing the path delays, which might violate the timing lower bound ensured by the shortest-path timing constraint.

5.4.2 Functionality Oriented Attacks

As discussed earlier, previous attacks [22–24] on logic locking such as the SAT attack algorithm are all functionality oriented, which means that an attacker only focuses on finding a functionality-correct key w.r.t. the combinational logic of the circuit. However, when the delay locking technique is utilized, the key obtained by these attacks cannot unlock the overproduced locked chip because they are not guaranteed to obtain a timing-correct key. A circuit based on a functionality-correct (but

not timing-correct) key will violate the pre-defined timing constraints and thus produce incorrect outputs, which will be shown in the experiment section (Section 5.5). As a result, the functionality oriented attacks fail to unlock a chip that is enhanced with the delay locking technique.

5.4.3 MILP Based Delay-key Attack

Assuming an attacker can obtain a correct functional-key using previously proposed attacks, in order to unlock the circuit, he has to find a correct delay-key that can satisfy all pre-defined timing constraints. Here we assume a stronger attack model, where the attacker knows the timing LB and UB of all paths and he intends to formulate a mixed integer linear programming (MILP) to solve the delay-key. A straightforward formulation can set UB and LB as constraints for each path delay and find a delay-key that satisfies these constraints. However, this direct formulation is impractical because the number of possible signal paths can be exponential in the total number of gates [92]. This issue can be handled by the classic technique which divides the constraints on path delay into constraints on a gate's arrival time [92]. Let a_i denotes the arrival time of the output of a gate or a primary input i. Let g_i denotes the delay of a gate i. If it's a TDK gate, then g_i is dependent on the

delay-key value denoted as x_i. We can formulate the above problem as:

$$
\begin{aligned}
&\text{find } \vec{x}, \vec{a} \\
&\text{s.t. } LB_j \leq a_j \leq UB_j, && \forall \text{PO } j \\
&\quad a_j + g_i \leq a_i, && \forall \text{gate } i, \forall j \in inputs(i) \\
&\quad g_i \leq a_i, && \forall \text{PI } i \\
&\quad g_i = x_i \times d_1 + (1 - x_i) \times d_0, && \forall \text{gate } i \in TDK \\
&\quad x_i = 0 \text{ or } 1, && \forall \text{gate } i \in TDK
\end{aligned}
\tag{5.7}
$$

But the above formulation might not recover the correct arrival time and delay-key because the timing constraint for a gate $a_i = \max\{a_j + g_i\}, \forall j \in inputs(i)$ is relaxed to be $a_i \geq a_j + g_i$ in order to form an MILP formulation. This relaxation will make the arrival time a_i inaccurate and result in an incorrect delay-key. As will be shown in Section 5.5, the resulting key values from above MILP formulation will violate the pre-defined timing constraints. The attacker can attempt to iteratively run the MILP attack and add new constraints on the key values to prune out the incorrect keys discovered in previous iterations. However, since each iteration can only prune out one incorrect key, when key-size is large, the execution time to find a correct delay-key will be exponential in the key-size, as will be shown in Section 5.5.

5.5 Experiments and Results

5.5.1 Experiment Setup

We validate the effectiveness of the delay locking technique using 8 sequential benchmarks from ISCAS89. The benchmark information is shown in Table 5.2. Each benchmark is synthesized using Cadence RTL compiler with SAED 90nm digital standard cell library. Timing information of the standard cell library is extracted for timing analysis.

For the TDK, when the delay-key $k_2 = 0$ the gate delay d_0 is the XOR/XNOR gate delay plus a buffer delay. When $k_2 = 1$, its gate delay is set to be $d_1 = r \times d_0$, where r is the delay ratio. The TDKs are implemented and simulated using Cadence Virtuoso and its gate area and delay are obtained for overhead evaluation. In the logic locking phase, we randomly generate a correct key and adopt the random key-gate insertion algorithm [20] to insert key-gates into the combinational block of the benchmark. The number of key-gates equals 10% the number of original gates, as shown in Table 5.2. In the delay locking phase, we apply three design techniques (Section 5.3.3.2) to improve delay obfuscation.

To evaluate the level of timing violation given an incorrect delay-key, we compute the TVR (defined in Eq. (5.6)) for the DLL circuit under the case that the functional-key values are all correct but the delay-key values are randomly generated. Noted that in the experiment we set the timing bounds to the balanced path delay for the easy of TVR computation and comparison. The TVR is averaged over

Table 5.2: Benchmark information and MILP based delay-key attack results.

Circuit	#Gates	#FFs	#Key gates	MILP Attack	
				Correctness ($\vec{K}_{guess}$)	TVR ($\vec{K}_{guess}$)
s1488	336	6	34	44.12%	28.13%
s5378	748	179	75	58.67%	39.25%
s9234	1014	211	101	59.41%	34.60%
s13207	1924	638	192	63.02%	33.60%
s15850	1952	534	195	63.08%	27.64%
s35932	4763	1728	476	68.07%	26.18%
s38417	5066	1636	507	60.95%	31.87%
s38584	6857	1426	686	58.60%	36.39%

1000 random key trails.

5.5.2 Results

5.5.2.1 Effectiveness of Proposed Design Techniques

In this experiment, we validate the effectiveness of our proposed design techniques (Section 5.3.3.2) in improving the level of timing violations. We assume technique 1 is always applied and evaluate the impact of other two techniques: path delay balancing and increasing TDK delay ratio r. Fig. 5.5 shows the TVR values of different delay ratios r when the path delay balancing is applied (bold lines) and when it's not applied (dash lines). As seen, without path delay balancing (dash lines), the TVR values of most benchmarks are below 5%, and increasing the delay

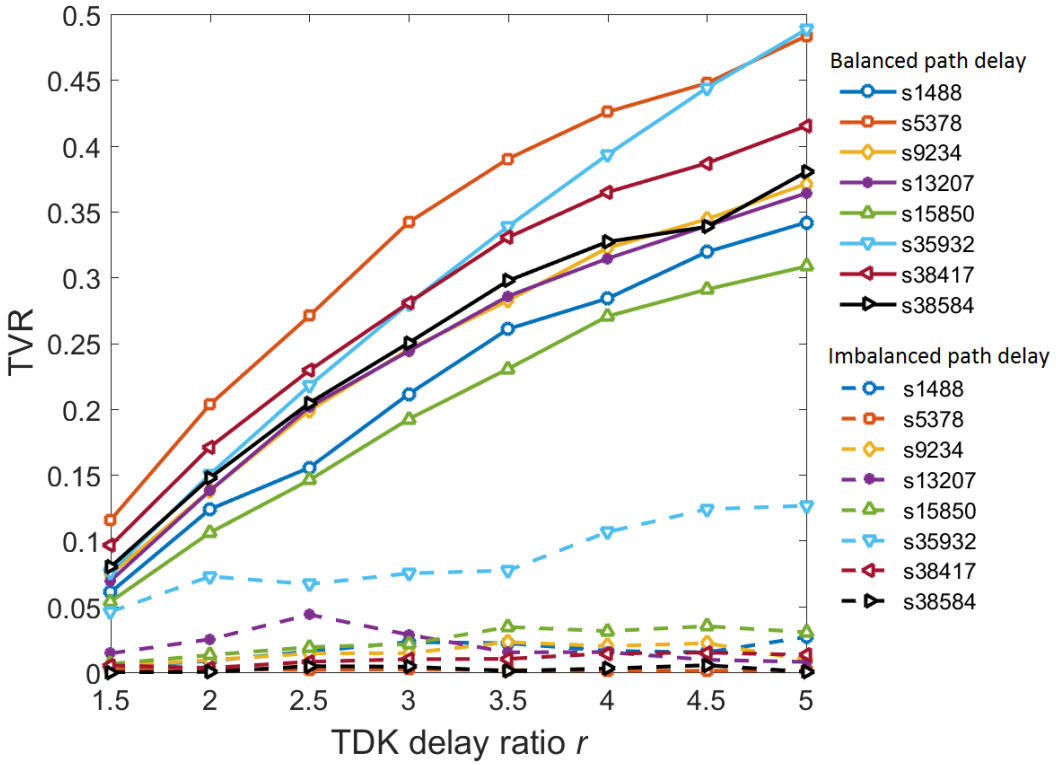


Figure 5.5: The impact of path delay balancing and TDK delay ratio r on the TVR for 8 ISCAS89 benchmarks.

ratio r cannot effectively achieve a higher TVR value. This is because that when path delays are imbalanced, most TDKs are not on the timing-critical paths so an incorrect delay-key cannot cause severe timing violations. When path delay balancing is applied (bold lines), we can see a remarkable improvement in TVR for all choices of r. Besides, as r increases, the TVR value increases from about 8% to 39%. The value of r can be designed to achieve a desired TVR threshold.

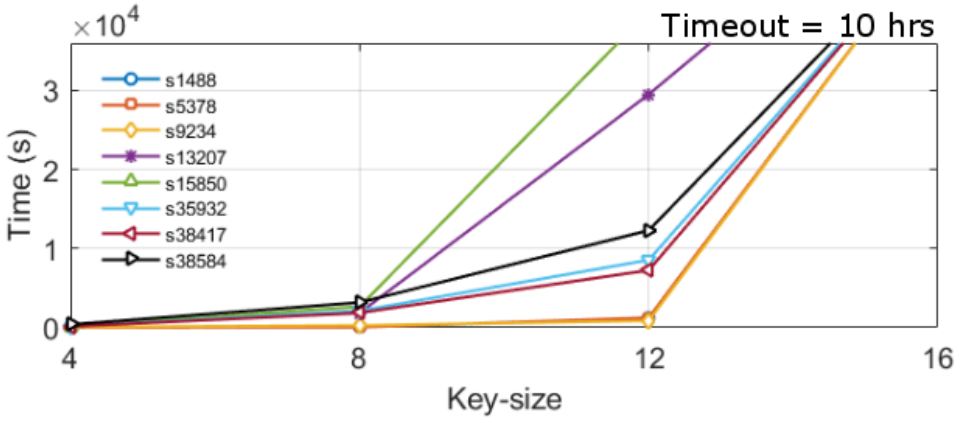
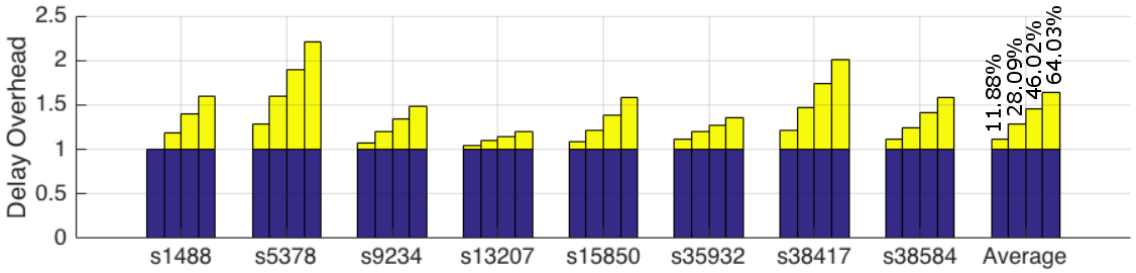


Figure 5.6: Iterative MILP attack results (Timeout is 10 hours)

5.5.2.2 MILP-based Delay-key Attack

As discussed in Section 5.4.3, a MILP formulation might be applied by an attacker to find the delay-key. However, such formulation requires necessary constraint relaxation to satisfy the linear formulation and thus fail to retrieve the original delay and the correct delay-key. To validate this analysis, we implemented the MILP formulation based attack. For each DLL circuit, r is set to 3 and path delay balancing is applied because they can result in a relatively large TVR value as shown in Fig. 5.5. The correctness (# bits that are the same as the correct key) and the corresponding TVR values of the MILP solution $\vec{K}_{guess}$ are computed and shown in Table 5.2. As seen, the resulting delay-keys are incorrect for all benchmarks and they will violate the timing constraints with an average TVR of 32.21%. We also implement an iterative MILP attack which iteratively performs the MILP attack and adds new constraints on the key values to prune out the incorrect keys discovered in previous iterations. The attack results on 8 benchmarks for key-size ranging

(a) Area overhead



(b) Delay overhead

Figure 5.7: Area and delay overhead for the DLL technique. Four bar plots of each benchmark correspond to TDK delay ratios $r = 2, 3, 4, 5$

from 4 to 16 are shown in Fig. 5.6. As seen, since each iteration can only prune out one incorrect key, as the key-size increases, the execution time for finding a correct delay-key increases exponentially. When key-size is 16, the attack timeouts ($\geq$10 hrs) for all benchmarks.

5.5.2.3 Overhead Evaluation

Fig. 5.7 shows the area and delay overhead of the DLL technique for 8 benchmarks when compared to the conventional XOR/XNOR based logic locking. For each benchmark, we vary the TDK delay ratio r and report its impact on area and delay. Four bar plots of each benchmark correspond to $r = 2, 3, 4, 5$. As seen,

when r increases from 2 to 5, the average area overhead increases from 4.36% to 5.29%. The area overhead mainly comes from the TDBs. A larger r requires a larger capacitive load so it results in a slightly higher area overhead. The impact of r is mainly reflected in the delay overhead. As seen, with r increases, the averaged delay overhead increases from 11.88% to 64.03%. This is because that a larger r will lead to a larger key-gate delay (if the correct delay-key is 1) and increase the delay for the overall circuit. Combined with the result in Fig. 5.5, we can see that our approach is capable of generating a tradeoff between the TVR and the performance overheads. By tuning the TDK delay ratio r, we can achieve a desired TVR with acceptable overheads.

5.6 Conclusion

In this chapter, we present a new technique called delay locking to enhance the security of existing logic locking techniques. A tunable delay key-gate is proposed to obfuscate both the functionality and timing profile of an IC design. An overall delay+logic locking design flow is proposed to increase the timing violation sensitivity to incorrect key values. The security of proposed delay locking technique is evaluated with previous attacks and a new attack based on MILP. Both analytical and experimental results show that such attacks fail to find the correct delay-key.

Chapter 6: Security-aware Design Flow for 2.5D IC Split Fabrication

6.1 Introduction

The increasing trend of outsourced fabrication for modern chips makes circuit designs vulnerable to IP piracy or counterfeiting by untrusted foundries. 2.5D IC technology has shown the capability to counter this threat. By limiting the interposer layer of a 2.5D IC that contains inter-chip connections to be fabricated in a trusted foundry, the complete exposure of original design to an untrusted foundry is prevented. This fabrication strategy is called split fabrication. In this chapter, we propose a security-aware physical design flow for 2.5D IC technology to prevent IP piracy. The partitioning phase utilizes the concepts of controllability and observability to conceal the functionality of a design and the placement phase generates obfuscated chip layouts that can withstand layout-geometry based attacks such as proximity attack. Simulation results show that our design flow is effective for producing secure chip layouts for outsourcing whose original netlist and functionality cannot be reverse-engineered based on the layout-geometry information. The contributions of this work are as follows.

- We propose a security-aware physical design flow for 2.5D IC split fabrication

to address the IP piracy threat in outsourced fabrication.

- A secure min-cut bi-partitioning algorithm based on the concepts of controllability and observability is proposed to reduce the cut-size while enforcing the controllability and observability of wires in the cut-set are relatively high so that incorrect connections between two partitions will lead to incorrect outputs. Hence incorrect reconnection of outsourced sub-netlists can't disclose the correct functionality.

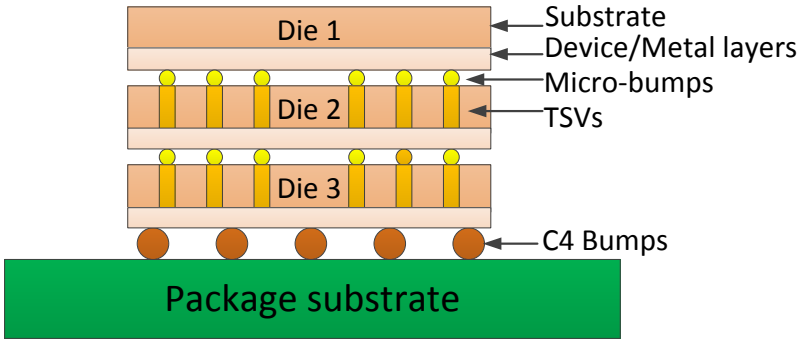
- A secure simulated-annealing based placement algorithm is proposed to thwart the layout-geometry based attack algorithm (*i.e.,* proximity attack) while balancing the performance overhead on area and wire-length.

- We evaluate the security level of our design flow under proximity attack on 8 publicly available benchmarks. Simulation results show that our approach can effectively prevent a malicious foundry from reverse-engineering the complete functionality and netlist (46.35% Hamming distance and 0.27% correctness of reconnection under proximity attack).

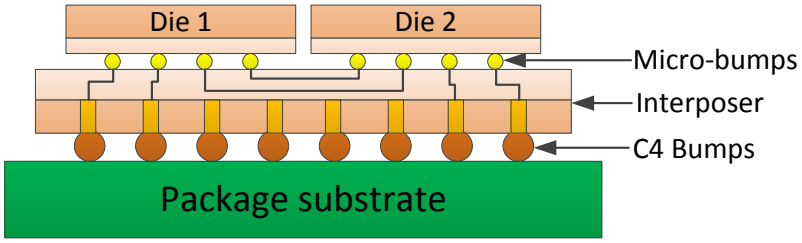
6.2 Preliminary

6.2.1 3D/2.5D Integration

Technology scaling which shrinks the physical feature size of transistors has long been an effective approach to improve chip performance. However, this approach is now experiencing asperities mainly due to the physical limit of transistor



(a)



(b)

Figure 6.1: Structures of (a) stacked 3D IC and (b) 2.5D IC.

miniaturization. This motivates the development of 3D integration technology.

3D integration is a technology that vertically integrates multiple 2D dies to create a single high-performance chip named 3D IC. A common configuration of 3D IC is shown in Fig. 6.1(a). Multiple 2D dies (which contain device/metal layers) are stacked and interconnected using vertical connections called Through-Silicon-Vias (TSVs). 3D integration reduces interconnect wire-length because two distant devices in a conventional 2D design can be placed vertically close to each other and connected with a shorter connection. The reduction in wire-length scales down interconnect power and delay, which can be leveraged by implementing a more highly connected architecture such as the high-bandwidth memory-on-chip architecture [93].

Moreover, 3D integration allows heterogeneous integration, which integrates components of different materials and technologies into a single chip. Recent years have seen a lot of research working on improving the performance and reliability of 3D integration technology [94].

Two common structures of 3D ICs are *stacked 3D IC* and *interposer-based 3D IC* (also known as *2.5D IC*). Fig. 6.1(a) illustrates the structure of a stacked 3D IC. Multiple TSV-penetrated dies are stacked and bonded vertically. However, the increased device density in stacked 3D ICs brings about thermal, power and reliability issues. To alleviate these issues, 2.5D IC has been proposed. The structure of 2.5D IC is shown in Fig. 6.1(b). Unlike the stacked 3D ICs, 2.5D IC places multiple dies side-by-side and bonds them on a silicon interposer through fine-pitch micro-bumps. The interposer contains horizontal chip-scale wires for inter-die connections as well as vertical TSVs to connect with external I/O pins. Note that in 2.5D ICs, the dies are not penetrated by the TSVs. The absence of TSVs in the dies of 2.5D IC makes it easier to design and fabricate than the TSV-penetrated stacked 3D IC. Although 2.5D ICs might not achieve the same amount of performance improvement as 3D ICs, it offers better cooling options, which is essential for high-performance computing systems. While commercial large-scale 3D IC is still being developed, large-volume commercial 2.5D products are already in the market, such as the Xilinx Virtex-7 2000T FPGA [95].

6.2.2 3D/2.5D IC Based Split Fabrication

As 3D/2.5D integration is becoming a promising technology for next-generation chip design, researchers have started to investigate it from a hardware security perspective [96]. One line of research focuses on utilizing 3D/2.5D IC technology to protect IC designs from being pirated or tampered during outsourced fabrication [34, 58, 62, 97–99]. In 3D integration, multiple dies (functional layers) can be fabricated independently on separate substrates and then integrated together into a single chip. This fabrication process offers inherent support for split fabrication. Split fabrication, as introduced in Section 2.2.3, is a layout-level circuit obfuscation technique. To adopt split fabrication, a designer can first partition a circuit design into different functional layers in the 3D chips. Then, he can send a portion of the layers at his discretion to a trusted foundry for secure fabrication while outsourcing the rest to an untrusted foundry for state-of-the-art fabrication technology. The final integration between two components is done in the trusted foundry. Because some information will be hidden from the untrusted foundry, split fabrication can prevent the supply chain attacks such as piracy, overbuilding, and counterfeiting. The split fabrication strategy of 3D/2.5D IC is adaptable to off-the-shelf 3D/2.5D IC fabrication process. Each die is an individual component that can be fabricated separately and then integrated together, either in a single foundry or in different foundries. Interconnecting separately made dies using 3D integration is already a proven technology [59]. Thus, the extra effort for 3D/2.5D IC to adopt the split fabrication is negligible.

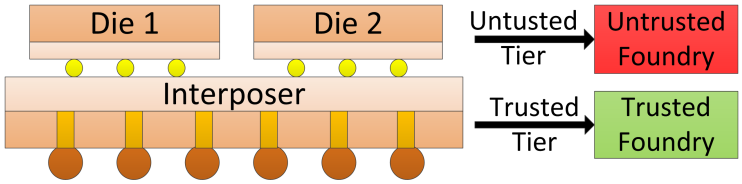


Figure 6.2: 2.5D IC based split fabrication.

In this work, we focus on 2.5D IC based split fabrication. 2.5D IC has less severe thermal and reliability challenges while offering a comparable performance improvement compared to the stacked 3D IC. Moreover, leveraging this 2.5D integration technology requires only minor modification to current IC design flow and fabrication process. Fig. 6.2 illustrates the basic idea of 2.5D split fabrication. The silicon interposer in a trusted foundry is fabricated in a trusted foundry (as the trusted tier) while the dies are outsourced to untrusted foundries (as the untrusted tier). If all untrusted foundries are independent (not colluded), an attacker in one untrusted foundry can only obtain the netlist of a die that's fabricated in this foundry. Even if the offshore foundries collude, they can at most obtain an incomplete design that lacks the interconnect wires in the interposer. The incomplete netlist will be incomprehensible if the wires in the interposer layer are intelligently selected.

6.3 Security-aware Design Flow for 2.5D ICs

As discussed, by fabricating the interposer of 2.5D IC in a trusted foundry while outsourcing the rest to an untrusted foundry, an attacker in the untrusted foundry can at most obtain an incomplete netlist which lacks the wires in the inter-

poser (the trusted tier).

However, this doesn't imply that a conventional performance-driven 2.5D IC design flow followed by a split fabrication strategy is security-optimal. In a performance-driven 2.5D IC design flow, a netlist is first partitioned in a way that minimizes the number of cut-wires to reduce the number of wires that need to be routed in the trusted tier. Then, corresponding layouts are generated using placement and routing algorithms which minimize layout area and routing wire-length. Although a min-cut partitioning has a lower performance overhead, it might not hide enough wires to fully obfuscate the functionality of the outsourced designs. Also, a performance-driven placement might place two connected pins/gates close-by, thereby leaking the information about the hidden connections that can be exploited by an attacker.

Here we introduce a security-aware 2.5D IC design and split fabrication flow that aims at thwarting hardware IP piracy. The security-aware 2.5D IC design and split fabrication flow is shown in Fig. 6.3. In the design stage, a gate-level netlist is first partitioned into two parts. The cut-wires are selected as the hidden wires in the interposer layer. After that, a placement and routing phase assigns exact physical locations for gates in two partitions and determines the proper intra-die routing as well as the inter-die routing in the interposer. This is followed by the fabrication stage, where the layout files of the dies are outsourced to an untrusted foundry while the interposer is fabricated in a trusted foundry for security. The final integration is also implemented in the trusted foundry. Note that this design and fabrication flow assumes only one untrusted offshore foundry that is responsible for fabricating two

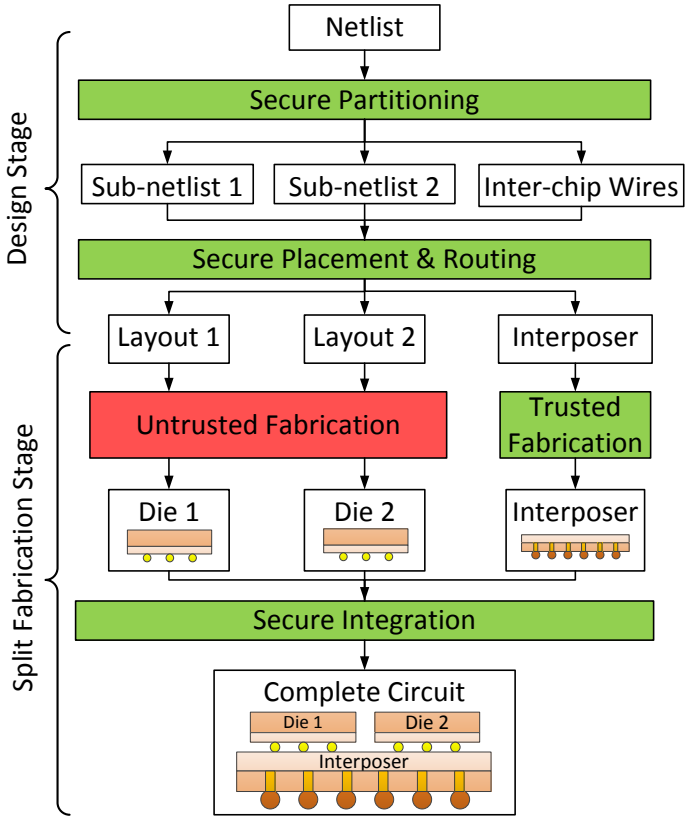


Figure 6.3: A security-aware 2.5D IC design and split fabrication flow.

dies. However, it's possible that two dies can be outsourced to different foundries and if these foundries are completely independent (no collusion), the information leakage to each foundry can be reduced. Moreover, this design flow focuses only on bi-partitioning for simplicity, but it would be possible to partition into more layouts and use more "independent" foundries for better security.

6.4 Problem Formulation

6.4.1 Attack Model

The attack model addressed in this work is widely used in previous anti-IP-piracy research [27, 100]. It assumes that the attacker is an untrusted foundry that has access to the GDSII layout files of two sub-netlists, but it lacks the knowledge of the correct interconnection between two sub-netlists. The interconnection in the interposer layer is not accessible to the attacker. Inferring the interconnect wires in the interposer layer by reverse-engineering a final product from the market is assumed thwarted by anti-reverse-engineering techniques such as camouflaged smart filling [101].

The attacker's goal is to retrieve the complete gate-level netlist and functionality. Since an ideal security-aware design will produce incorrect output logics on applying incorrect connections, any random reconnection should lead to a completely different functionality. Therefore, the attacker's first concern is to determine the missing connections based on two sub-netlists and chip layout geometries, and generate a *reconstructed circuit* so that he can gain profit from pirating and overbuilding the IC.

6.4.2 Problem Statement

The goal of the security-aware 2.5D IC design flow is to thwart IP piracy by producing a security-aware partitioning and placement solution that can obfuscate

the original functionality while preventing the leakage of the correct interconnections between two partitions. The research problem can be defined as follows:

Given a netlist of a combinational circuit and the Boolean function F that maps its primary inputs (PIs) $\vec{X}$ to its primary outputs (POs) $\vec{Y}$: $\vec{Y} = F(\vec{X})$, we want to find a bipartition and a corresponding gate-level placement result, so that the placement result of two partitions will disclose the least functionality and netlist of the original circuit at a minimum performance cost.

6.4.3 Security Objectives

1) Functionality Obfuscation. One objective of 2.5D split fabrication is to obfuscate the functionality of the outsourced design (the untrusted tier). To do so, we select and hide a set of wires into the trusted tier such that the functionality of the untrusted tier (or a reconstructed circuit that's inferred based on the untrusted tier) differs substantially from the original functionality. By obfuscating the functionality, an attacker who has the knowledge of the untrusted tier cannot infer or utilize the functionality of the original complete design, thereby protecting the outsourced design from piracy and overproduction. *Hamming distance (HD)* is widely used to quantify the security level of functionality obfuscation [27,34,100,102]. It's defined as the number of different output bits between an original netlist and a reconstructed netlist on applying the same input vector. Given one input vector $\vec{X_i}$, the function of original netlist F will produce an output vector $\vec{Y_i} = F(\vec{X_i})$, while the function of reconstructed netlist F' will produce another output vector $\vec{Y_i}' = F'(\vec{X_i})$, the

HD between two outputs $HD(\vec{Y_i}', \vec{Y_i})$ is the number of different bits in two output vectors, and the normalized HD of two functions can be calculated as follows:

$$HD(F, F') = \frac{1}{n} \sum_{i=1}^{n} \frac{HD(\vec{Y_i}', \vec{Y_i})}{|\vec{Y}|} \times 100\% \tag{6.1}$$

where n is the number of input vectors and $|\vec{Y}|$ is the number of output bits. Since the objective of functionality obfuscation is to restrain the attacker's ability to infer or utilize the correct functionality, $HD(F, F')$ approaching 50% is desirable, which indicates that the functionality of the reconstructed netlist deviates substantially away from the original functionality.

2) Layout Obfuscation. The security of 2.5D split fabrication rests upon the assumption that the attacker does not know the hidden portion (the trusted tier) and cannot infer it based on the exposed portion of design (the untrusted tier). Otherwise, the attacker can reconstruct the complete design and continue to conduct his attacks. To infer the hidden connections in the trusted tier, Rajendran *et al.* [27] proposed an attack called *proximity attack*. The attack is based on the observation that modern floorplanning and placement tool will place two connected pins closely in the untrusted tier so as to reduce the wire-length. However, the physical proximity of two connected pins leaks the information of the hidden connections. Since the layout information for each die is known to the attacker, he can iteratively connect an output pin in one die to its closet input pin in other die and thus reconstruct the circuit. Therefore, it's necessary to obfuscate the layout (by placing two connected pins far away) in order to prevent the leakage of the trusted tier in 2.5D split fabrication. *Proximity attack correctness* is a security metric that's used to quantify the

layout obfuscation level under the proximity attack. For 2.5D split fabrication, it's defined as the percentage of correct connections that are recovered by the proximity attack algorithm. Attack correctness approaching 0% is desirable for a secure layout design, which indicates that the attacker cannot infer the correct connections in the trusted tier.

Based on these two security metrics, the objective of our problem can be formulated as follows:

$$\text{minimize } |HD - 50\%| + Correctness \tag{6.2}$$

A secure design flow for 2.5D IC should achieve two objectives: a) incorrect functionality will be produced when incorrect connections are made between two partitions, *i.e.,* the HD between the functionalities of the original netlist and that of the netlist reconstructed using proximity attack algorithm approaches 50%; b) the proximity attack algorithm has 0% attack correctness.

6.5 Proposed Approach

6.5.1 Secure Partitioning

The partitioning phase plays a pivotal role in functionality obfuscation because it determines the hidden wires in the interposer layer. Fig. 6.4 illustrates a bi-partitioning of the c17 circuit from the ISCAS85 benchmark. The cut-wires are selected as the hidden wires that will be routed in the interposer layer. The resulting cut-wires have a significant impact on the incorrectness of output logic of

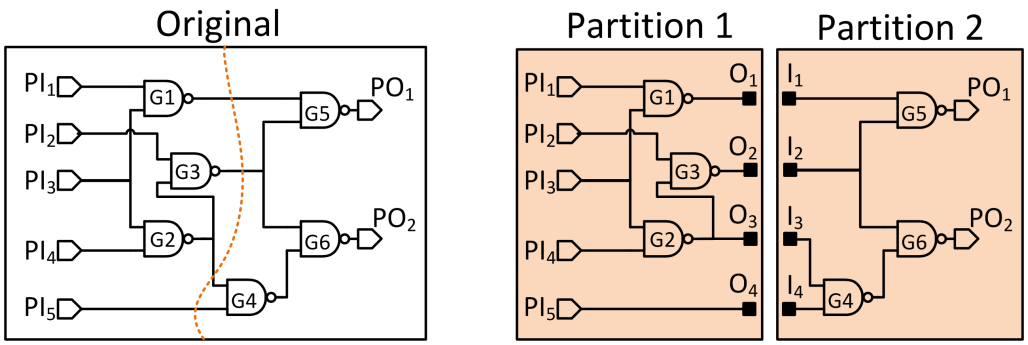


Figure 6.4: A bi-partitioning of the c17 circuit from ISCAS85 benchmark. The cut-wires are selected as the hidden wires that will be routed in the interposer.

a reconstructed netlist, because they decide whether faults can be generated and propagated to POs when incorrect connections are made.

To evaluate the capability of fault occurrence and fault propagation for a cut-set, we utilize the concepts of *controllability* and *observability*. Controllability and observability are two characteristics that are widely used in IC testing and security techniques. Controllability of an internal wire is the sensitivity of the wire w.r.t. the logic transition of PIs. It quantifies the ability to set a wire to some values (1 or 0) through PIs in order to activate a fault (due to incorrect reconnections) inside a circuit. Observability of a wire is the sensitivity of POs w.r.t. the logic transition of the internal wire. It quantifies the ability to observe faults in POs when the logic value of a wire inside the circuit is flipped. In order to activate and produce more faults when incorrect connections are made between two partitions, we aim at selecting cut-wires with high controllability and observability. The controllability $CTRL(w)$ and observability $OBS(w)$ of a wire w can be simulated and normalized to a value between 0 to 1 [34], where 1 indicates high controllability/observability.

The secure min-cut problem is to find a bi-partitioning with minimum cut-size while satisfying balance constraint and security constraint. The balance constraint ensures that two partitions have roughly equal sizes while the security constraint enforces that the controllability and observability of the wires in the cut-set are relatively large. The overall algorithm is based on Fiduccia-Mattheyses (FM) algorithm [103], a linear time heuristic approach to solve hypergraph bi-partitioning problem. The overall algorithm is as follows:

- Initialization: a balanced partitioning is randomly initialized so that two partitions have roughly equal sizes. PI pins and PO pins are separated into two partitions. Moreover, the controllability and observability of all wires are simulated.

- Maintenance: after initialization, the FM algorithm will iteratively move a gate that has the maximum cut-size drop from one block to another while maintaining the following two constraints:

  - Balance constraint: $\frac{|A(P_1)-A(P_2)|}{A(P_1)+A(P_2)} \leq B_{th}$, where $A(P_1)$, $A(P_2)$ are the sizes of two partitions P_1 and P_2, and B_{th} is a pre-defined balance threshold $0 \leq B_{th} \leq 1$.

  - Security constraint: if a gate's output wire w is in the cut-set and it has high controllability/observability $CTRL(w)+OBS(w) \geq S_{th}$, then don't move this gate. S_{th} is a pre-defined security threshold $0 \leq S_{th} \leq 2$.

- Termination: After all possible gate moves, the algorithm obtains a series

of moves that will result in the most cut-size reduction, which produces a new partitioning solution. The algorithm is continued until it cannot find a partitioning solution with smaller cut-size. Then, a final partitioning solution is generated and each gate is assigned to a partition.

We normally run the FM algorithm multiple times with random initial partitioning solution and select the best partitioning solution with minimum cut-size as the final solution.

6.5.2 Secure Placement

The placement phase is designed to thwart the proximity attack by obfuscating the layouts of the untrusted tier so as to mislead the proximity attack algorithm into making wrong connections. The goal of secure placement is to minimize the area, intra-chip wire-length, inter-chip wire-length and proximity-attack correctness.

The secure 2.5D IC placement algorithm is based on a B*-tree and simulated annealing (SA) based 2.5D IC placement algorithm proposed by Ho et al. [104]. Fig. 6.5 shows the overall flow of the secure placement algorithm. The placement algorithm utilized the B*-tree to represent a compacted placement solution [105]. Two B*-trees are firstly constructed to represent the geometry relationship for all gates and I/O pins of two sub-netlists. A node in the B*-tree represents a gate or an I/O pin and each B*-tree represents a compacted placement for one sub-netlist. Using two B*-trees allows us to optimize the placement of two sub-netlists simultaneously. Three node perturbation operations are implemented in the SA

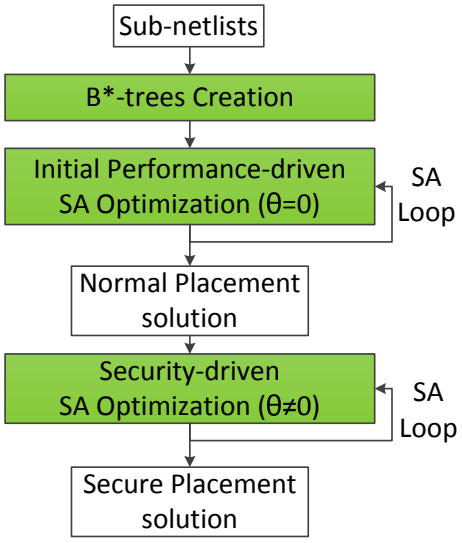


Figure 6.5: B*-tree and SA based secure placement algorithm flow [34].

process, as defined in [104]:

- Rotation: the rotation of a gate or I/O pin.
- Move within a B*-tree: the move of a gate or an I/O pin within same die.
- Swap two nodes within a B*-tree: the swap of two gates or I/O pin within the same die.

After perturbation, two new B*-trees are formed and corresponding compact placements for two chips can be obtained. Based on the placement solution, we can calculate its area, inter-chip wire-length, intra-chip wire-length and perform the proximity attack to obtain the proximity-attack correctness. The proximity attack algorithm will take the coordinates of I/O buffers and two sub-netlists as input data, generate a reconnection and calculate its correctness.

The cost function of SA optimization is defined as:

$$\Phi = \alpha \times Area + \beta \times WL_{intra} + \gamma \times WL_{inter} + \theta \times Correctness \tag{6.3}$$

where α, β, γ and θ are user-specified weighting parameters, $Area$ is the total area of two chips, WL_{intra} is the total intra-chip wire-length, WL_{inter} is the total inter-chip wire-length and $Correctness$ is the proximity-attack correctness obtained by proximity attack algorithm. Two SA processes are used to generate an effective and secure placement, as shown in Fig. 6.5. The first performance-driven ($\theta = 0$) SA process creates an initial placement that has optimized area and total wire-length. Based on this initial placement, the second security-driven ($\theta \neq 0$) SA process attempts to trade-off between performance and security.

6.6 Experiments and Results

6.6.1 Experiment Setup

We examined our proposed design flow on 8 combinational circuits from ISCAS-85 benchmarks [106] and ITC'99 benchmarks [107]. Table 6.1 shows the benchmark details. A TSMC-180nm standard cell library is used in placement phase. The controllability and observability values are computed using 1000 random input vectors for each benchmark. Notice that using more input vectors can increase the accuracy of estimation of these two values, but it will take longer computation time. In the results shown below, we find that 1000 random input vectors are enough. The HD between reconstructed netlist and original netlist is determined by 1000 random in-

put vectors for each benchmark, which are not necessarily the same as the previous input vectors used for computing controllability and observability.

In the partitioning phase, we set the balance threshold B_{th} to be 0.1. The security threshold S_{th} varies from 1.01 to 1.3 for different benchmarks, which are the ones that lead to the best tradeoff between cut-size and HD during simulation. To study the relation between S_{th}, HD and cut-size, and to find the best S_{th}, we run the partitioning algorithm with a set of security threshold values and calculate the HD between randomly reconstructed netlist and original netlist. The impact of S_{th} on cut-size and HD are shown in Fig. 6.6 and Fig. 6.7. As S_{th} increases, the cut-size and HD decreases for all benchmarks since a large S_{th} indicates a loose constraint, which means that only a few wires with high controllability and observability will be locked in cut-set during partitioning. Based on this simulation results, we define **Secure Partitioning (SecPart)** as the partitioning with S_{th} that makes HD larger than 40%. Also, we define **Normal Partitioning (NormPart)** as the partitioning that doesn't consider the security constraint. Since the security constraint will increase the cut-size, one interesting question to ask is that whether we can achieve the same HD by using the NormPart algorithm but enforce that the partitioning result should have the same cut-size as the one produced by the SecPart algorithm. We implement the normal partitioning with a cut-size lower-bound that's set to the cut-size of secure partitioning solution (denoted as **NormPart_LargeSize**). All the partitioning algorithms are run with 1000 random initializations and the best solution with minimum cut-size is selected as the final solution.

In the placement phase, in order to determine the optimal weights for the cost

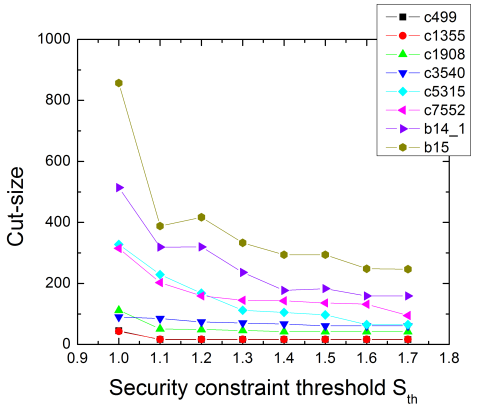


Figure 6.6: Impact of security constraint S_{th} on cut-size

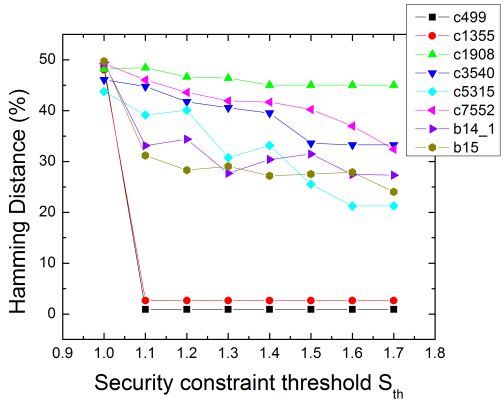


Figure 6.7: Impact of security constraint S_{th} on HD

function, we test different setups on all benchmarks and define the setup $\alpha = 0.2, \beta = 0.7, \gamma = 0.1, \theta = 0$ as **Normal Placement (NormPlace)** since it can obtain a relatively optimal result in area and total wire-length. For **Secure Placement (SecPlace)**, we increase θ to 0.05 and decrease γ to 0.05. The reason we decrease the inter-chip wire-length weight γ is that we want to weaken the correlation between connectedness and layout proximity of inter-die I/O buffers.

6.6.2 Results

Table 6.1 shows the partitioning results of three partitioning settings, namely NormPart, SecPart and NormPart_LargeSize. Comparing NormPart and SecPart, we can see that HD increases from 13.24% to 46.35% on average. This is because that we have enforced the security constraint to select enough cut-wires with high controllability/observability so that more faults will be produced for an incorrectly reconstructed netlist. However, the security constraint inevitably increases the cut-size

Table 6.1: Benchmark information and partitioning results of NormPart, NormPart_LargeCutsize, and SecPart.

Benchmark	#PIs	#POs	#Gates	NormPart		NormPart_LargeSize		SecPart	
				Cutsize	HD	Cutsize	HD	Cutsize	HD
c499	41	32	202	16	0.86%	45	48.20%	45	49.84%
c1355	41	32	546	16	7.08%	43	45.01%	43	49.96%
c1908	33	25	880	35	20.09%	37	33.46%	37	44.79%
c3540	50	22	1669	57	32.82%	74	33.28%	74	42.67%
c5315	178	123	2307	30	8.65%	168	19.13%	168	41.07%
c7552	207	108	3512	25	5.46%	155	14.34%	155	48.55%
b14_1	277	299	4048	99	14.85%	386	19.14%	386	44.76%
b15	485	519	7022	168	16.14%	625	27.76%	625	49.12%
Average	-	-	-	56	13.24%	192	30.04%	192	46.35%

of secure partitioning. As seen, the cut-size of SecPart is 3.4× the cut-size of NormPart on average. The extra cut-wires will increase the performance overhead such as area and wire-length in the placement phase. Comparing NormPart_LargeSize and SecPart, we can see that although these two cases have the same cut-size, SecPart can ensure 46.35% HD while NormPart_LargeSize can only achieve 30.04% HD. Therefore, with security constraint, the secure partitioning algorithm can achieve 50% HD more efficiently.

In order to evaluate the overall design flow, we compare four possible combinations, namely NormPart + NormPlace, NormPart + SecPlace, SecPart + NormPlace and SecPart + SecPlace in terms of attack correctness, Hamming distance, area and total wire-length.

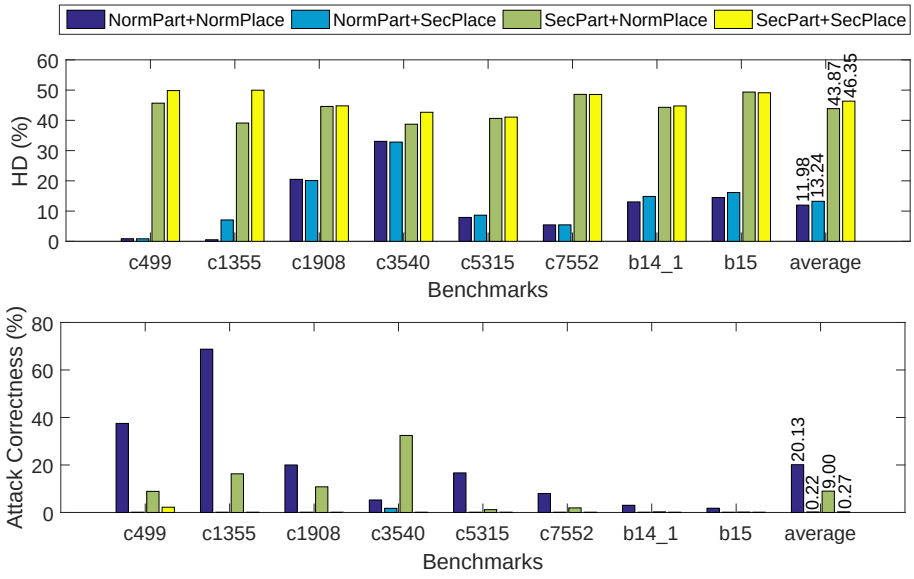


Figure 6.8: HD and attack correctness for four design flows.

Fig. 6.8 shows the correctness and HD of proximity attack for four cases. For 'NormPart+NormPlace', the attack correctness is 20.13% and HD is only 11.98% because no security constraint is enforced in the NormPart to conceal the functionality, and the NormPlace doesn't minimize attack correctness during SA optimization. When SecPlace is performed on NormPart, we can see that the attack correctness is limited to 0.22%, and the HD increases to 13.24%, which is still far below 50% as a large amount of functionality is exposed due to the normal min-cut partitioning. For the case 'SecPart+NormPlace', the HD increases to 43.87%, which proves the effectiveness of SecPart in concealing the functionality of a design. Finally, if we perform SecPlace on top of SecPart, compared to the 'SecPart+NormPlace' case, the attack correctness is reduced from 9.00% to 0.27% and the HD increases from 43.87% to 46.35%. The 'SecPart+SecPlace' design flow achieves the optimal security among four design flows. Overall, the SecPart algorithm is capable of approaching

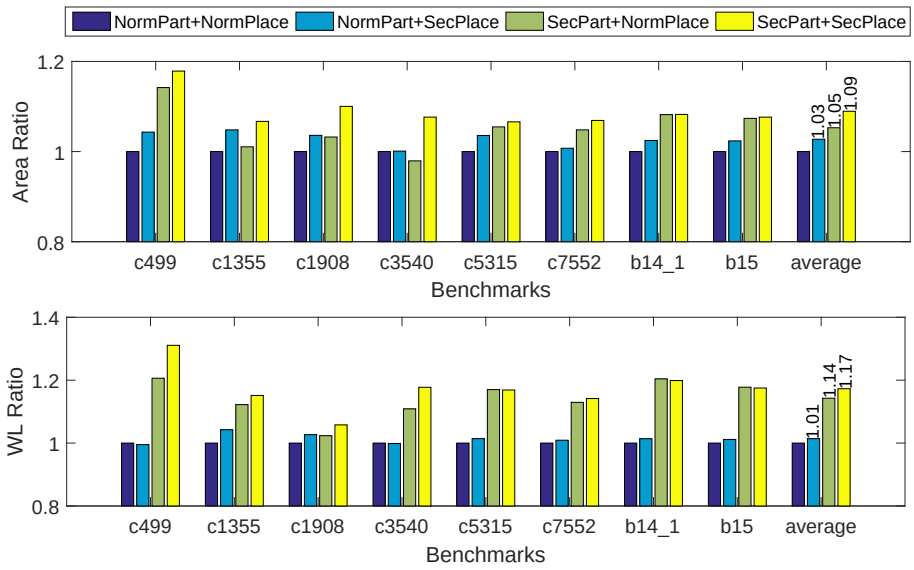


Figure 6.9: Area and total wire-length overhead for four design flows.

50% HD, and the SecPlace algorithm can effectively achieve 0% attack correctness.

Fig. 6.9 shows the area and total wire-length for four cases. Chip area and wire-length are two metrics that are commonly used to evaluate the performance of gate placement algorithm [104]. The 'NormPart+NormPlace' design flow is considered as a baseline for calculating overheads. As seen, the main overheads come from the SecPart, as it requires a larger cut-set than NormPart to ensure 50% HD, which will inevitably increase the area and wire-length. The average overheads for SecPart are 5.29% on the area and 14.27% on total wire-length. The SecPlace algorithm contributes to additional overhead because it perturbs the layout geometry to produce a placement with 0% attack correctness. Overall, the average overheads for 'SecPart+SecPlace' design flow are 8.95% on the area and 17.27% on total wire-length.

Finally, we study the tradeoff between a more gradual degradation in cut-

Table 6.2: Tradeoff between HD, area and total wire-length on the c7552 circuit

Placement	Cut-size	Area (μm^2)	Area Overhead	WL (μm)	WL Overhead	Correctness	HD
Normal	25	272053	0%	1048730	0%	8.00%	5.45%
Secure	25	274075	0.74%	1058320	0.91%	0%	5.46%
	65	283525	4.22%	1106230	5.48%	0%	20.31%
	105	280691	3.18%	1133420	8.08%	0%	31.34%
	145	281941	3.63%	1169020	11.47%	0%	38.53%
	155	290849	6.91%	1197160	14.15%	0%	48.55%

size and the security obtained, since the large cut-size due to the secure min-cut algorithm is the main source of performance overhead. Our objective is to study the impact of security constraint threshold S_{th} on cut-size and HD, and perform tradeoff analysis between HD and performance.

We use the c7552 circuit to demonstrate the tradeoff between HD and performance due to its large cut-size increase between normal and secure min-cut. The cut-size decreases from 155, which is the cut-size when the security threshold S_{th} is set to 1.2. By gradually increasing the security threshold S_{th} to 2, the algorithm will produce solutions with less cut-size. For a set of partitioning solutions with less cut-size, we perform secure placement and proximity attack and calculate the HD between the reconstructed netlist and the original netlist.

Table 6.2 shows the tradeoff between security and performance. NormPart+NormPlace is used as a baseline for calculating overheads. Our approach is capable of generating a tradeoff between security and other performance parameters. For performance-

driven designs, we can increase the security threshold S_{th} to produce a partitioning with less cut-size.

6.7 Conclusion

In this chapter, we present a security-aware physical design flow for 2.5D IC to counter IP piracy in outsourced fabrication. In partitioning phase, we propose a secure partitioning algorithm that can generate a bipartition with 46.35% HD on applying incorrect reconnection. In placement phase, we equip an SA-based placement algorithm with proximity attack correctness evaluation which limits the correctness of proximity attack to 0.27%. Experiment results have shown that our approach is capable of generating layout files that are fully obfuscated and resilient to proximity attack at a low performance cost. In addition, a tradeoff between security and other performance parameters has been shown which could be utilized by chip designer to estimate the performance overhead for a certain security improvement.

Chapter 7: Conclusion and Future Research Directions

In Chapter 1, we summarized various hardware-based attacks in different phases of an IC's life cycle. The main goal of our work is to develop design obfuscation techniques to enhance the security of outsourced IC designs that are fabricated in possibly untrustworthy foundries. These techniques can help building trust between IC design companies and fabrication foundries so as to create a win-win scenario. In Chapter 2, we provided background on IC supply chain attacks and discussed countermeasures such as logic locking and split fabrication.

In Chapter 3 and Chapter 4, we investigated emerging attacks on logic locking techniques and proposed new locking techniques that can thwart such attacks in a provably-secure manner. Specifically, Chapter 3 presented a circuit block called Anti-SAT that can mitigate the SAT attack on logic locking. We show that the number of iterations required by the SAT attack to reveal the correct key in the Anti-SAT block is an exponential function of the key-size, thereby making the attack computationally infeasible. The Anti-SAT block is integrated into a locked circuit to increase its resistance to the SAT attack. A unified obfuscation technique has been proposed to protect the Anti-SAT block from potential removal attacks. Based on Anti-SAT, Chapter 4 investigated both attack and defense methodologies for locked

neural chips. We proposed new attack methodology that exploits existing AppSAT attack and neural model fine-tuning in effective ways. To counter this attack, we proposed a secure locking scheme based on a co-design of the locking infrastructure (called Strong Anti-SAT) and the functional modules. These proposed techniques were validated with rigorous proof and extensive experiments.

In Chapter 5, we explored new opportunities in obfuscating the timing profile of a circuit design and proposed the delay locking technique. For delay locking, the key to a locked circuit not only determines its functionality, but also its timing profile. A functionality-correct but timing-incorrect key will result in timing violations, hence making the circuit malfunction. Such locking scheme can thwart many functionality-oriented attacks (*e.g.,* SAT attack) because they cannot be utilized to decipher a timing-correct key.

In Chapter 6, we studied the security implications of 3D/2.5D ICs and proposed a security-aware physical design flow for 2.5D IC to counter supply chain attacks in outsourced fabrication. Simulation results show that our design flow is effective for producing secure chip layouts for outsourcing whose original netlist and functionality cannot be reverse-engineered based on the layout-geometry information. A trade-off between security and other performance parameters has been shown which could be utilized by chip designer to estimate the performance overhead for a certain security improvement.

7.1 Future Work

In this dissertation, we have developed various unconventional design obfuscation techniques to enhance the hardware security. There are still many possibilities for improvement. The following summarizes potential future research directions to extend the research work presented in this dissertation.

7.1.1 Security in Emerging Hardware Designs

In recent years, researchers have proposed numerous innovative hardware designs and architectures that are different from conventional circuit designs. Neural network chip, as discussed in Chapter 4, is one example of the emerging hardware. Another emerging hardware design that has not been discussed in this dissertation is called approximate computing circuit. Unlike conventional circuits, the approximate circuit is a hardware that trades computation correctness/quality for reduced power and area. By design, the approximate circuit can output incorrectly at a tolerably low frequency and magnitude, but it can save a significant amount of power and area. Existing research on approximate computing focuses mainly on improving its performance and reliability. However, the security aspect of these emerging hardware is not fully investigated. We are interested in studying the security and vulnerability implications of these technologies. We plan to investigate the effectiveness of existing supply chain attacks on these new hardware designs and look for new countermeasures to improve their security. New attack model assumptions and new security metrics shall also be investigated for evaluating the security level

of these novel hardware designs and architectures.

7.1.2 Parametric Locking

Conventional logic locking techniques purely focus on the Boolean logic level. Thus, they are subject to logic-analysis based attacks such as the SAT attack. In Chapter 5, we explored new obfuscation possibility by obfuscating the timing profile of a circuit. Moving forward, we plan to investigate new circuit locking techniques that obfuscate other parameters of a circuit such as power, heat dissipation and so on. We refer this type of locking as parametric locking. By doing so, a key to a locked circuit would not only determines its functionality, but also its parametric behavior. A functionality-correct but parameter-incorrect key will result in parametric violations (such as power or temperature violations) and thus make the circuit malfunction or deviate from its normal behavior. The advantage of parametric locking is that it's immune to logic-analysis based attack because the parametric locking does not depend on the Boolean logic. Besides, analog components (such as power management, heat dissipation) are parts of hardware IPs that would be attractive to attackers, but corresponding obfuscation techniques for these components are not fully investigated in existing literature. Obfuscating these subsidiary circuit components is an interesting future research direction.

7.1.3 3D IC Security

While 3D integration is initially developed to overcome the obstacles in device miniaturization, it has presented various security advantages in different security techniques and applications. The stacking structure and high-density nature of 3D integration offer a natural defense for side-channel attacks as it adds significantly more complexity for an attacker to extract a meaningful signal from the complicated background noise. Moreover, reverse-engineering becomes challenging since hardware designs can be protected inside the firmly stacked substrates. In addition, with 3D heterogeneous integration, novel non-CMOS security primitives can be integrated with CMOS processor to achieve a comprehensive system with optimal security and performance. As for future works, we would like to investigate design techniques for 3D ICs that utilize these unique advantages to thwart hardware-based attacks. With the effort made in 3D IC security characterization and modeling, future chip designers can take security into consideration at an early phase of the design while optimizing the chip for performance and power.

Bibliography

[1] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown. *arXiv preprint arXiv:1801.01207*, 2018.

[2] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. *arXiv preprint arXiv:1801.01203*, 2018.

[3] Navin Shenoy. Firmware updates and initial performance data for data center systems. `https://newsroom.intel.com/news/firmware-updates-and-initial-performance-data-for-data-center-systems/`.

[4] Sally Adee. The hunt for the kill switch. *iEEE SpEctrum*, 45(5):34–39, 2008.

[5] A Rawnsley. fishy chips: Spies want to hack-proof circuits. *Wired, Jun*, 24, 2011.

[6] Sergei Skorobogatov and Christopher Woods. Breakthrough silicon scanning discovers backdoor in military chip. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 23–40. Springer, 2012.

[7] IHS Technology. Ihs technology press release: Top 5 most counterfeited parts represent a $169 billion potential challenge for global semiconductor industry. 2012.

[8] Mohammad Tehranipoor and Farinaz Koushanfar. A survey of hardware trojan taxonomy and detection. 2010.

[9] Kaushik Vaidyanathan, Bishnu P Das, and Larry Pileggi. Detecting reliability attacks during split fabrication using test-only beol stack. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6. ACM, 2014.

[10] Randy Torrance and Dick James. The state-of-the-art in IC reverse engineering. In *Cryptographic Hardware and Embedded Systems-CHES 2009*, pages 363–381. Springer, 2009.

[11] Ujjwal Guin, Ke Huang, Daniel DiMase, John M Carulli, Mohammad Tehranipoor, and Yiorgos Makris. Counterfeit integrated circuits: a rising threat in the global semiconductor supply chain. *Proceedings of the IEEE*, 102(8):1207–1228, 2014.

[12] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in CryptologyCRYPTO99*, pages 388–397. Springer, 1999.

[13] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems-CHES 2004*, pages 16–29. Springer, 2004.

[14] David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[15] Daniel J Bernstein. Cache-timing attacks on AES, 2005.

[16] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In *Smart Card Programming and Security*, pages 200–210. Springer, 2001.

[17] Dakshi Agrawal, Bruce Archambeault, Josyula R Rao, and Pankaj Rohatgi. The EM sidechannel (s). In *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 29–45. Springer, 2003.

[18] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *Cryptographic Hardware and Embedded SystemsCHES 2001*, pages 251–261. Springer, 2001.

[19] Farinaz Koushanfar. Hardware metering: A survey. In *Introduction to Hardware Security and Trust*, pages 103–122. Springer, 2012.

[20] Jarrod A Roy, Farinaz Koushanfar, and Igor L Markov. Epic: Ending piracy of integrated circuits. In *Proceedings of the conference on Design, Automation and Test in Europe*, pages 1069–1074. ACM, 2008.

[21] Richard Wayne Jarvis and Michael G McIntyre. Split manufacturing method for advanced semiconductor circuits, March 27 2007. US Patent 7,195,931.

[22] Jeyavijayan Rajendran, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. Security analysis of logic obfuscation. In *Proceedings of the 49th Annual Design Automation Conference*, pages 83–89. ACM, 2012.

[23] Stephen M Plaza and Igor L Markov. Solving the third-shift problem in IC piracy with test-aware logic locking. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 34(6):961–971, 2015.

[24] Pramod Subramanyan, Sayak Ray, and Sharad Malik. Evaluating the security of logic encryption algorithms. In *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on*, pages 137–143. IEEE, 2015.

[25] Muhammad Yasin, Samah Mohamed Saeed, Jeyavijayan Rajendran, and Ozgur Sinanoglu. Activation of logic encrypted chips: Pre-test or post-test? In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 139–144. IEEE, 2016.

[26] Muhammad Yasin, Bodhisatwa Mazumdar, Sk Subidh Ali, and Ozgur Sinanoglu. Security analysis of logic encryption against the most effective side-channel attack: Dpa. In *2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, pages 97–102. IEEE, 2015.

[27] Jeyavijayan JV Rajendran, Ozgur Sinanoglu, and Ramesh Karri. Is split manufacturing secure? In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1259–1264. EDA Consortium, 2013.

[28] Yujie Wang, Pu Chen, Jiang Hu, and Jeyavijayan JV Rajendran. The cat and mouse in split manufacturing. In *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2016.

[29] Jonathon Magaña, Daohang Shi, and Azadeh Davoodi. Are proximity attacks a threat to the security of split manufacturing of integrated circuits? In *Proceedings of the 35th International Conference on Computer-Aided Design*, page 90. ACM, 2016.

[30] Ping-Lin Yang and Malgorzata Marek-Sadowska. Making split-fabrication more secure. In *Proceedings of the 35th International Conference on Computer-Aided Design*, page 91. ACM, 2016.

[31] Yang Xie and Ankur Srivastava. Mitigating sat attack on logic locking. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 127–146. Springer, 2016.

[32] Yang Xie and Ankur Srivastava. Anti-sat: Mitigating sat attack on logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[33] Yang Xie and Ankur Srivastava. Delay locking: Security enhancement of logic locking against ic counterfeiting and overproduction. In *Proceedings of the 54th Annual Design Automation Conference 2017*, page 9. ACM, 2017.

[34] Yang Xie, Chongxi Bao, and Ankur Srivastava. Security-aware design flow for 2.5 d ic technology. In *Proceedings of the 5th International Workshop on Trustworthy Embedded Devices*, pages 31–38. ACM, 2015.

[35] Yang Xie, Chongxi Bao, and Ankur Srivastava. 3d/2.5 d ic-based obfuscation. In *Hardware Protection through Obfuscation*, pages 291–314. Springer, 2017.

[36] Yang Xie, Chongxi Bao, and Ankur Srivastava. Security-aware 2.5 d integrated circuit design flow against hardware ip piracy. *Computer*, 50(5):62–71, 2017.

[37] Yang Xie, Chongxi Bao, Caleb Serafy, Tiantao Lu, Ankur Srivastava, and Mark Tehranipoor. Security and vulnerability implications of 3d ics. *IEEE Transactions on Multi-Scale Computing Systems*, 2(2):108–122, 2016.

[38] Yang Xie, Chongxi Bao, Yuntao Liu, and Ankur Srivastava. 2.5 d/3d integration technologies for circuit obfuscation. In *Microprocessor and SOC Test and Verification (MTV), 2016 17th International Workshop on*, pages 39–44. IEEE, 2016.

[39] Yu-Wei Lee and Nur A Touba. Improving logic obfuscation via logic cone analysis. In *2015 16th Latin-American Test Symposium (LATS)*, pages 1–6. IEEE, 2015.

[40] Abhishek Chakraborty, Yang Xie, and Ankur Srivastava. Template attack based deobfuscation of integrated circuits. In *Computer Design (ICCD), 2017 IEEE International Conference on*, pages 41–44. IEEE, 2017.

[41] Kaveh Shamsi, Meng Li, Travis Meade, Zheng Zhao, David Z Pan, and Yier Jin. Appsat: Approximately deobfuscating integrated circuits. In *Hardware Oriented Security and Trust (HOST), 2017 IEEE International Symposium on*, pages 95–100. IEEE, 2017.

[42] Yuanqi Shen and Hai Zhou. Double dip: Re-evaluating security of logic encryption algorithms. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pages 179–184. ACM, 2017.

[43] Hai Zhou, Ruifeng Jiang, and Shuyu Kong. Cycsat: Sat-based attack on cyclic logic encryptions. In *Computer-Aided Design (ICCAD), 2017 IEEE/ACM International Conference on*, pages 49–56. IEEE, 2017.

[44] Mohamed El Massad, Siddharth Garg, and Mahesh Tripunitara. Reverse engineering camouflaged sequential circuits without scan access. In *Computer-Aided Design (ICCAD), 2017 IEEE/ACM International Conference on*, pages 33–40. IEEE, 2017.

[45] Rajat Subhra Chakraborty and Swarup Bhunia. Harpoon: an obfuscation-based soc design methodology for hardware protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(10):1493–1502, 2009.

[46] Rajat Subhra Chakraborty and Swarup Bhunia. Rtl hardware ip protection using key-based control and data flow obfuscation. In *VLSI Design, 2010. VLSID'10. 23rd International Conference on*, pages 405–410. IEEE, 2010.

[47] Jeyavijayan Rajendran, Huan Zhang, Chi Zhang, Garrett S Rose, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. Fault analysis-based logic encryption. *Computers, IEEE Transactions on*, 64(2):410–424, 2015.

[48] Bicky Shakya, Navid Asadizanjani, Domenic Forte, and Mark Tehranipoor. Chip editor: leveraging circuit edit for logic obfuscation and trusted fabrication. In *Proceedings of the 35th International Conference on Computer-Aided Design*, page 30. ACM, 2016.

[49] James B Wendt and Miodrag Potkonjak. Hardware obfuscation using PUF-based logic. In *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*, pages 270–277. IEEE Press, 2014.

[50] Alex Baumgarten, Akhilesh Tyagi, and Joseph Zambreno. Preventing IC piracy using reconfigurable logic barriers. *IEEE Design & Test of Computers*, 2010.

[51] Soroush Khaleghi, Kai Da Zhao, and Wenjing Rao. IC piracy prevention via design withholding and entanglement. In *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*, pages 821–826. IEEE, 2015.

[52] Bao Liu and Brandon Wang. Embedded reconfigurable logic for ASIC design obfuscation against supply chain attacks. In *Proceedings of the conference on Design, Automation and Test in Europe*, page 243. European Design and Automation Association, 2014.

[53] Sophie Dupuis, Papa-Sidi Ba, Giorgio Di Natale, M-L Flottes, and Bruno Rouzeyre. A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans. In *On-Line Testing Symposium (IOLTS), 2014 IEEE 20th International*, pages 49–54. IEEE, 2014.

[54] Kaushik Vaidyanathan, Bishnu P Das, Ekin Sumbul, Renzhi Liu, and Larry Pileggi. Building trusted ics using split fabrication. In *Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on*, pages 1–6. IEEE, 2014.

[55] Karthikeyan Vaidyanathan, Renzhi Liu, Ekin Sumbul, Qiuling Zhu, Franz Franchetti, and Larry Pileggi. Efficient and secure intellectual property (IP) design with split fabrication. In *Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on*, pages 13–18. IEEE, 2014.

[56] Meenatchi Jagasivamani, Peter Gadfort, Michel Sika, Michael Bajura, and Michael Fritze. Split-fabrication obfuscation: Metrics and techniques. In *Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on*, pages 7–12. IEEE, 2014.

[57] Lang Feng, Yujie Wang, Jiang Hu, Wai-Kei Mak, and Jeyavijayan Rajendran. Making split fabrication synergistically secure and manufacturable. In

Computer-Aided Design (ICCAD), 2017 IEEE/ACM International Conference on, pages 321–328. IEEE, 2017.

[58] Tezzaron. 3D-ICs and integrated circuit security. http://www.tezzaron.com/about/papers/3D-ICs_and_Integrated_Circuit_Security.pdf, 2008.

[59] Jonathan Valamehr, Timothy Sherwood, Ryan Kastner, David Marangoni-Simonsen, Ted Huffmire, Cynthia Irvine, and Timothy Levin. A 3-d split manufacturing approach to trustworthy system development. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(4):611–615, 2013.

[60] Paul Franzon, Steve Lipa, and Lisa McIlrath. Trusted fabrication through 3d integration. Technical report, North Carolina State University Raleigh United States, 2017.

[61] Patrick Dorsey. Xilinx stacked silicon interconnect technology delivers breakthrough fpga capacity, bandwidth, and power efficiency. *Xilinx White Paper: Virtex-7 FPGAs*, 2010.

[62] Frank Imeson, Ariq Emtenan, Siddharth Garg, and Mahesh V Tripunitara. Securing computer hardware using 3d integrated circuit (ic) technology and split manufacturing for obfuscation. In *USENIX Security Symposium*, pages 495–510, 2013.

[63] Johann Knechtel, Ozgur Sinanoglu, Ibrahim Abe M Elfadel, Jens Lienig, and Cliff CN Sze. Large-scale 3d chips: Challenges and solutions for design automation, testing, and trustworthy integration. *IPSJ Transactions on System LSI Design Methodology*, 10:45–62, 2017.

[64] Aarti Gupta and Pranav Ashar. Integrating a boolean satisfiability checker and bdds for combinational equivalence checking. In *VLSI Design, 1998. Proceedings., 1998 Eleventh International Conference on*, pages 222–225. IEEE, 1998.

[65] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu, and Jeyavijayan Rajendran. Security analysis of anti-sat. Cryptology ePrint Archive, Report 2016/896, 2016. `http://eprint.iacr.org/2016/896`.

[66] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu, and Jeyavijayan Rajendran. Removal attacks on logic locking and camouflaging techniques. *IEEE Transactions on Emerging Topics in Computing*, 2017.

[67] Armin Biere. Lingeling, plingeling and treengeling entering the sat competition 2013. *Proceedings of SAT Competition*, 2013, 2013.

[68] Muhammad Yasin, Jeyavijayan Rajendran, Ozgur Sinanoglu, and Ramesh Karri. On improving the security of logic locking. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 2015.

[69] Helion Technology. High Performance AES (Rijndael) cores for ASIC. `http://www.heliontech.com/downloads/aes_asic_helioncore.pdf`, 2015.

[70] Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan JV Rajendran, and Ozgur Sinanoglu. Sarlock: Sat attack resistant logic locking. In *Hardware Oriented Security and Trust (HOST), 2016 IEEE International Symposium on*, pages 236–241. IEEE, 2016.

[71] Xiaolin Xu, Bicky Shakya, Mark M Tehranipoor, and Domenic Forte. Novel bypass attack and bdd-based tradeoff analysis against all known logic locking attacks. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 189–210. Springer, 2017.

[72] Kaveh Shamsi, Meng Li, Travis Meade, Zheng Zhao, David Z Pan, and Yier Jin. Cyclic obfuscation for creating sat-unresolvable circuits. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pages 173–178. ACM, 2017.

[73] Muhammad Yasin, Abhrajit Sengupta, Benjamin Carrion Schafer, Yiorgos Makris, Ozgur Sinanoglu, and Jeyavijayan JV Rajendran. What to lock?: Functional and parametric locking. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pages 351–356. ACM, 2017.

[74] Mohamed El Massad, Siddharth Garg, and Mahesh V Tripunitara. Integrated circuit (ic) decamouflaging: Reverse engineering camouflaged ics within minutes. In *NDSS*, 2015.

[75] Cunxi Yu, Xiangyu Zhang, Duo Liu, Maciej Ciesielski, and Daniel Holcomb. Incremental sat-based reverse engineering of camouflaged logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

[76] Meng Li, Kaveh Shamsi, Travis Meade, Zheng Zhao, Bei Yu, Yier Jin, and David Z Pan. Provably secure camouflaging strategy for ic protection. In *Proceedings of the 35th International Conference on Computer-Aided Design*, page 28. ACM, 2016.

[77] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu, and Jeyavijayan Rajendran. Camoperturb: secure ic camouflaging for minterm protection. In *Proceedings of the 35th International Conference on Computer-Aided Design*, page 29. ACM, 2016.

[78] Tianshi Chen and et al. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM Sigplan Notices*, volume 49, pages 269–284. ACM, 2014.

[79] Song Han and et al. Eie: efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 243–254. IEEE Press, 2016.

[80] Yu-Hsin Chen and et al. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 367–379. IEEE, 2016.

[81] Norman Jouppi, Cliff Young, Nishant Patil, David Patterson, et al. In-datacenter performance analysis of a tensor processing unit^{TM}. In *44th International Symposium on Computer Architecture (ISCA)*, 2017.

[82] Vinay K Chippa, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *Proceedings of the 50th Annual Design Automation Conference*, page 113. ACM, 2013.

[83] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[84] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[85] Lili Song, Ying Wang, Yinhe Han, Xin Zhao, Bosheng Liu, and Xiaowei Li. C-brain: A deep learning accelerator that tames the diversity of cnns through adaptive data-level parallelization. In *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2016.

[86] Philipp Gysel and et al. Hardware-oriented approximation of convolutional neural networks. *arXiv preprint arXiv:1604.03168*, 2016.

[87] Stephen A. Cook and et al. Finding hard instances of the satisfiability problem: A survey. pages 1–17. American Mathematical Society, 1997.

[88] Stefan Schoenmackers and et al. Satisfy this: An attempt at solving prime factorization using satisfiability solvers. 2004.

[89] Jeng-Liang Tsai, DongHyun Baik, Charlie Chung-Ping Chen, and Kewal K Saluja. A yield improvement methodology using pre-and post-silicon statistical clock scheduling. In *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, pages 611–618. IEEE Computer Society, 2004.

[90] Vishal Khandelwal and Ankur Srivastava. Variability-driven formulation for simultaneous gate sizing and postsilicon tunability allocation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2008.

[91] S Kim, J Kim, and S-Y Hwang. New path balancing algorithm for glitch power reduction. *IEE Proceedings-Circuits, Devices and Systems*, 148(3):151–156, 2001.

[92] Chung-Ping Chen, Chris CN Chu, and DF Wong. Fast and exact simultaneous gate and wire sizing by lagrangian relaxation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(7):1014–1025, 1999.

[93] Philip Garrou, Christopher Bower, and Peter Ramm. *Handbook of 3d integration: volume 1-technology and applications of 3D integrated circuits*. John Wiley & Sons, 2011.

[94] Tiantao Lu, Caleb Serafy, Zhiyuan Yang, Sandeep Samal, Sung Kyu Lim, and Ankur Srivastava. Tsv-based 3d ics: Design methods and tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[95] Kirk Saban. Xilinx stacked silicon interconnect technology delivers breakthrough fpga capacity, bandwidth, and power efficiency. *Xilinx, White Paper*, 2011.

[96] Yang Xie, Chongxi Bao, Caleb Serafy, Tiantao Lu, Ankur Srivastava, and Mark Tehranipoor. Security and vulnerability implications of 3d ics. *IEEE Transactions on Multi-Scale Computing Systems*, 2016.

[97] Kan Xiao, Domenic Forte, and Mark Mohammed Tehranipoor. Efficient and secure split manufacturing via obfuscated built-in self-authentication. In *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on*, pages 14–19. IEEE, 2015.

[98] Seetharam Narasimhan, Wen Yueh, Xinmu Wang, Saibal Mukhopadhyay, and Swarup Bhunia. Improving IC security against trojan attacks through integration of security monitors. *IEEE Design & Test of Computers*, 29(5):37–46, 2012.

[99] Michael Bilzor. 3D execution monitor (3D-EM): Using 3D circuits to detect hardware malicious inclusions in general purpose processors. In *Proceedings of the 6th International Conference on Information Warfare and Security*, page 288. Academic Conferences Limited, 2011.

[100] Masoud Rostami, Farinaz Koushanfar, Jeyavijayan Rajendran, and Ramesh Karri. Hardware security: Threat models and metrics. In *Proceedings of the International Conference on Computer-Aided Design*, pages 819–823. IEEE Press, 2013.

[101] Ronald P Cocchi, James P Baukus, Lap Wai Chow, and B Jiangyun Wang. Circuit camouflage integration for hardware ip protection. In *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*, pages 1–5. IEEE, 2014.

[102] Jeyavijayan Rajendran, Ozgur Sinanoglu, and Ramesh Karri. Regaining trust in vlsi design: Design-for-trust techniques. *Proceedings of the IEEE*, 102(8):1266–1282, 2014.

[103] Charles M Fiduccia and Robert M Mattheyses. A linear-time heuristic for improving network partitions. In *Design Automation, 1982. 19th Conference on*, pages 175–181. IEEE, 1982.

[104] Yuan-Kai Ho and Yao-Wen Chang. Multiple chip planning for chip-interposer codesign. In *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, pages 1–6. IEEE, 2013.

[105] Yun-Chih Chang, Yao-Wen Chang, Guang-Ming Wu, and Shu-Wei Wu. B*-trees: a new representation for non-slicing floorplans. In *Proceedings of the 37th Annual Design Automation Conference*, pages 458–463. ACM, 2000.

[106] Franc Brglez. Neutral netlist of ten combinational benchmark circuits and a target translator in FORTRAN. In *Special session on ATPG and fault simulation, Proc. IEEE Int. Symp. Circuits and Systems*, 1985.

[107] *ITC'99 Benchmarks* [Online]. Available: `http://www.cad.polito.it/downloads/tools/itc99.html`.