When Alice meets the "Hardware" Bob — Attacks and Solutions in the Digital World

by

Yadi Zhong

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
May 4, 2024

Keywords: Cryptanalysis, GIFT-COFB, Fault Attacks, Boolean Satisfiability (SAT), Test Pattern Generation, Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK)

Approved by

Ujjwal Guin, Chair, Associate Professor of Electrical and Computer Engineering
Vishwani D. Agrawal, Professor Emeritus of Electrical and Computer Engineering
Adit D. Singh, Professor of Electrical and Computer Engineering
Mehdi Sadi, Assistant Professor of Electrical and Computer Engineering
Akond Rahman, Assistant Professor of Computer Science and Software Engineering

Abstract

Modern cryptography and the digital world (CMOS mainly) have been advancing side-by-side. Although cryptographic algorithms only need to be proven with theoretical soundness, cryptographic engineering on digital device requires more efforts to achieve a secure implementation than simply just design theories. Although properties like energy-efficiency, low-latency, and minimal area overhead are desired in hardware-based implementations for cryptographic modules, they could potentially weaken the theoretical justification of cryptosystems, where adversaries can break the secret key inside the hardware.

In this dissertation, we develop novel attack to uniquely determine the secret key for hardware implementation of GIFT-COFB, one of the ten finalists for National Institute of Standards and Technology (NIST) Lightweight Cryptography Standardization Process. The 2-round partial unrolled design of GIFT-COFB is shown to be the most energy-efficient among all other r-round partial unrolling and fully unrolled settings. Our proposed chosen-plaintext attack can effectively break the master key K on this 2-round partial unrolled GIFT-COFB. In addition, we present two chosen-plaintext attacks on multicycle AES implementations with fault-based attacks. Both attacks on GIFT-COFB and multicycle AES include explanations from algebraic cryptanalysis perspective.

In parallel, the adoption of horizontal business models in semiconductor manufacturing is negatively affected by the overproduction of integrated circuits (ICs) and the piracy of intellectual properties (IPs), which compromised the integrity of the digital world's semiconductor supply chain. Logic locking emerges as a primary design-for-security measure to counter these threats, where ICs become fully functional only when unlocked with a secret key. However, Boolean satisfiability-based attacks have rendered most locking schemes ineffective. This gives rise to numerous defenses and new locking methods to achieve SAT resiliency. Subsequent attacks have been proposed to target these newly proposed solutions. The reasons behind the effectiveness of SAT attack and the following SAT-based attack have yet to be explored. In this

dissertation, we provide a unique perspective on SAT attack efficiency based on conjunctive normal form (CNF) stored in SAT solver. We demonstrate how this attack learns new relations between keys in every iteration using distinguishing input patterns and the corresponding oracle responses. Each input-output pair gives additional CNF clauses of unknown keys to be appended to SAT formulation, which leads to an exponential reduction in incorrect key values. Overall, SAT attack is shown to break most locking scheme within the linear iteration complexity of key size. Our analysis provides a new perspective on the capabilities of SAT attack against multiplier benchmark c6288 with possibly new directions to achieve SAT resiliency.

In the digital world, it is also crucial to reduce manufacturing defect escapes in today's safety-critical applications requires increased fault coverage. However, generating a test set using commercial automatic test pattern generation (ATPG) tools that lead to zero-defect escape is still an open problem. It is challenging to detect all stuck-at faults to reach 100% fault coverage. It remains challenging to detect hard-to-detect and redundant faults for large VLSI circuits. More optimization needs to be done as undetected faults still exist under the state-of-the-art commercial ATPG tools. Rather than attacking logic locking with SAT solvers, in this dissertation, we propose a novel test pattern generation approach constructively using the powerful SAT attack on logic locking. A stuck-at fault is modeled as a locked gate with a secret key, where it can effectively deduce the satisfiable assignment with reduced backtracks under key initialization of the SAT attack. The input pattern that determines the key is a test for the stuck-at fault. We propose two different approaches for test pattern generation. First, a single stuck-at fault is targeted, and a corresponding locked circuit with one key bit is created. This approach generated one test pattern per fault. We also consider a group of faults and convert the circuit to its locked version with multiple key bits. The inputs obtained from the SAT attack tool are the test set for detecting this group of faults. Our approach finds test patterns for all hard-to-detect faults that were previously undetected in commercial ATPG tools. The proposed test pattern generation approach can efficiently detect redundant faults with ITC'99 benchmarks. The results show that we can detect all the hard-to-detect faults and identify redundant faults, and a 100% stuck fault coverage is achieved.

Finally, we consider privacy-enhancing solutions to offer additional benefits for securing the digital world. In this dissertation, we develop an efficient, secure, and on-demand communication protocol using zero-knowledge proofs (ZKPs) that allow the prover to provide evidence of its secret without revealing that to the verifier. The edge device, acting as the prover, convinces the central server, the verifier, of the unique PUF response stored inside the device without requiring the actual storage of PUF responses on the server. The non-interactive characteristic of zk-SNARK, Zero-Knowledge Succinct Non-Interactive Argument of Knowledge, offers better optimization to authentication frequency, communication bandwidth between device and server, and protection of device-specific secret, all of which contribute to constructing our proposed device authentication framework.

Dedication

Dedicated to my dear parents.

Dear Lord Jesus Christ, the everlasting Light. “And so I come to the realization that I cannot fully trust in anyone to help me in my necessities save only You – my hope, my trust, my comfort, and my most faithful Friend ... Only You Yourself, my God, can stand by me, help me, comfort, counsel, teach, and defend me.”

— The Imitation of Christ by St. Thomas A Kempis

And singing the canticle of Moses, the servant of God, and the canticle of the Lamb, saying: Great and wonderful are Thy works, O Lord God Almighty; Just and true are Thy ways, O King of ages. Who shall not fear Thee, O Lord, and magnify Thy name? For Thou only art Holy: for all nations shall come, and shall adore in Thy sight, because Thy judgments are manifest.

— Revelation 15:4-5 (Douay-Rheims)

Galatians 6:7-9, Philippians 4:6-7, Ephesians 6:10-12.

I really appreciate the anonymous individuals who wrote the following Bible verses on the steps outside of Broun Hall, which I saw on April 22, 2024. These verses (Colossians 1:16, Philippians 1:6) mean very much to me and I am grateful that the Lord has planned all these.

ALL HAIL KING JESUS!

For in Him were all things created in heaven and on earth, visible and invisible, whether thrones, or dominations, or principalities, or powers. All things were created by Him and in Him. — Colossians 1:16

Being confident of this very thing: that He who hath begun a good work in you will perfect it unto the day of Christ Jesus. — Philippians 1:6

Acknowledgments

Praise be to the Almighty God for this PhD journey! You are the Way, the Truth, and the Life, and You are the source that keep me going! This PhD is made possible only because of You.

I would like to express my gratitude to my parents for continually uplifting me through my journey through graduate school at Auburn, without your support, I could not have made it this far. I really appreciate my dad for patiently waiting for me every day to finish my daily tasks; and my mom for teaching me a whole lot more with her own experience as a professor. It is great blessings from God that to learn Thy way of righteousness and daily strive toward Thy heavenly bliss! Words cannot express the goodness of the Lord and His plan in this journey as we grow in trusting Him in everything, and experiencing that God is in control of everything, both small and great. Christ Jesus is the ruler of all and Thy works are manifest! I am grateful for your constant warning, encouragement, and advice!

Thank you to my advisor Dr. Ujjwal Guin for providing me the opportunity both as an undergrad researcher (thanks to Alex Jones for turning down the opportunity in the first place) and a graduate student at Auburn. Thank you for heavy editing on my plethora of paper drafts. I appreciate the encouragement, and scolding as well (these all helped me to improve my research), for all rejections and major revisions and minor revisions, and I learned the importance of consistent revising to improve these papers.

I would like to extend my gratitude and appreciation to the committee members of my dissertation: Dr. Vishwani D. Agrawal, Dr. Adit D. Singh, and Dr. Mehdi Sadi. Thank you all for your interest, valuable suggestions, and time which makes this thesis possible. I learned a lot from your research and professional expertise. I am particularly grateful of having the opportunity to

take your classes in both undergraduate and graduate studies. These include Dr. Singh's Digital Logic Circuits (ELEC 2200), where I was first introduced to recycled ICs and SRAM power-up states, VLSI Design (ELEC 6770) and Advanced VLSI Design (ELEC 7770), and Dr. Sadi's Computer Aided-Design of Digital Integrated Circuits (ELEC 6250) and AI and Neuromorphic Hardware (ELEC 6970). In addition, Dr. Agrawal's excellent VLSI testing book has been an valuable resource to me. Thank you to my university reader Dr. Akond Rahman for detailed comments and feedback that helped improve the quality of this dissertation.

I would also like to thank my colleagues in the System Engineering and Security Lab, Ayush and Yuqiao, for assisting and teaching me a great amount in areas I was unfamiliar with when I first started my graduate school. I appreciate all the help I received from my colleagues, Pratiksha, Kaniz, Austin, Caleb, Josh, Colton, and Chris, during my time working towards a PhD. I am grateful for the insightful/funny discussions I had with all the undergrads in our lab, William, Holden, Craig, Mukarram, Pascal, Farrok, Gaines (now Master student in the same lab), Ammar, and Josh. You are great friends to me.

A big Thank You goes to my coauthors, Josh and Ammar. I enjoyed working with you all and I am grateful for all collaborations, brainstorming, discussions, and writings.

I would like to express my gratitude to all excellent faculty members in Department of Electrical and Computer Engineering. In both my undergraduate and graduate studies, I have taken many exciting and intensive courses, which were critical in strengthening my foundation in various diverse topics related to signal processing, electronics, embedded systems, computer networks, VLSI design, digital system designs, and hardware security. These academic courses helped me comprehend and describe the novelty in my research work. I appreciate all the technical support I received from Mr. Autry May, Mr. Michael Eddy, and Mr. John Tennant of ECE Department. My sincere appreciate to Mrs. Linda Allgood, Mrs. Mary Lloyd, Mrs. Elizabeth Gowan, and Mrs. Faith Fain in the ECE Department for their excellent administrative support, guidance, and help in both my undergraduate and graduate years. I am grateful for everything the department has provided during my studies.

Finally, my sincere thanks to all of my labmates for making the research experience more enjoyable and rewarding. Your help ranging from brainstorming new ideas to simply assisting with menial tasks when I am busy means very much to me. Y'all made my day.

Dear Lord, the ruler of all, the King of kings, the Lord of lords, and also the Advisor of advisors, no one can start this PhD journey or make it to completion save You. It is truly a joyful thing, always and everywhere, to give thanks to You and count the blessings You have bestowed upon me. Lord, I praise You for the numerous graces You have showered on your daughter on this journey and guided me through the rejections, storms, temptations of relying on others instead of fully trust in You, who knows both the visible and invisible things, past, present, and future.

Contents

List of Figures

List of Tables

List of Abbreviations

AES-NI Intel Advanced Encryption Standard New Insturctions

AES Advanced Encryption Standard

ASIC Application-Specific Integrated Circuit

ATPG Automatic Test Pattern Generation

CCA Chosen-Ciphertext Attack

CMOS Complementary Metal Oxide Semiconductor

CNF Conjunctive Normal Form

COFB Combined Output Feedback Mode

CPA Chosen Plaintext Attack

CRP Challenge-Response Pair

CRS Common Reference String

DFA Differential Fault Analysis

DFT Design-for-Testability

DIP Distinguishing Input Pattern

ECDHE Elliptic Curve Diffie-Hellman Key Exchange

EUF-CMA Existential Unforgeability under Chosen-Message Attack

FA Fault Attack

FPGA Field Programmable Gate Array

FSM Finite State Machine

HLS High-Level Sythesis

IC Integrated Circuit

IoT Internet of Things

IP Intellectual Property

LUT Look-Up Table

LWC Lightweight Cryptography

MAC Message Authentication Code

NIST National Institute of Standards and Technology

NP Non-Deterministic Polynomial Time Complexity

NVM Non-Volatile Memory

PET Privacy Enhancing Technology

PPT Probabilistic Polynomial Time

PRNG Pseudo Random Number Generator

PUF Physically Unclonable Function

QAP Quadratic Arithmetic Program

R1CS Rank-1 Constraint System

RE Reverse Engineering

sa0 Stuck-at 0

sa1 Stuck-at 1

SAT Boolean Satisfiability

SCA Side-Channel Attack

SoC System-on-a-Chip

SRAM Static Random Access Memory

TLS Transport Layer Security

zk-SNARK Zero-Knowledge Succinct Non-Interactive Argument of Knowledge

ZKP Zero-Knowledge Proof

Part I

Alice and the “Hardware” Bob

Chapter 1

Introduction

Cryptography is everywhere. Whether in checking the latest weather updates, sending emails, or downloading software packages, we are using cryptography, however implicit it may seem, every day. Dear reader, you are now reading this dissertation in pdf format obtained from Auburn University Electronic Theses and Dissertations (AUETD) database, `https://etd.auburn.edu/`. You have already made a secure connection with AUETD server before you can download the dissertation to your electronic device, *i.e.,* PC, laptop, tablet, or phone. How is this connection established between your device and the server, you may ask? Or how this dissertation is downloaded securely without malicious modifications by potential adversarial parties in-transit? The answer is in Transport Layer Security (TLS) 1.3. TLS 1.3 protects network communications with client initiating the handshake protocol with server using Elliptic Curve Diffie-Hellman key exchange (ECDHE) server to agree on shared secret and subsequently the session keys; then it proceeds with record layer for both parties, where authenticated encryption with associated data (AEAD) is performed based on shared secret keys for secure communication. As shown in Figure 1.1, the authenticated encryption scheme AES_256_GCM is applied to encrypt the following communications with AUETD server, which is how the dissertation is delivered to your digital device.

In the abstract view, one may consider the main business of cryptography in TLS 1.3 as securing messages transmitted between two parties, A and B, in the midst of passive and active adversaries. Parties A and B, on the other hand, are generally referred to as Alice and Bob, since Ron Rivest coined them in the seminal work of RSA in 1978 [1]. With a plethora of attackers trying to break cryptographic schemes which Alice and Bob are using in order

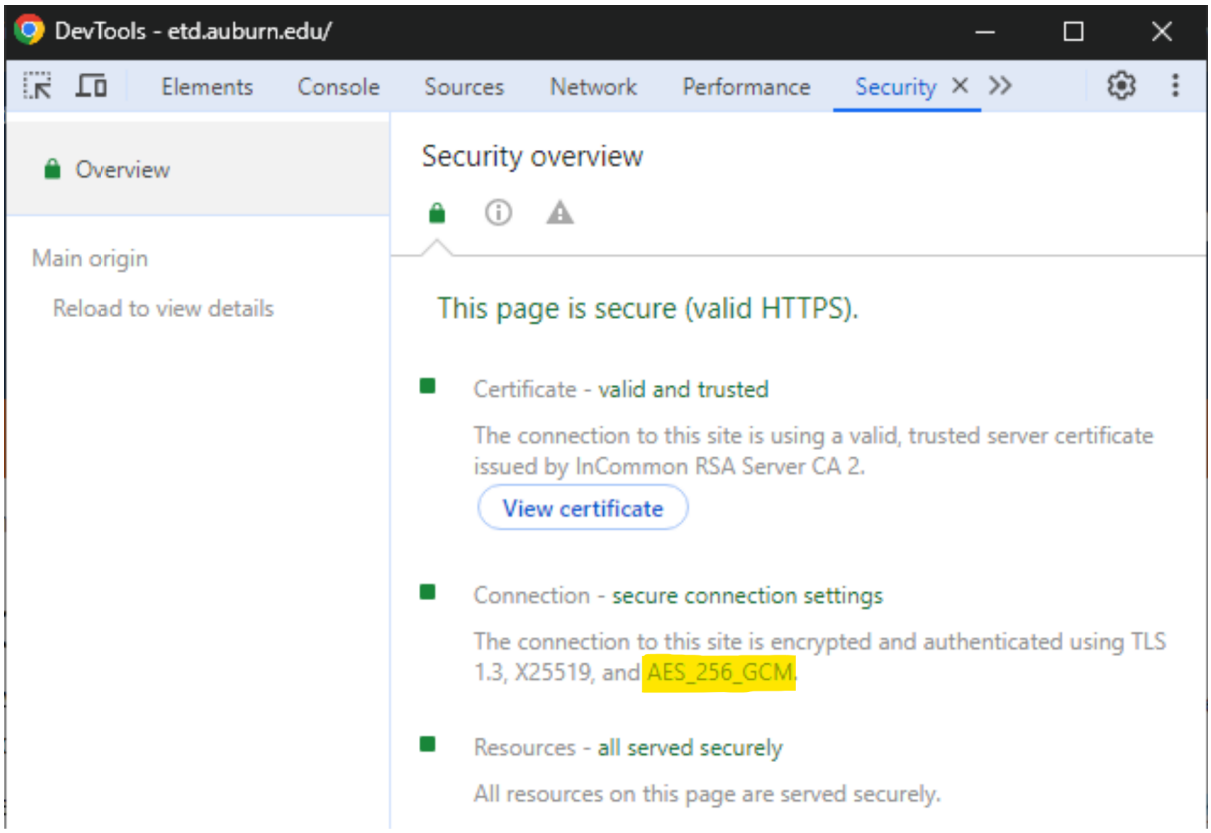

Figure 1.1: TLS 1.3 and authenticated encryption scheme AES_256_GCM are used to secure communications with AUETD server, `https://etd.auburn.edu/`. Screenshot was taken with Google Chrome.

to extract their secret messages, these adversaries are given various names according to their roles, *i.e.,* Eve the eavesdropper, Mallory the malicious attacker, Oscar the opponent, Trudy the intruder, etc. They can launch network-based attacks such as message injection attack, message dropping attack, re-ordering attack, reflection attack, replay attack, etc. In theory, on the other hand, they are categorized by ciphertext-only attacks, known-plaintext attacks, chosen-plaintext attacks (CPA), chosen-ciphertext attacks (CCA), or existential unforgeability under chosen-message attacks (EUF-CMA) for digital signatures. Correspondingly, security notions for passive and active attacks have been developed by cryptographic community: EAV–security, CPA–security, CCA–security, and EUF-CMA–security, etc.

Digital world, in parallel with the timing for development of modern cryptography, has been advancing with complementary metal oxide semiconductor (CMOS) technology and the ever-shrinking technology nodes based on Moore's law. The transformation in the digital realm,

dear reader, brings the technical innovations in PCs, laptops, tablets, phones, etc., which you are now using it to read this dissertation. In addition, the incentive to achieve power efficiency and reducing energy consumption drives the development and designs of microcontrollers, embedded device, Internet of Things (IoTs), low-power application-specific integrated circuits (ASICs) and field programmable gate arrays (FPGAs).

Considering from cryptographic engineering perspective, the digital world has certainly facilitated algorithmic development of efficient implementations for both power reduction, performance increase, and area/footprint minimization on diverse platform from servers to PCs, embedded systems, ASICs, and FPGAs. Intel's Advanced Encryption Standard (AES) New Instructions (AES-NI)* is one notable example which runs round-based AES in dedicated hardware inside the CPU. It achieves up to $10\times$ performance increase than the software counterparts. Processor families since Intel Westmere and AMD Bulldozer of 2010 support AES-NI for cryptographic hardware accelerations. This exemplifies the fact that optimizations in hardware implementations offer better performance in cryptographic engineering than software implementations.

Nevertheless, digital world also brings new sets of attacks to cryptosystem designs as never before. In addition to passively or actively attacking through communication channels, ubiquitous hardware provides novel attack venues to adversaries. Side-channel attacks (SCAs) and fault attacks (FAs) give attackers the additional capabilities to recover the secret inside hardware device, *e.g.,* encryption/decryption keys for cryptographic algorithms. These include power-based SCA, electromagnetic-based SCA, timing-based SCA, Flush+Reload attacks, cache template attacks, speculative execution attacks, load value injection attacks, clock glitching attacks, voltage glitching attacks, undervolting attacks, laser injection attacks, to just name a few. Attacks like above exploits the hardware the adversaries are currently in possession of, *i.e.,* ASICs, FPGAs, microcontrollers, and DRAMs, or even launch these attack remotely for some SCAs (hardware is required still). One of the earliest record of SCA attacks can be dated back to World War II, where scientists from Bell Labs decoded Signal Corps teletypewriter's encrypted message with 75% accuracy [2]. This demonstrates the successful

*Breakthrough AES Performance with Intel® AES New Instructions, Intel Corporation, 2010.

side-channel based attack on hardware, *i.e.,* the typewriter, where key press happened about 80 feet away from the observation site.

1.1 Thesis Statement and Research Questions

In the dissertation, we follow two main themes: (i) hardware-based attacks, and (ii) hardware-based solutions. This section explains the intuitions behind our title "When Alice meets the 'Hardware' Bob — Attacks and Solutions in the Digital World" and Part I of this dissertation, including the roles which Alice and Bob played in each theme.

When focusing on hardware-based attacks, Alice, our main character, will become the adversary in this dissertation, who is launching attacks to break the secret key and derive its secret value with a series of challenge-response pairs via interactions with the "Hardware" Bob. This challenge-response pair can be thought of as plaintext-ciphertext pairs from a cryptographic module, or simply input-output response of a hardware locked circuit. Note that in Kerchoff's principle, information about cryptographic designs or systems are public save the secret key. In hardware security domain, the locked circuit generally can be obtained by some means or from specific entities (may or may not be malicious), while the key which locks the design is kept secure, *i.e.,* tamper-proof memory. The goal for Alice the adversary, on the other hand, is to break these keys and successfully determine their values with the help of some challenge-response pairs, known or chosen by purpose.

We consider hardware-based solutions in the last part of this dissertation. Our character Alice, then, returns to its former benign profession. In VLSI testing, Alice is mainly concerned with increasing stuck-at faults coverage for Bob the hardware (hardware circuits) in automatic test pattern generation (ATPG). More specifically, two paths are considered: (i) test pattern generation for hard-to-detect faults where commercial ATPG tools failed to find suitable patterns, and (ii) identification of redundant faults where no test pattern will be needed. Another solution we explore in this dissertation is device authentication performed in a large digital-ecosystem, such as IoT nodes for an in-field applications. The "hardware" Bob represents the IoT edge node that is required to prove and demonstrate to Alice, acting as server

and the verifier, of its authenticity. In this scenario, Alice is applying privacy-enhancing cryptography to build achieve authentications of edge device for central server, while taking into consideration the malicious parties' potential disruptions of authentication process or attempts to stole device-specific proprietary information. Her overarching goals are to find solutions to hardware-related problems in VLSI testing and hardware security.

Therefore, we expand the scope of Alice and Bob, along with their interactions, though traditional defined in cryptography alone, into hardware security and VLSI testing realms as well. The digital world, on the other hand, relies heavily on hardware as its foundation, along with the ever-evolving development and advancement in chip design, manufacturing, testing, assembly and packaging process. For the above reasons, the dissertation aims to look at research problems from the hardware perspective. In particular, (i) how hardware can be exploited or facilitate additional attack vectors to break the internal secret? (ii) How would hardware be a core benefit to help form new solutions for hardware assurance and quality assurance?

1.2 Contributions

This dissertation proposes novel attacks and/or solutions in cryptography, hardware security, VLSI testing as mentioned above. The contributions of this dissertation are summarized as follows:

In Part I (Chapters 1, 2), we begin this dissertation with introductory discussions on Alice and the "Hardware" Bob, the big picture of attacks and solutions in the digital world, as well as preliminaries. Background and necessary preliminaries in cryptography's substitution-permutation network, authenticated encryption with associated data, semiconductor supply chain, zero-knowledge proof (ZKP) and its properties are introduced in Chapter 2.

In Part II (Chapters 3, 4, 5, 6), we discuss multiple hardware-based attacks in broader scope of cryptography and hardware security. It ranges from algebraic cryptanalysis on hardware-based GIFT-COFB and multicycle AES; the analysis of Boolean satisfiability (SAT) attack, its iteration complexity on various logic locking schemes; to fault-based attack on logic locking from ATPG perspective to retrieve its internally programmed secret key. These chapters include the following publications:

Hardware-based Attacks

• **Y. Zhong**, and U. Guin, "Chosen-Plaintext Attack on Energy-Efficient Hardware Implementation of GIFT–COFB," in IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 1-4, 2022.

• **Y. Zhong**, and U. Guin, "Fault-Injection Based Chosen-Plaintext Attacks on Multicycle AES Implementations," in ACM Great Lakes Symposium on VLSI (GLSVLSI), pp. 1-6, 2022.

• **Y. Zhong**, and U. Guin, "Complexity Analysis of the SAT Attack on Logic Locking," in IEEE Transactions on Computer-Aided Design of Integrated Circuits, pp. 1-14, 2023.

• **Y. Zhong**, A. Jain, M.T. Rahman, N. Adadi, J. Xie, and U. Guin, "AFIA: ATPG-Guided Fault Injection Attack on Secure Logic Locking," Journal of Electronic Testing: Theory and Applications, pp. 1-20, 2022.

Chapter 3 develops the chosen-plaintext attack (CPA) on the most-energy-efficient hardware implementation of GIFT-COFB, one of the ten finalists in the NIST Lightweight Cryptography Standardization Process [3]. Only a constant number of nonce-tag pairs is required to break the 128-bit key with $\boldsymbol{O}(2^4)$ attack complexity [4]. Chapter 4 presents the fault injection attacks on Advanced Encryption Standard (AES), a widely adopted symmetric cipher standardized by NIST in 2000. By injecting faults on a single-bit register, the overall attack complexity to break a 128-bit key is $\boldsymbol{O}(2^8)$. This complexity can be further extended to 192-bit and 256-bit keys as well [5]. Chapter 5 describes the full anatomy of the SAT attack, one the most effective break against logic locking. Linear iteration complexity is observed for the traditional locking on ISCAS'85 benchmarks. Analysis and explanations are given to provide additional insights into the SAT-based attacks on post-SAT solutions [6]. Chapter 6 introduces AFIA: ATPG-guided fault injection attack on secure logic locking [7]. The average complexity to break the secret key of a locked circuit is linear to the key length. A quadratic complexity is only needed in the worst-case scenario to completely determine the key bits.

In Part III (Chapters 7, 8, 9), we target hardware-based solutions in VLSI testing and device authentication. In addition, future work with possible solutions and mitigation in hardware security and VLSI testing are discussed. Papers included in this theme are listed as follows:

Hardware-based Solutions

- **Y. Zhong**, and U. Guin, "A Comprehensive Test Pattern Generation Approach Exploiting SAT Attack for Logic Locking," in IEEE Transactions on Computers, pp. 1-13, 2023.
- **Y. Zhong**, J. Hovanes, and U. Guin, "On-Demand Device Authentication using Zero-Knowledge Proofs for Smart Systems," in Great Lakes Symposium on VLSI (GLSVLSI), pp. 569-574, 2023.
- J. Hovanes, **Y. Zhong**, and U. Guin, "A Novel IoT Device Authentication Scheme Using Zero-Knowledge Proofs," in GOMACTech, pp. 1-4, 2023.
- **Y. Zhong**, A. Ebrahim, U. Guin, and V. Menon, "A Modular Blockchain Framework for Enabling Supply Chain Provenance," in IEEE Physical Assurance and Inspection of Electronics (PAINE), pp. 1-7, 2023.

Chapter 7 proposes novel test pattern generation approaches using the efficient SAT attack on logic locking. We exploit the linear iteration complexity in the SAT attack for generating tests for undetected faults, of which the commercial ATPG tools, *e.g.,* TetraMAX II, failed to find suitable test patterns or determine as reduced faults. The equivalence of a stuck-at fault (*sa0*, *sa1*) is converted to its equivalent key gates. Our proposed approaches achieve 100% fault coverage by identifying redundant faults and producing test patterns for undetected faults [8]. Chapter 8 provides a novel device authentication protocol to for IoT devices [9,10]. The widely available SRAM PUF on each edge device is its device fingerprint. This proposed zk-SNARK-based approach protects secrecy of the PUF response even if the central server is compromised by adversaries. It is also resilient against machine learning-based PUF attacks since it is impossible for an adversary to observe any challenge-response pair in a meaningful way. Chapter 9 concludes the dissertation and discusses possible future directions in cryptography, hardware security, and VLSI testing.

A list of all 9 co-authored peer-reviewed publications along with links to source code are included in the Appendix.

1.3 Organization of this Thesis

The rest of this dissertation is organized as follows and shown in Figure 1.2.

Part II: Hardware-based Attacks

In Chapter 3, we propose a chosen-plaintext attack on the most-energy-efficient hardware implementation of GIFT-COFB by Caforio et al. As any attacker could have access to the IoT device, he/she can apply a nonce to the GIFT-COFB encryption oracle and observe the 2-round update from the 128-bit output tag. Multiple nonce-tag pairs can be captured through resetting the oracle and assigning new values to the input nonce. With only a few nonce-tag pairs, the adversary can recover both round keys of round 1 and 2, and use them to derive the 128-bit secret key, K, through the reverse of key schedule. The attack was published at HOST 2022 (Washington DC) and has been cited in NIST IR 8454*:

- **Y. Zhong**, and U. Guin, "Chosen-Plaintext Attack on Energy-Efficient Hardware Implementation of GIFT–COFB," in IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 1-4, 2022.

In Chapter 4, we propose two attacks to break the multicycle AES implementations. Both attacks take three plaintext-ciphertext pairs to evaluate one key byte. The first attack is designed to break vulnerable AES implementations which leaks round operation to the output. An adversary can also access the internal scan chains and observe the round register value. By varying one byte in plaintext, we showed that the effect of three other key bytes of the same MixColumns operation, along with key addition, can be eliminated by XORing the same output byte using only two plaintext-ciphertext pairs. However, one more plaintext-ciphertext pair is necessary to remove the redundant solution to uniquely determine the correct key byte. The second attack focuses on breaking a multicycle AES implementation where the contents in

*NIST IR 8454: Status Report on the Final Round of the NIST Lightweight Cryptography Standardization Process, June 2023.

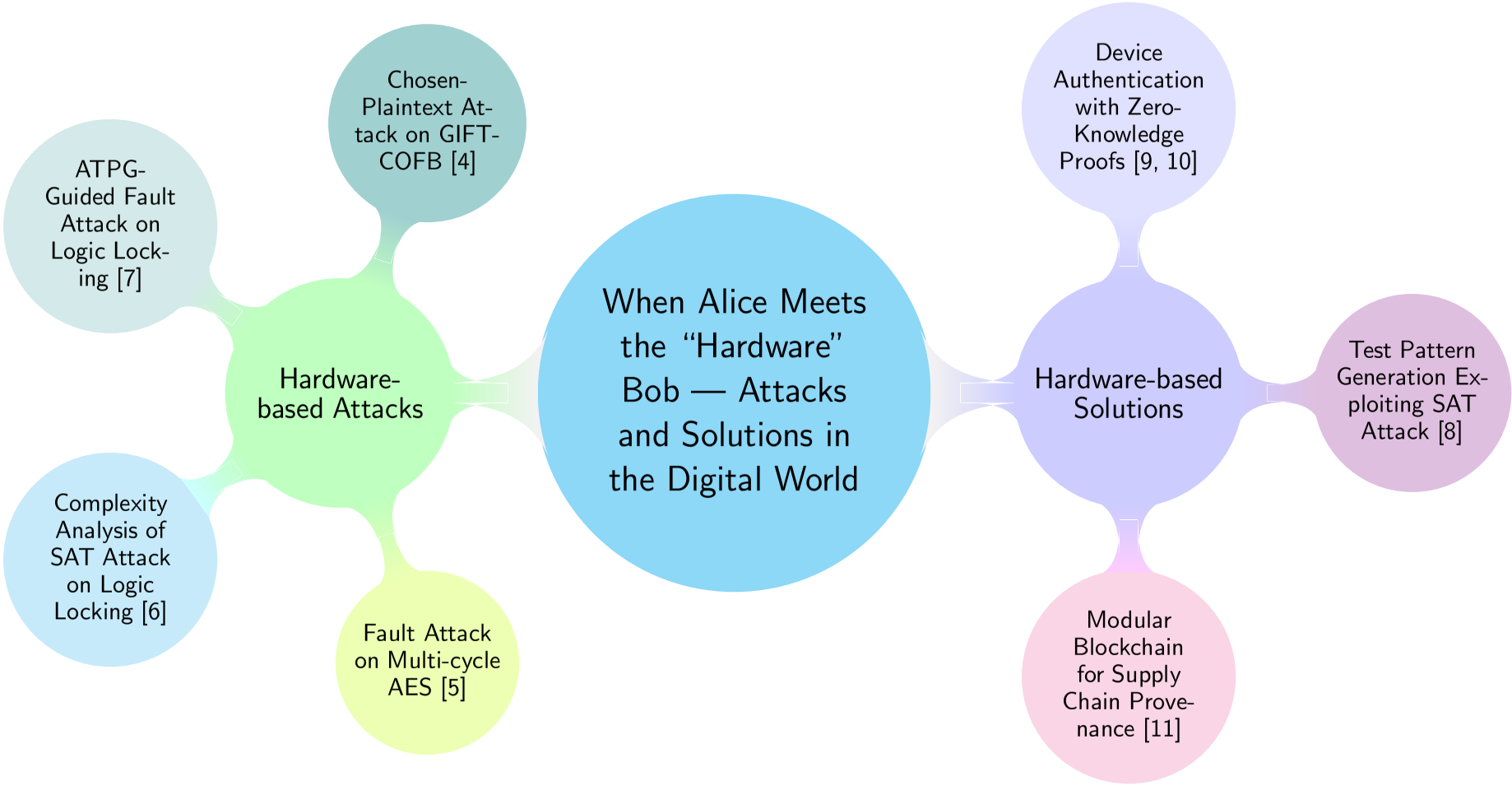

Figure 1.2: Overview of publications included in this dissertation.

ciphertext registers are updated once all the round computations are complete. The adversary can launch the attack by injecting a fault in the flag register which signals the completion of all computations (e.g., done). Once a fault is injected, the internal states of the round registers are dumped as the ciphertext. An adversary can perform the same steps mentioned in the first attack to determine the key after observing the first round result from the ciphertext. This attack is also applicable to the encryption round that skips MixColumns. As no faults are injected in the internal computations of AES, the traditional fault detection schemes can not identify this attack. We demonstrated both attacks on different AES implementations with 128-bit key size (AES-128) from the OpenCores benchmarks with the key search space complexity of $\boldsymbol{O}(2^8)$. The same attacks can be applied to other AES implementations with 192/256-bit keys. Our attack was published at GLSVLSI 2022 (Irvine, CA):

- **Y. Zhong**, and U. Guin, "Fault-Injection Based Chosen-Plaintext Attacks on Multicycle AES Implementations," in ACM Great Lakes Symposium on VLSI (GLSVLSI), pp. 1-6, 2022.

In Chapter 5, we present two novel aspects to analyze the iteration complexity of the oracle-guided SAT attack. First, we provide a detailed analysis of the SAT attack based on the conjunctive normal form (CNF) clauses stored in the SAT solver. The SAT attack iteratively finds DIPs to eliminate an equivalent class of incorrect keys. We explore what the attack learned after finding a DIP at each iteration. We show that the SAT tool is implicitly constructing a relationship between different key bits by applying the DIP to the oracle and observing the correct response. Please note that the desired goal for any logic locking technique is to achieve an exponential iteration complexity of key size so that an adversary cannot determine the correct key value within given time constraints. However, our analysis points to the linear growth of the required patterns or iterations rather than the desired exponential increase with keys. Using examples, we showed how the attack uses a DIP to eliminate a class of equivalent keys which results in the linear iteration complexity. We also confirmed that the complexity gets even lower for circuits with multiple overlapping logic cones. Note that a logic cone can be described as a directed graph where the input nodes and gates point toward the sole output.

Second, one interesting observation is that the complexity (*i.e.,* number of iterations/DIPs) of the SAT attack often reduces with increasing key size. We provided detailed explanations of why it takes fewer iterations to find the correct key when we locked a circuit with a larger key size. Finally, we analyzed the SAT attack complexity for a circuit locked with point functions. Our complexity analysis was published at IEEE Transactions on Computer-Aided Design of Integrated Circuits, 2023:

- **Y. Zhong**, and U. Guin, "Complexity Analysis of the SAT Attack on Logic Locking," in IEEE Transactions on Computer-Aided Design of Integrated Circuits, pp. 1-14, 2023.

In Chapter 6, we details how an adversary can extract the secret key from a locked netlist, even if all the existing countermeasures are in place. An adversary can determine the secret key by injecting faults at the key registers, which hold the key value during normal operation, and performing differential fault analysis. We present a key sensitization-based ATPG-guided fault injection attack (AFIA), to break any locking scheme. The entire process can be performed in three steps. First, we processed the locked netlist and converted it into a directed graph to extract all logic cones and construct a key-cone association matrix that records the distribution of keys among different cones. This structural analysis facilitates total fault reduction for the subsequent test pattern generation. Second, it is necessary to select an input pattern that produces an incorrect response for the target key bit only while keeping its dependent keys at faulty states. This can be achieved by using a constrained automatic test pattern generation (ATPG) to generate such a test pattern, which is widely popular for testing VLSI circuits. A 1-bit key value can be determined by generating a test pattern that can detect the stuck-at fault (*saf*) at the target key (corresponds to that key bit) while keeping its dependent keys at logic 1 (or logic 0). This is due to the fact that dependencies are often inserted to prevent direct sensitization of a key bit to the output by test patterns due to other key lines blocking its path. In our proposed approach, the pattern which detects a stuck-at 1 (*sa1*) fault at one key line with logical constraints for the recovered key lines is sufficient to determine that key bit. One can also use stuck-at 0 (*sa0*) fault to derive such pattern. Note that the fault-free and faulty responses are always the complements under the test pattern which detects that fault, which in turn can to derive the

key-bit value. The same process needs to be applied for other key bits to generate similar input patterns, and this results in at most $\mathcal{K}$ patterns for determining the entire key of size $\mathcal{K}$. Note that one test pattern can detect multiple key bits when they are placed in different logic cones (of no dependencies). Third, we applied these test patterns to only one instance of unlocked chip and collect the responses. Faults can be injected at the blocking key registers using laser fault injection equipment and obtain the key value by comparing the output responses with test patterns' generated by constrained ATPG. Our fault attack on logic locking was published at Journal of Electronic Testing: Theory and Applications, 2022:

- **Y. Zhong**, A. Jain, M.T. Rahman, N. Adadi, J. Xie, and U. Guin, "AFIA: ATPG-Guided Fault Injection Attack on Secure Logic Locking," Journal of Electronic Testing: Theory and Applications, pp. 1-20, 2022.

Part III: Hardware-based Solutions

In Chapter 7, we demonstrate the novel miter construction for test pattern generation of stuck-at-0 (*sa0*) and stuck-at-1 (*sa1*) faults, where each fault has its equivalent locked circuit to be applied with the existing SAT-based attack. We target undetected faults where commercial ATPG tools TetraMAX II fall short in producing test patterns. Our work focuses on identifying the redundant ones from these undetected faults and finding suitable patterns for detecting non-redundant faults to increase fault coverage further. The equivalence of a stuck-at fault (*sa0*, *sa1*) is an AND or OR key-gate, respectively. Once the stuck fault is converted to a key-dependent AND/OR gate, SAT attack is applied to solve the key and return the distinguishing input patterns it used in deriving the key value. A distinguishing input pattern returned by the SAT attack allows us to sensitize a stuck fault and propagate the faulty response to the output. *To the best of our knowledge, this research was the first attempt to apply the SAT-based logic locking attack on the equivalent keyed circuit to (i) find test patterns for undetected faults, (ii) identify redundant faults, and (iii) reduce test pattern count for the combination of multiple faults.* Our SAT attack-based test pattern generation was published at IEEE Transactions on Computers, 2023:

- **Y. Zhong**, and U. Guin, "A Comprehensive Test Pattern Generation Approach Exploiting SAT Attack for Logic Locking," in IEEE Transactions on Computers, pp. 1-13, 2023.

Chapter 8 proposes an efficient, secure, and on-demand device authentication protocol using Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK). The signature from a physically unclonable function (PUF) is used to create the proof, which can be verified by the central server acting as a verifier without storing the actual PUF response. The edge device periodically generates proofs using a self-generated random seed. The proof can then be made public for verification for anyone with the common reference string (CRS). This allows repeated authentication by verifying the identity of an edge device without access to the PUF secret. This work was done in collaboration with my colleague Joshua Hovanes. Our zk-SNARK-based solutions were published at GLSVLSI 2023 (Knoxville, TN) and GOMACTech 2023 (San Diego, CA):

- **Y. Zhong**, J. Hovanes, and U. Guin, "On-Demand Device Authentication using Zero-Knowledge Proofs for Smart Systems," in Great Lakes Symposium on VLSI (GLSVLSI), pp. 569-574, 2023.

- J. Hovanes, **Y. Zhong**, and U. Guin, "A Novel IoT Device Authentication Scheme Using Zero-Knowledge Proofs," in GOMACTech, pp. 1-4, 2023.

In Chapter 9, we conclude the dissertation and discuss possible future directions. This include possible solutions to secure semiconductor supply chain with modular blockchain-based approaches [11], future work in heterogeneous integration, 3D packaging, as well as possible hardware-based applications with privacy-enhancing technologies (PETs).

Chapter 2

Background and Preliminaries

2.1 Substitution-Permutation Network, Authenticated Encryption, Authenticated Encryption with Associated Data

Modern cryptography and digital world (CMOS mainly) have grown side-by-side. Although cryptographic designs only need to be proven sound theoretically, cryptographic engineering on digital device requires more efforts to achieve a secure implementation than just theoretical soundness.

Since NIST's standardization of Advanced Encryption Standard (AES), block ciphers subsequently designed are largely based on substitution-permutation network (SPN) or its tweakable version. AES has 10, 12, or 14 encryption rounds for 128-bit, 192-bit, or 256-bit key, respectively. Each encryption round includes SubBytes, ShiftRows, MixColumns, and AddRoundKey. All are linear operations except SubBytes, where sixteen 8-bit Sboxes have non-linear property between its input and output byte. Similarly, GIFT–128 is a 128-bit SPN block cipher with 40 encryption rounds [12]. Each round contains, SubCells, PermBits, AddRoundKey. GIFT Sbox (GS) is a non-linear function of 4-bit input/output with algebraic degree of 3.

Authenticated Encryption (AE) achieves secrecy and integrity simultaneously in the presence of an active adversary. It consists of symmetric encryption and Message Authentication Code (MAC), where generally encrypt-then-MAC is adopted. In terms of secrecy, AE need to satisfy CCA-Security. For authentication, unforgeability must be met, where any tampered

encrypted messages by attackers will be discovered. Authenticated Encryption with Associated Data (AEAD), on the other hand, also includes associated data as the input, which is authenticated but not encrypted, *e.g.,* packet header. AEAD has nonce N, message M, and associated data A as inputs; ciphertext C of the same length as the original message, and message authentication tag T (fixed length) as outputs. NIST initiated the process to standardize lightweight cryptography (LWC) back in 2015 requiring submissions in AEAD with optional hashing functionalities. Emphasis is placed on schemes suitable for constrained environment like RFID tags, industrial controllers, sensor nodes and smart cards [13]. GIFT-128 with combined output feedback mode (COFB) was selected as one of the ten finalists in the third round of NIST LWC competition.

2.2 Globalization of Semiconductor Supply Chain

The integrated circuits (ICs) are fundamental to virtually every technology in the Department of Defense (DoD), industrial and commercial spaces. Moore's Law has guided the microelectronics industry for decades to enhance the performance of ICs. The continuous addition of new functionalities in SoCs has forced design houses to adopt newer technology nodes ever-decreasing transistor size to continuously improve operational speed, reduce power consumption, overall die area, and the resultant cost of a chip. The exponential growth of electronics becomes feasible due to the globalization of semiconductor design, manufacturing, and test processes, and the markets and nation-state incentives which support the much needed investments production capability and capacity. Building and maintaining a fabrication unit (foundry) requires a multi-billion dollar investment [14]. As a result, a system-on-a-chip (SoC) design house acquires intellectual properties (IPs) from many vendors and sends the design to a foundry for manufacturing, typically located offshore due to the horizontal integration in the semiconductor industry. At present, the majority of the SoC design houses no longer design the complete SoC and/or manufacture chips on their own. As a result, the trusted foundry model is no longer assumed to be valid for producing ICs, where the trustworthiness of microelectronic parts is often questioned.

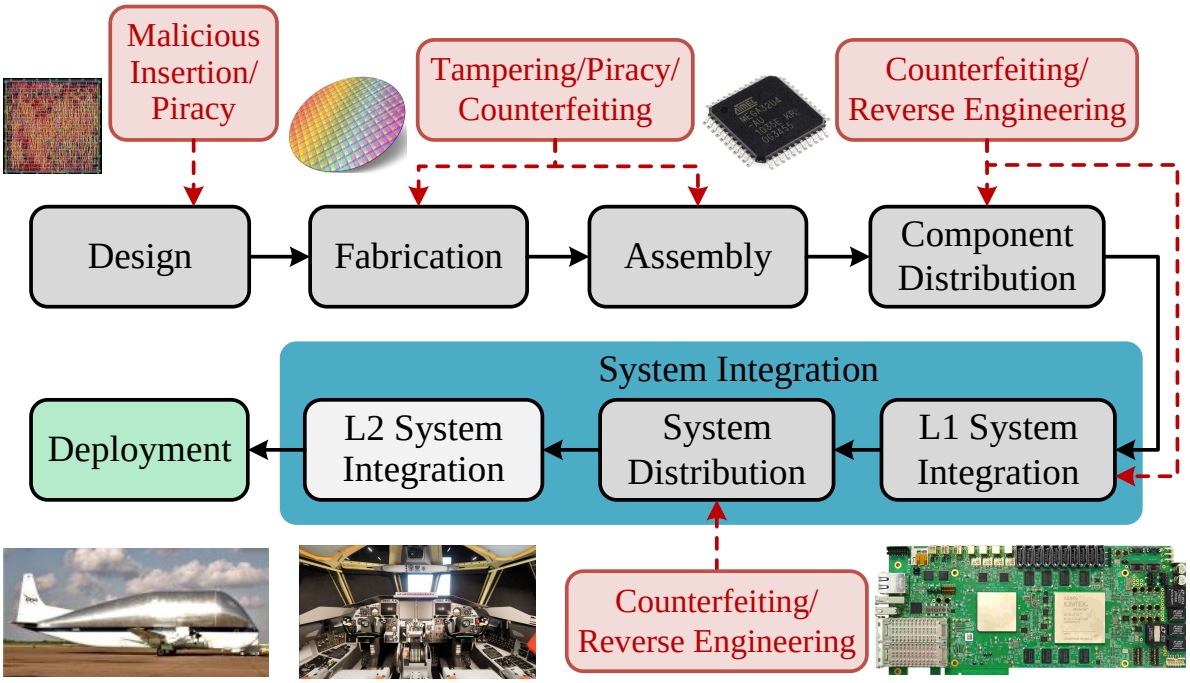

Figure 2.1: Overview of electronics supply chain and possible attack surface.

With horizontal business model in semiconductor supply chain, ensuring the security, integrity, and authenticity of electronic components, systems, and the supply chain that delivers them becomes highly challenging. Unfortunately, the same globalization opens Pandora's box of threats, including (i) counterfeit ICs [15–19], (ii) piracy of intellectual properties (IPs) and cloning [20–23], and (iii) malicious modifications or tampering with hardware Trojans [24–26], as shown in Figure 2.1. These threats could present in multiple stages in the electronics supply chain, including design, fabrication, assembly, distribution, and system integration, etc. Due to the sophistication of today's critical infrastructures, electronic products could be manufactured from multiple levels of system integration. For example, smaller systems, such as FPGAs and microcontroller boards, can be assembled first using discrete components, and we call it level-1 (L1) system integration. Complex systems, such as helicopter electronics, require the integration of multiple smaller L1 systems, and we denote it as level-2 (L2) system integration. Due to the complex globalized supply chain, an adversary could control or disrupt the supply chain or launch cyber attacks by exploiting hardware vulnerabilities. The hardware hack reported by Bloomberg shows a tiny chip, the size of a grain of rice, can be covertly hidden in a larger

system to infiltrate data in U.S. companies [27, 28]. The threat was undetected due to the difficulty of observing and understanding the complete functionality of the electronic system and its supply chain. Observing and understanding the electronics and supply chain operations has many challenges and, if improperly conducted, can expose further vulnerabilities and threats to intellectual property and trade secrets.

2.3 Zero-Knowledge Proof

First developed in 1980s by Goldwasser, Micali, and Rackoff [29], a zero-knowledge proof (ZKP) allows the prover (P) to convince the verifier (V) that a statement is true, yet without revealing any details about its secret witness. The verifier learns no additional information beyond the validity of the statement. In addition, Goldreich et al. [30] showed that ZKP proof systems exist for all non-deterministic polynomial (NP) languages.

The definition for ZKP is described as follows:

Definition

- Let L be a NP-language, with witness relation $\mathcal{R}$, s.t. $L = \{x | \exists w \ : \ \mathcal{R}(x, w) = 1\}$
- Let P be the prover and V be the verifier.
- P needs to convince V that $x \in L$ and also that P knows w.

Details of how P proves a statement or how V convinces of the validity of the statement itself are associated with the three properties of ZKP [29] – completeness, soundness, and zero-knowledge. The definition below is based on Groth'16 [31] and Bünz et al. [32]:

Properties of ZKP

- Completeness: For every $x \in L$,

$$\mathsf{Pr}\big[(\sigma, \tau) \leftarrow \mathsf{Setup}(R); \pi \leftarrow \mathsf{Prove}(R, \sigma, x, w) : \mathsf{Vrf}(R, \sigma, x, \pi) = 1\big] = 1 - \epsilon.$$

Note that ϵ is negligible, and $\epsilon = 0$ for perfect completeness for honest provers. (i.e., honest prover P convinces the verifier V of the secret w.)

- Soundness: For all probabilistic polynomial time (PPT) adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c}(R,z)\leftarrow\mathcal{R}(1^\lambda);(\sigma,\tau)\leftarrow\mathsf{Setup}(R);(x,\pi)\leftarrow\mathcal{A}(R,z,\sigma):\\ x\notin L_R \text{ and } \mathsf{Vrf}(R,\sigma,x,\pi)=1\end{array}\right]\rightarrow 0,$$

where λ is the security parameter. (i.e., cheating prover or adversary $\mathcal{A}$ fakes proofs with negligible probability.)

- Zero-Knowledge: For every PPT verifier $\mathcal{V}$, there exist a PPT simulator $\mathcal{S}$ with $\forall x \in L$, probability ensembles of $\left\{\mathsf{View}_{\mathcal{V}}\left(<P,\mathcal{V}>(x)\right)\right\}$ and $\left\{\mathcal{S}(\mathcal{V},x)\right\}$ are computationally indistinguishable. Equivalently, For all PPT adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{l}(\sigma,\tau)\leftarrow\mathsf{Setup}(R);\\ \pi\leftarrow\mathsf{Prove}(R,\sigma,x,w)\end{array}:\mathcal{A}(R,z,\sigma,\tau,\pi)=1\right]\approx$$
$$\Pr\left[\begin{array}{l}(\sigma,\tau)\leftarrow\mathsf{Setup}(R);\\ \pi\leftarrow\mathcal{S}(R,\tau,x)\end{array}:\mathcal{A}(R,z,\sigma,\tau,\pi)=1\right].$$

In other words, the proof reveals nothing to the verifier V except $x \in L$. In particularly, proofs should not reveal prover P's secret witness w. (i.e., verifier learns nothing except the validity of proofs.)

Part II

Hardware-Based Attacks

Chapter 3

Chosen-Plaintext Attack on Energy-Efficient Hardware Implementation of GIFT-COFB

3.1 Introduction

The recent advancement of the Internet of Things (IoT) results in more connected electronic devices than ever. Vast chunks of data are being transferred over the unsecured channel for increasingly ubiquitous computing. This may give rise to the potential breach of confidentiality, integrity, and authentication if, somehow, the devices transmit information without the proper protection mechanism. As IoT devices are resource-constrained and low-cost, have limited area, and less computation power, the previously standardized Advanced Encryption Standard (AES) block cipher is not suited for these devices. Thus, NIST instantiated the process to standardize lightweight cryptographic algorithm, stressing its importance on RFID tags, sensor nodes, industrial controllers, and smart cards [33]. Among all the candidates submitted to NIST, ten were selected as the finalists [34]. All finalists ensured tight security bounds and ensured efficient implementation in both hardware and software. One of the ten finalists is GIFT-COFB [35], which integrates combined-feedback mode (COFB) with GIFT-128 cipher [12] to offer authenticated encryption with associated data.

GIFT block cipher, as shown in Figure 3.1, belongs to the Substitution-Permutation Networks (SPNs), which utilizes 4-bit Sbox and bit permutation PermBits as the underlying SPN. Compared to the lightweight block cipher PRESENT [36], GIFT provides better efficiency in both area and performance as well as mitigates the vulnerability to the linear hull attack [37] against PRESENT. Although there have been a few attacks against GIFT cipher, such as cache attack [38], side-channel attack [39], and fault-injection attack [40], GIFT-COFB has shown

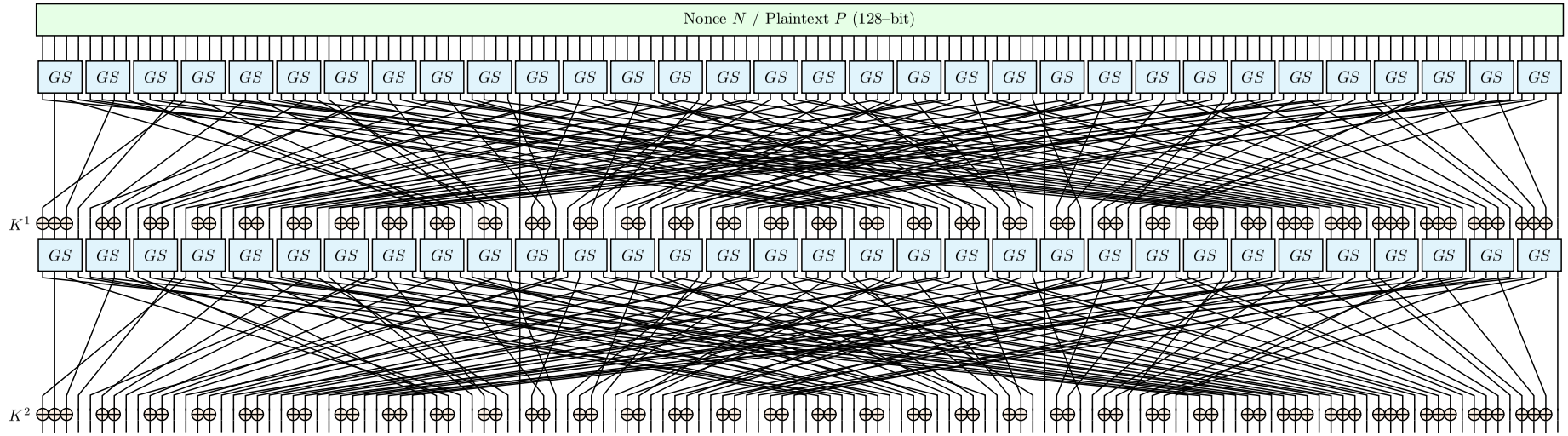

Figure 3.1: GIFT-128 block cipher.

to be secure against large encryption queries [41–43]. Various forgery attacks described in [41–43] requires the attacker to perform either $\boldsymbol{O}(2^{64})$ encryption or decryption blocks to break GIFT-COFB's authentication. As lightweight cryptography is implemented on IoT devices, it is equally important to examine the security bound for the same authenticated encryption in its hardware implementation.

The rest of the chapter is organized as follows. We briefly introduce the background for hardware implementations of GIFT-COFB and describe the threat model in Section 3.2. Our proposed attack is presented in Section 3.3. The result for the proposed attack is described in Section 3.4. Finally, we summarize this chapter in Section 3.5.

3.2 NIST Lightweight Cryptography Finalist GOFT-COFB

As IoT devices are resource-constrained, having multicycle hardware implementations of lightweight ciphers on these embedded systems offer better area, power, and performance efficiency. It offsets the inherent limitations of these devices. Each clock cycle finishes one round of encryption, storing the intermediate result to the round registers. However, the major weakness in multicycle hardware implementation of crypto primitives is that the output for each encryption round is directly assigned to the ciphertext variable [44]. Consequently, by saving the result straight to the ciphertext, the adversary is granted access to the internal rounds, where the intermediate round result is updated at every clock cycle.

3.2.1 Threat Model

The threat model is given to defining the capabilities and intentions of an adversary, and is summarized as follows:

- An adversary has access to a fully functional IoT device where the secret key K is stored in the non-volatile memory (NVM). The possession of this device allows the adversary to perform the chosen-plaintext attack and observe the corresponding ciphertext.
- The adversary can also acquire the gate-level netlist of the corresponding cipher implemented in the IoT device. It can be extracted either through IC reverse engineering [45]

or from GDSII files [46]. An untrusted foundry or a rouge employee of an SoC design house can pirate the GDSII file to an adversary.

3.2.2 Lightweight Hardware Implementation of GIFT-COFB

Under the requirement of authenticated encryption [33], GIFT-COFB supports the encryption of a fixed-length nonce, followed by variable-length associated data and variable-length plaintexts. GIFT-128 [12] block cipher, which consists of 40 encryption rounds is used in GIFT-COFB [47, 48]. Although the typical architecture for the multicycle implementation of block ciphers is to compute every round per clock cycle, it would be sub-optimal for GIFT-COFB. The computation of ciphertext could incur undesired latency for GIFT-COFB in authenticated encryption compared to other finalists whose underlying block ciphers are of a smaller round size [48]. It is possible to incorporate multiple rounds into one clock cycle at the cost of increased area utilization in replicating the round function. Caforio et al. [48] examined different partial r-round unrolling scenarios and the fully unrolled setting of GIFT-COFB and found that the minimum energy consumption of 0.251 nJ/128-bits is observed when two encryption rounds finish at each clock cycle ($r = 2$) along with clock gating and register borrowing.

We focus on this hardware implementation of GIFT-COFB. The datapath of interest is shown in Figure 3.2. Inputs, outputs, and wire names are presented in blue and the module instantiation names are in green. At the start of the authenticated encryption, GIFT-COFB loads a 128-bit nonce into the 128-bit register *state_reg* with selection bit *core_load* at logic 1 for the first clock cycle. The output from *state_reg* is XORed with the 128-bit input data (either the associated data or plaintext) to generate the ciphertext. After the nonce is passed to the *state_reg*, the control signal *core_load* is changed to 0. The default value for control signal *core_done* is logic 0, which allows the output from *state_reg* to pass through the multiplexer and it updates to logic 1 when the encryption reaches the 40^{th} round. Two GIFT round functions, *rf1* and *rounds[1]*, are serially connected. Each round function has two other inputs, the 6-bit round constants and the 128-bit round key. The 128-bit *tag* receives the output from round function *rounds[1]*. These details can be found in *aead.vhd* and *controller.vhd* [47].

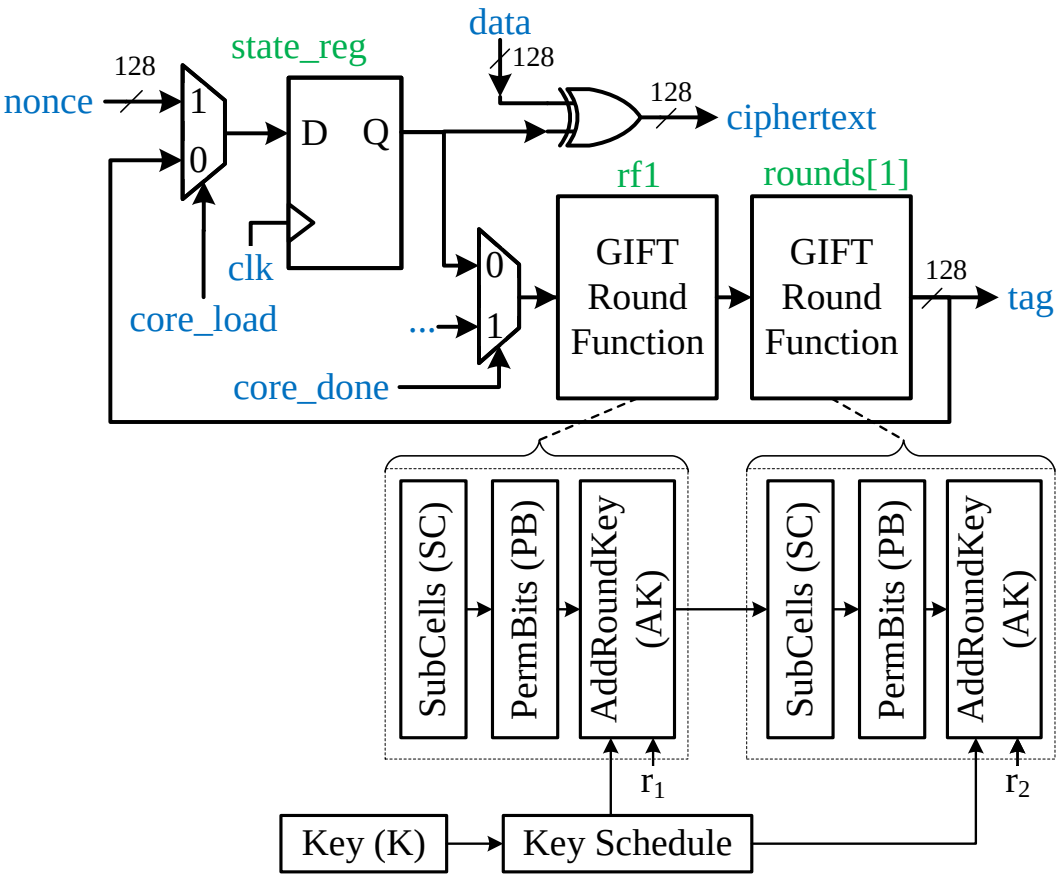

Figure 3.2: Hardware implementation of 2-round partial unrolled GIFT-COFB.

GIFT-COFB begins authenticated encryption by loading nonce and encrypting it with GIFT cipher and the master key K at the speed of 2 encryption rounds per clock cycle. Hence, the attacker could observe the second round result of GIFT's encryption of nonce or any consecutive 2-round (before the reach of 40^{th} round) through output variable *tag*. The remaining question for the adversary is to recover the secret key K under the 2-round encryption of GIFT. Without the loss of generality, we analyze the first two rounds of nonce encryption to explain our attack methodology.

3.3 Chosen-Plaintext Attack on GIFT-COFB

Lightweight crypto modules (e.g., GIFT-COFB) are designed to support authenticated encryption for low-cost IoT devices, which can be deployed in diverse locations. These devices can easily be accessed by an adversary, who can launch an attack to extract the secret key. This

Δx \ Δy	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	2	2	0	2	2	2	2	2	0	0	2
2	0	0	0	0	0	4	4	0	0	2	2	0	0	2	2	0
3	0	0	0	0	0	2	2	0	2	0	0	2	2	2	2	2
4	0	0	0	2	0	4	4	6	0	2	0	0	0	2	0	0
5	0	0	2	0	0	2	2	0	2	0	0	0	2	2	2	4
6	0	0	4	6	0	0	0	2	0	0	2	0	0	0	2	0
7	0	0	2	0	0	2	2	0	2	2	2	4	2	0	0	0
8	0	0	0	4	0	0	0	4	0	0	0	4	0	0	0	4
9	0	2	0	2	0	0	0	2	2	0	2	0	2	2	0	0
a	0	4	0	0	0	0	0	0	0	2	2	0	0	2	2	0
b	0	2	0	2	0	0	0	2	2	2	0	0	2	0	2	0
c	0	0	4	0	4	0	0	0	2	0	2	0	2	0	2	0
d	0	2	2	0	4	0	0	0	0	0	2	2	0	2	0	2
e	0	4	0	0	4	0	0	0	2	2	0	0	2	2	0	0
f	0	2	2	0	4	0	0	0	0	2	0	2	0	0	2	2

Figure 3.3: Differential Distribution Table (DDT) of GIFT Sbox

section presents a novel chosen-plaintext attack that breaks GIFT-COFB on the order of $O(2^4)$. Our attack can recover 4-bit key group in parallel even with the nonlinearity in Sboxes. Let us begin by analyzing the round function to launch the attack.

GIFT round function consists of three steps, SubCells (SC), PermBits (PB), and AddRoundKey (AK), as shown in the dashed boxes in Figure 3.2. SubCells comprises 32 Sboxes, where each 4-bit input cell is transformed by one 4-bit Sbox ($\mathtt{s}(\cdot)$), a non-linear and bijective mapping in $GF(2^4)$. Figure 3.3 shows the differential properties of GIFT Sbox, $D_S(\Delta x, \Delta y) = \sharp\Big\{x \in \mathbb{F}_2^4 \,\Big|\, S(x) \oplus S(x \oplus \Delta x) = \Delta y\Big\}$, in differential distribution table (DDT). PermBits performs bit-level permutation, and AddRoundKey is the modulo-2 addition of the 128-bit from PermBits with a 64-bit round key at bit location $[95...32]$. A 6-bit constant is XORed with bit locations 23, 19, 15, 11, 7, and 3, and constant 1 is XORed with bit location 127. We denote the 128-bit input nonce as N and the 128-bit output after 2 rounds as T, the 64-bit round key associated with the round functions as $\widehat{K^1}$ and $\widehat{K^2}$, and the corresponding round constants as

$\widehat{r^1}$ and $\widehat{r^2}$. In uniformity with the 128-bit vector expressions below, we extend the 64-bit round keys, $\widehat{K^1}, \widehat{K^2}$, to 128-bit K^1 and K^2 by zero-padding into the remaining bit locations. Similarly, we expanded $\widehat{r^1}$ and $\widehat{r^2}$ to the zero-padded 128-bit r^1 and r^2. In the following, we present the detailed steps to obtain the secret key K.

- *Step 1 – Sample collection:* Two tags corresponding to two nonces are collected. The adversary first chooses a nonce, N, passes it to the oracle, and captures the corresponding output tag, T, after the first clock cycle of authenticated encryption. As two encryption rounds are performed in one clock cycle, we can compute the tag as:

$$T = \overbrace{\texttt{PB}(\texttt{SC}(\underbrace{\texttt{PB}(\texttt{SC}(N)) \oplus K^1 \oplus r^1}_{\text{1st round}})) \oplus K^2 \oplus r^2}^{\text{2nd round}} \tag{3.1}$$

  Since SubCells and PermBits are deterministic, the adversary can calculate $\texttt{PB}(\texttt{SC}(N))$ directly. We denote the XOR of r^1 and $\texttt{PB}(\texttt{SC}(N))$ as Q, where $Q = \texttt{PB}(\texttt{SC}(N)) \oplus r^1$. We can then rewrite Equation 3.1 as:

$$T = \texttt{PB}(\texttt{SC}(Q \oplus K^1)) \oplus K^2 \oplus r^2 \tag{3.2}$$

  Under a different input nonce N', $N' \neq N$, the adversary can obtains the 2-round output T' and derive the corresponding $Q' = \texttt{PB}(\texttt{SC}(N')) \oplus r^1$ as seen earlier:

$$T' = \texttt{PB}(\texttt{SC}(Q' \oplus K^1)) \oplus K^2 \oplus r^2 \tag{3.3}$$

  For the adversary, this can be achieved by resetting the GIFT-COFB crypto-system and then providing a difference nonce, N'.

- *Step 2 – Dependency Removal of round key K^2:* The adversary removes the dependency of the second round key K^2 by XORing Equation 3.2 and 3.3,

$$T \oplus T' = \texttt{PB}(\texttt{SC}(Q \oplus K^1)) \oplus \texttt{PB}(\texttt{SC}(Q' \oplus K^1)) \tag{3.4}$$

The linearity of PermBits with its additive property allows us to rewrite Equation 3.4 as,

$$T \oplus T' = \texttt{PB}(\texttt{SC}(Q \oplus K^1) \oplus \texttt{SC}(Q' \oplus K^1)). \tag{3.5}$$

Since bit permutation in PermBits is reversible, it is straightforward for an adversary to derive back to the output from SubCells (SC),

$$\texttt{PB}^{-1}(T \oplus T') = \texttt{SC}(Q \oplus K^1) \oplus \texttt{SC}(Q' \oplus K^1) \tag{3.6}$$

where $\texttt{PB}^{-1}$ is the inverse bitwise permutation of PB. Since $\texttt{PB}^{-1}(T \oplus T')$ is a constant, we denote it as $L = \texttt{PB}^{-1}(T \oplus T')$.

- *Step 3 – Decomposition of K^1 to 16 key cells:* Equation 3.6 can be further split into 32 4-bit cells. The middle sixteen cells can be represented as

$$L_j = \texttt{s}(Q_j \oplus K^1_j) \oplus \texttt{s}(Q'_j \oplus K^1_j) \tag{3.7}$$

where cell index j satisfies $j \in \{23, 22, ..., 16\}$ since the 64-bit round key is only effective at bit indices $[95...32]$. However, it is not possible to uniquely determine each key cell K^1_j with Q_j and Q'_j, where $Q_j \neq Q'_j$, derived from two noncea N and N', where $N \neq N'$ and Equation 3.7 alone, where different values for a key cell K^1_j could lead to the same L_j (see details in Section 3.4).

- *Step 4 – Recovery of each key cell in K^1:* The adversary can now restart the GIFT-COFB with a third nonce, N'', and acquire its *tag* T''. Repeating Steps 1-3, the attacker obtains $Q'' = \texttt{PB}(\texttt{SC}(N'')) \oplus r^1$, $L' = \texttt{PB}^{-1}(T \oplus T'')$, and

$$L'_j = \texttt{s}(Q_j \oplus K^1_j) \oplus \texttt{s}(Q''_j \oplus K^1_j). \tag{3.8}$$

With carefully chosen nonces $\{N, N', N''\}$, each key cell K^1_j can be uniquely recovered with Equation 3.7 and 3.8 by exhaustive search on all 16 combinations of 4-bit key, as

shown in Section 3.4. The attacker can compute all sixteen key cells in K^1 in parallel, since each 4-bit key has its own Equation 3.7 and 3.8 independent of other key bits.

- *Step 5 – Recovery of K^2 and the secret key K:* After deriving the entire round key K^1, the second round key K^2 can be simply obtained through

$$K^2 = T \oplus \texttt{PB}(\texttt{SC}(\texttt{PB}(\texttt{SC}(N)) \oplus K^1)) \oplus r^2, \tag{3.9}$$

where T, $\texttt{PB}(\texttt{SC}(\texttt{PB}(\texttt{SC}(N))$, and K^1 are all known to the attacker. Then, the master key K is recovered through the reversing of key schedule [12] with round keys K^1 and K^2, $K = \{K^2_{[95..64]}||K^1_{[95..64]}||K^2_{[63..32]}||K^1_{[63..32]}\}$, where $||$ denotes the concatenation of bit vectors. The complexity of this attack lies in the recovery of K^1, not K^2. Thus, the attack fully deciphers the secret key K.

3.4 Result and Discussions

In this section, we quantitatively analyze the number of different nonces required for the attacker to derive the secret key K based on the construction of GIFT's Sbox. The computation complexity arises primarily from the computation of round key K^1, since K^2 is computed through Equation 3.9 with $\boldsymbol{O}(1)$. First, we show that having two nonce-tag pairs is insufficient to derive the value for any key cells. When we obtain two different nonce-tag pairs, we can perform Steps 1-3 and proceed to Equation 3.7 of Section 3.3, where each key cell can be solved independently. To cover all the possibilities of Q_j and Q'_j, we exhaustively search all 16 combinations of K^1_j under every possible pair of $\{Q_j, Q'_j\}$ with $Q_j \neq Q'_j$. From our simulation [49], under any fixed $\{Q_j.Q'_j\}$, where $Q_j \neq Q'_j$, at least two solutions exist for a key cell K^1_j giving the same value of L_j.

On the other hand, if the attacker applies three nonce N, N', and N'' and obtains the corresponding tag output, he/she subsequently gets Equation 3.7-3.8 for each key cell from Steps 1-4. Out of all possible, $\binom{16}{3} = 560$, pairs of $\{Q_j, Q'_j, Q''_j\}$, where $Q_j \neq Q'_j \neq Q''_j$, there are 416 pairs that give different $\{L_j, L'_j\}$ for all 16 possible key cells, allowing the adversary

to uniquely determine each 4-bit key [49]. This enables the recovery of one key cell through the exhaustive search of $2^4 = 16$ possible combinations to find the correct solution satisfying Equation 3.7 and 3.8. All sixteen key cells in K^1 can be solved separately and in parallel, resulting in the time complexity of $O(2^4)$ on searching the correct key value. The construction of nonce can be easily derived under the desired values for Q's with $N = \mathtt{SC}^{-1}(\mathtt{PB}^{-1}(Q \oplus r^1))$.

3.5 Summary

this chapter presents a novel chosen-plaintext attack on the energy-efficient hardware implementation of GIFT-COFB, one of the finalists for NIST's Lightweight Cryptography. The proposed attack exploits the 2-round instantiation of GIFT block cipher inside the partial-unrolled structure of GIFT-COFB. The entire secret key can be solved with a minimum of three *nonce-tag* pairs. The key cells can be recovered in parallel by solving the corresponding constraint equations, resulting in a $O(2^4)$ worst-case complexity.

Chapter 4

Fault-Injection Based Chosen-Plaintext Attacks on Multicycle AES Implementations

Advanced Encryption Standard (AES) [50] is one of the most common encryption algorithms used in various applications and protocols, *e.g.,*, disk encryption, Internet Protocol Security, Transport Layer Security, *etc.*. With the objective to speed up the execution of algorithms, there is an increasing demand for creating dedicated hardware [51, 52] other than optimizing software. This holds for cryptographic algorithms as well [53, 54], and their hardware implementations offer better performance on both encryption and decryption. For example, these low-cost hardware implementations of AES can be well suited for different Internet of Things (IoT) and Cyber-Physical Systems (CPS) for enabling data security, which is practically non-existent for low-cost IoT devices [55]. The hardware implementation of AES is generally multicycle, where each round uses the same hardware resource to provide better area efficiency. Due to the extensive use of AES across diverse sectors, hardware designers can reference the open-source HDL designs which are available in OpenCores [56]. These implementations offer designers greater flexibility by choosing the one that matches their design criteria.

Over the years, different researchers have proposed various types of attacks on AES, and successful countermeasures have also been proposed. AES attacks can be divided into three categories – algebraic attacks [57–60], differential fault analysis [61–65], and side-channel attacks [66, 67]. Dunkelman et al. [60] have theoretically explored the effect of excluding the MixColumns transformation. For a reduced-round AES-128 with a single round, they can eliminate the majority of keys by sequentially guessing four bytes in the round key and discard those that failed consistency checks. It needs 2^{16} trial encryptions, and the search complexity is $\boldsymbol{O}(2^{32})$. Bouillaguet et al. [59] require $\boldsymbol{O}(2^{40})$ simulations [68] with one known-plaintext of

a full round of encryption for key derivation. They conjecture the time complexity under the same setup with one more known-plaintext; however, the attack may not uniquely determine the AES key. Besides algebraic analysis, differential fault analysis becomes popular where errors or faults are introduced either inside the computation of a particular round or within the Key Schedule algorithm. Blömer et al. [61] proposed an attack on AES by resetting a bit after the XOR of input key and plaintext to zero. Observing the difference in output ciphertext, it helps the attacker decipher one key bit per fault. However, this attack needs to inject faults at the metal wires with extreme timing precision [69]. Moradi et al. [62] and Pogue et al. [70] perform differential fault analysis to extract the secret key, where faults are assumed at the encryption rounds. Ali et al. [64], Giraud et al. [65], and Kim [63] have successfully retrieved the key with differential fault analysis when faults occur at the Key Schedule. Multiple fault detection schemes [71–73] have been proposed to identify faults through either error detection codes or partial replication of the internal computation of AES. Several fault-resilient AES implementations have also been proposed [74–76].

This chapter is organized as follows. We briefly introduce the background for AES and describe the threat model in Section 4.1. Our proposed attacks are presented in Section 4.2. A theoretical perspective for our proposed attacks is analyzed in Section 4.3. Finally, we conclude the chapter in Section 4.4.

4.1 Existing AES Implementation, Notations, and Threat Model

The hardware implementations of AES are often multicycle, where area efficiency is assured since all the rounds use the same resources at different clock cycles. Depending on the key size, the ciphertext is produced after 10, 12, or 14 clock cycles. If each encryption takes more than one clock cycle to finish, the ciphertext is generated after an integer multiple of 10, 12, or 14 clock cycles for AES-128, AES-192, and AES-256, respectively, as illustrated in Figure 4.1. Each encryption round contains SubBytes (SB), ShiftRows (SR), MixColumns (MC), and AddRoundKey ($\bigoplus$) modules, while the last one skips MC, as shown inside the dashed box of Figure 4.2.

Notations: We use the following notations to maintain uniformity across the entire paper.

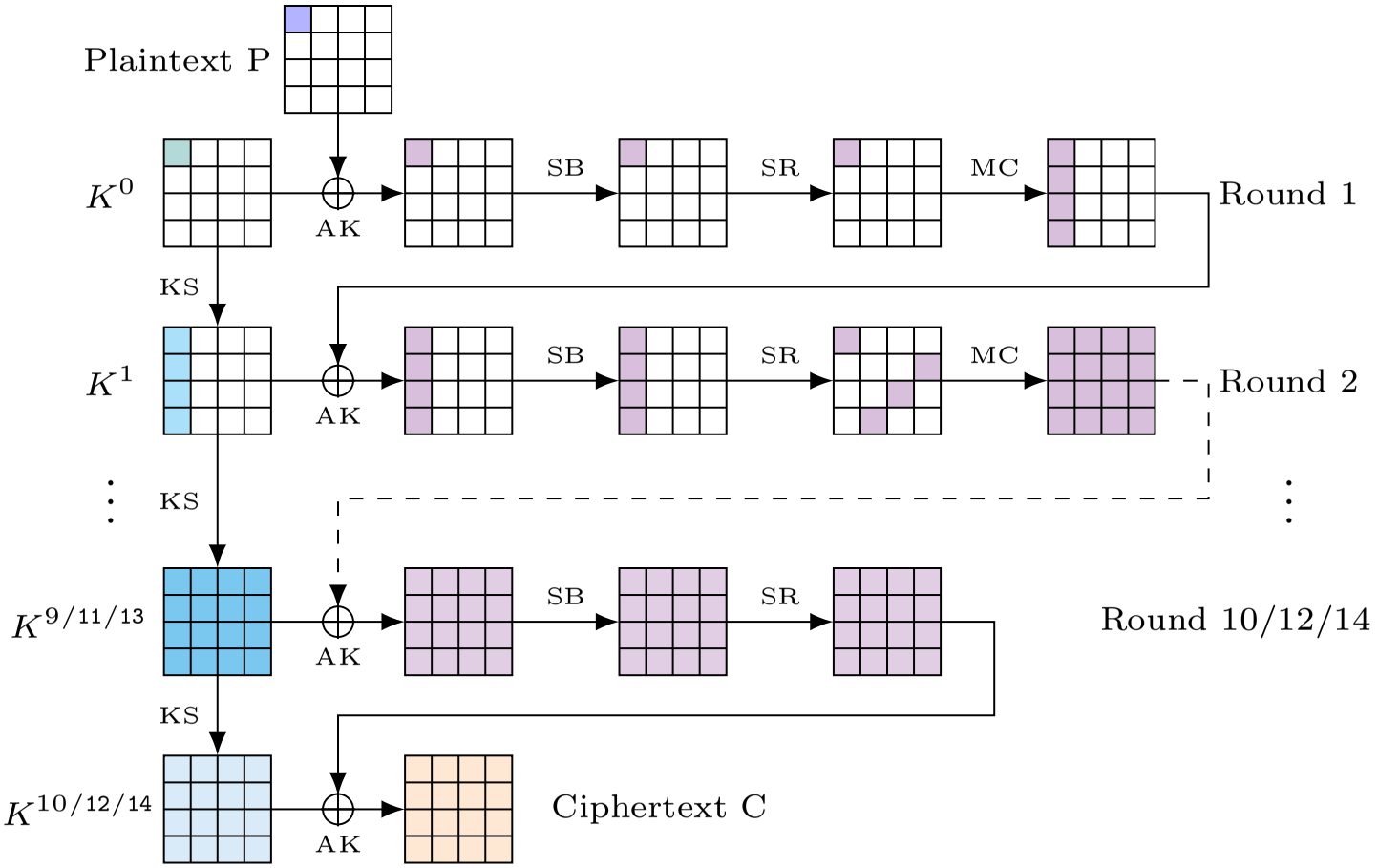

Figure 4.1: Encryption rounds in AES-128, AES-192, and AES-256.

- We adopt the following notations, where the superscript index (i) in a variable indicates the current encryption round and the subscript index (j) are for the byte index, from 0 to 15. For example, the j^{th} byte in i^{th} round key is K_j^i and R^i is the result for i^{th} encryption round. We use K, without any superscript, to refer to the input key, which can be either 16, 24, or 32-byte for AES-128, AES-192, or AES-256. To facilitate the derivation in the subsequent sections, we abbreviate the round one result R^1 to R. Aside from the input key, the size for all other variables, *e.g.,*, plaintext P, round register R^i, round key K^i, and ciphertext C, are 16 bytes. Note that, in this chapter, the ciphertext register C may not contain the actual encrypted data. C, as explained in Section 4.2, can store the internal round result, before the complete encryption finishes. We use ciphertext and round one result R interchangeably.

- The S-box function is denoted as $s(\cdot)$. Round constant is RC.

4.1.1 Threat Model

As the hardware implementations of different crypto primitives become prevalent, an adversary can launch the attacks by physically accessing the device. This section describes the adversarial capabilities that help to carry out the attacks. The threat model is summarized as follows:

- The attacker possesses a fully functional chip where the secret key has already been programmed. For example, an electronic device that ensures secure communication can be obtained from an IoT/CPS application. By having the device, an adversary can apply a plaintext and observe its corresponding ciphertext.

- The adversary can obtain the gate-level netlist of the AES implementation. It can be either acquired through IC reverse engineering [45] or from the GDSII files [46]. As the majority of the IC production is offshore, an untrusted foundry can also provide the reverse-engineered netlist to the adversary.

- The adversary can have access to the design-for-testability (DFT) or scan architecture to observe the internal state of the design (e.g., round registers). He/she can launch our

proposed first attack to obtain the secret key. Note that the DFT architecture provides the necessary support for manufacturing tests [77]. If not, the attacker can use fault injection equipment to inject a fault in the completion-indicator (CI) register to launch our proposed second attack. Laser fault injection equipment can induce very precise faults and target a single flip-flop [78]. As this equipment is available at universities, we assume that an adversary also has the means to acquire such equipment.

4.2 Proposed Attacks on Multicycle AES Implementations

We present two attacks to efficiently break multicycle AES implementations. The first attack exploits the minor issues in the implementations [79–81], where the round registers and ciphertext are updated simultaneously in every round. The second attack, however, requires fault injection for breaking a correct multicycle AES implementation which assigns the round result to the ciphertext output only when it is in the final round [82]. Both attacks work on all three key sizes of AES. For simplicity of discussion, in this section, we will first show all the attacks on AES-128. The same attack methodology can be applied to AES-192 and AES-256 with constant overhead in worst-case search complexity, which we briefly describe how to extend both attacks to key sizes larger than 128-bit at the end.

4.2.1 Proposed Exhaustive Key-Byte Search Attack

Our first attack is specific to the multicycle AES implementations, where the output for each encryption round can be observed as ciphertext C, where Figure 4.3 illustrates the first two encryption rounds of AES, the target of this attack. Figure 4.2 shows an abstract view of the multicycle AES implementation when a designer incorrectly implements AES or an adversary has access to the internal scan chains. For example, one implementation from OpenCores [79] assigns the i^{th} round result, $MC^i \oplus K^i$, straight to the ciphertext output, and other implementations [80, 81] connect the ciphertext to the round registers. Both data-paths are highlighted in blue in Figure 4.2.

The traditional AES implementation mixes different key bytes in such a way that an adversary cannot remove the interdependency among the key bytes in the ciphertext. As a result,

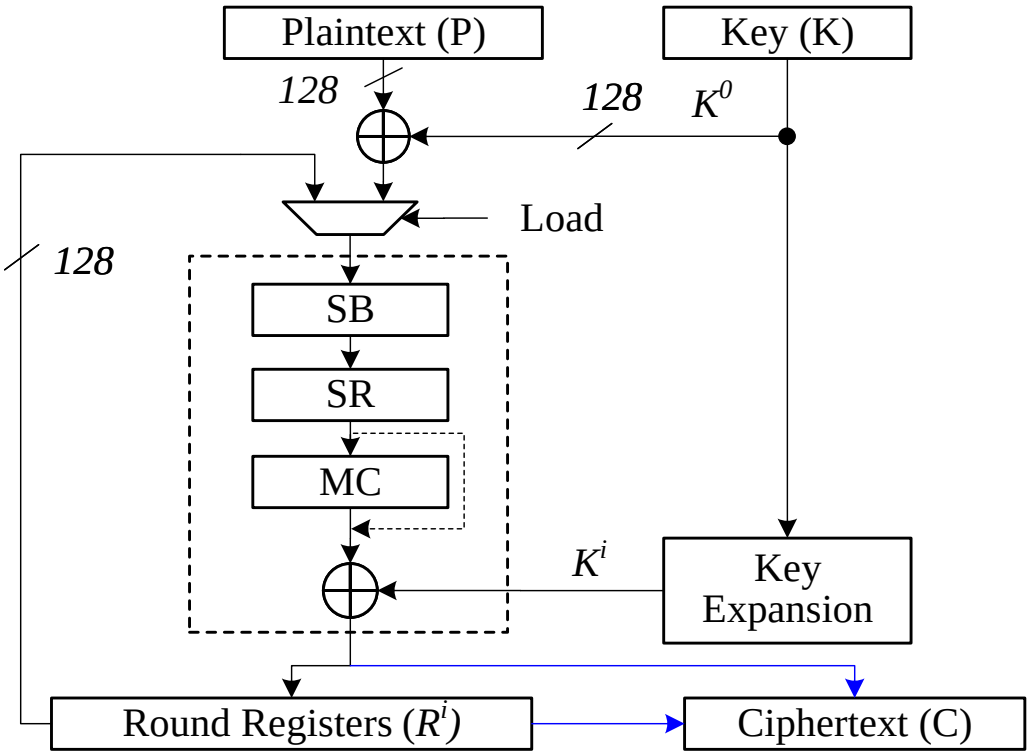

Figure 4.2: Hardware implementation of multicycle AES-128.*

AES remains secure no matter how many plaintext-ciphertext pairs one can observe. However, if an adversary can observe the round outputs stored in the round registers of a multicycle AES implementation, it is possible to remove the dependency across the key bytes. We show that one key byte can be determined without knowing the other key bytes in this attack. Thereby, we determine a key byte through simulating all 2^8 key combinations and compare the result with one byte round register value from the working chip under attack to derive the correct key. In the following, we present the detailed steps to obtain the first key byte, K_0. Similar analysis can be performed to reveal the other key bytes.

- Step-1: The adversary chooses two plaintexts, P and P', and observes the two corresponding round outputs R and R' after the first clock cycle from the chip.

*This figure is based on AES designs from OpenCores [79–81].

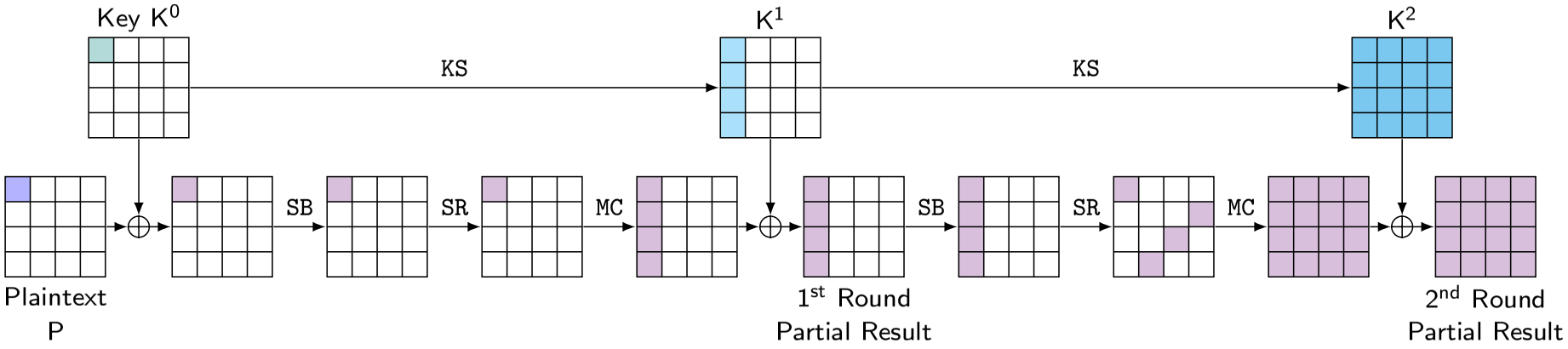

Figure 4.3: First two encryption rounds of AES.

- Step-2: The first byte (R_0) of round output R is computed using the following equation:

$$\begin{aligned} R_0 &= \{MC(SR(SB(P \oplus K^0))) \oplus K^1\}_0 \\ &= \{MC(SR(SB(P \oplus K^0)))\}_0 \oplus K_0^1 \\ &= [02 \otimes s(P_0 \oplus \boldsymbol{K_0}) \oplus 03 \otimes s(P_5 \oplus \boldsymbol{K_5}) \oplus (P_{10} \oplus \boldsymbol{K_{10}}) \oplus \\ &\quad s(P_{15} \oplus \boldsymbol{K_{15}})] \oplus [K_0 \oplus s(K_{13}) \oplus RC]. \end{aligned} \tag{4.1}$$

From Equation 4.1, we can observe that the value of R_0 depends on K_0, K_5, K_{10}, K_{13}, and K_{15}. Key bytes which can be derived from the equation are highlighted as boldface letters. At this point, it is sub-optimal to brute force all five key bytes as they cannot be uniquely determined. Multiple collisions occur for the 2^{40} key combinations that lead to the same 8-bit R_0 value. Therefore, we choose another plaintext P' with the following properties:

$$P_0 \neq P_0', \text{ and } P_i = P_i', \;\; i = 5, 10, 15$$

The R_0' can be computed using the following equation:

$$\begin{aligned} R_0' &= \{MC(SR(SB(P' \oplus K^0))) \oplus K^1\}_0 \\ &= \{MC(SR(SB(P' \oplus K^0)))\}_0 \oplus K_0^1 \\ &= [02 \otimes s(P_0' \oplus \boldsymbol{K_0}) \oplus 03 \otimes s(P_5 \oplus \boldsymbol{K_5}) \oplus (P_{10} \oplus \boldsymbol{K_{10}}) \oplus \\ &\quad s(P_{15} \oplus \boldsymbol{K_{15}})] \oplus [K_0 \oplus s(K_{13}) \oplus RC]. \end{aligned} \tag{4.2}$$

The computation details can be found in [50].

- Step-3: R_0 and R'_0 are XORed to remove the dependency for other key bytes.

$$\begin{aligned} R_0 \oplus R'_0 &= [02 \otimes s(P_0 \oplus \boldsymbol{K_0}) \oplus 03 \otimes s(P_5 \oplus \boldsymbol{K_5}) \oplus (P_{10} \oplus \boldsymbol{K_{10}}) \\ &\quad \oplus s(P_{15} \oplus \boldsymbol{K_{15}})] \oplus [K_0 \oplus s(K_{13}) \oplus RC] \oplus [02 \\ &\quad \otimes s(P'_0 \oplus \boldsymbol{K_0}) \oplus 03 \otimes s(P_5 \oplus \boldsymbol{K_5}) \oplus (P_{10} \oplus \boldsymbol{K_{10}}) \oplus \\ &\quad s(P_{15} \oplus \boldsymbol{K_{15}})] \oplus [K_0 \oplus s(K_{13}) \oplus RC] \\ &= 02 \otimes s(P_0 \oplus \boldsymbol{K_0}) \oplus 02 \otimes s(P'_0 \oplus \boldsymbol{K_0}) \end{aligned} \tag{4.3}$$

We can rewrite Equation 4.3 as:

$$s(P_0 \oplus \boldsymbol{K_0}) \oplus s(P'_0 \oplus \boldsymbol{K_0}) = M_0 \tag{4.4}$$

where, M_0 is a constant.

- Step-4: Brute-force attack is performed using Equation 4.4.

The only unknown in Equation 4.4 is K_0, allowing the attacker to enumerate all 256 combinations of K_0 to find the one that satisfies it. However, there exists more than one solution due to the nonlinearity introduced by the S-box.

Claim-1. There exist two solutions for Equation 4.5.

$$s(p \oplus \boldsymbol{k}) \oplus s(p' \oplus \boldsymbol{k}) = m \tag{4.5}$$

where, p, p', k, and m are of one byte, and $p \neq p'$.

Observation. There are 128 unique values for byte m for all 256 combinations of k under any fixed p, p' and $p \neq p'$. One can verify the above observation using code available in GitHub [83]. We denote the 128 unique values of m as m_i, $i \in 0, 1, ..., 127$ and $m_i \neq m_j$ when $i \neq j$.

Proof. For each valid m_i, there is at least one unique key k_i^I, $i \in 0, 1, ..., 127$, that satisfies:

$$s(p \oplus k_i^I) \oplus s(p' \oplus k_i^I) = m_i \tag{4.6}$$

Due to $m_i \neq m_j$, when $i \neq j$, and the bijective property of the S-box, $k_i^I \neq k_j^I$, when $i \neq j$, with fixed p, p', $p \neq p'$. We denote the set of these 128 solutions as Group *I*. The proof for the existence of another solution for each m_i is sufficient for validating the claim.

Let us consider another key byte k_i^{II} with the form $k_i^{II} = k_i^I \oplus p \oplus p'$. Clearly, $k_i^{II} \neq k_i^I$ since $p \neq p'$.

Applying the value of k_i^{II} in Equation 4.6, we compute:

$$\begin{aligned} & s(p \oplus k_i^{II}) \oplus s(p' \oplus k_i^{II}) \\ & = s(p \oplus k_i^I \oplus p \oplus p') \oplus s(p' \oplus k_i^I \oplus p \oplus p') \\ & = s(k_i^I \oplus p') \oplus s(k_i^I \oplus p) = m_i \end{aligned} \tag{4.7}$$

Next, we show that all these 128 k_i^{II}'s are unique as well. Now consider any two solutions k_i^{II} and k_j^{II}, $\forall i \neq j$. As $k_i^I \neq k_j^I$, then $(k_i^I \oplus p \oplus p') \neq (k_j^I \oplus p \oplus p')$. This results $k_i^{II} \neq k_j^{II}$. This proves that no two k_i^{II}, $k_j^{II}, i \neq j$ are the same, and we denote the set of these 128 solutions as Group *II*.

Finally, the proof will be complete, if we show that there is no overlap between the solutions in Group *I* (k_i^I's) and Group *II* (k_i^{II}'s), where each group contains 128 unique key within. Let us assume that k_i^{II} belongs to Group *I* and denote with index j, where $i \neq j$, $k_i^{II} = k_j^I$. Substitute k_j^I in Equation 4.6, and we get

$$s(p \oplus k_j^I) \oplus s(p' \oplus k_j^I) = m_j. \tag{4.8}$$

Combining Equation 4.8 and Equation 4.7 for k_i^I, we have $m_i = m_j$, which contradict the uniqueness of 128 m_i's for $i \neq j$. Thus, there are no common solutions between Group

I and Group *II* and the 256 solutions from the joint groups make up all the possible key-byte combinations. Therefore, we proved that there exist only two solutions for each m_i that satisfies Equation 4.5. □

- Step-5: The double-solution is removed by selecting another plaintext (P'') with P''_0 differs from both P_0 and P'_0 (i.e., $P''_0 \neq P_0 \neq P'_0$), and keeping $P''_5 = P_5$, $P''_{10} = P_{10}$, and $P''_{15} = P_{15}$ unchanged. Using Step-2 and Step-3, we obtain the following equation:

$$s(P_0 \oplus \boldsymbol{K_0}) \oplus s(P''_0 \oplus \boldsymbol{K_0}) = N_0, \tag{4.9}$$

where, N_0 is a constant. Equation 4.9 is applied on the two previously obtained solutions (i.e., K_0^I and K_0^{II}) to determine the correct key byte.

Claim-2. Both solutions, K_0^I and K_0^{II}, cannot be valid under both Equations 4.4 and 4.9.

Proof. Let us assume that both solutions, K_0^I and K_0^{II}, are valid and satisfy Equation 4.9. As a result, we can write,

$$\begin{aligned} s(P_0 \oplus K_0^I) \oplus s(P''_0 \oplus K_0^I) &= N_0 \\ \text{and} & \\ s(P_0 \oplus K_0^{II}) \oplus s(P''_0 \oplus K_0^{II}) &= N_0 \end{aligned}$$

Using Claim-1, we can write $s(P_0 \oplus K_0^I) = s(P''_0 \oplus K_0^{II})$. Also, with Claim-1 and Equation 4.4, we can write $s(P_0 \oplus K_0^I) = s(P'_0 \oplus K_0^{II})$. This results, $s(P'_0 \oplus K_0^{II}) = s(P''_0 \oplus K_0^{II})$. This can't be true as $P' \neq P''$. □

In the same manner, key bytes K_5, K_{10}, and K_{15} are determined through either of the first four-bytes from round output, R_0, R_1, R_2, R_3, by varying the corresponding plaintext byte, P_5, P_{10}, or P_{15} and constraining the other three plaintext bytes to remain unchanged.

To find the remaining key bytes, we need to consider three bytes of the round register, one from $R_4 - R_7$, $R_8 - R_{11}$, and $R_{12} - R_{15}$ each. These three bytes are sufficient to find

the remaining key bytes as their MixColumns transformation incorporate all 12 key bytes. For example, we consider R_4, R_8, and R_{12}, as shown below:

$$\begin{aligned} R_4 &= [02 \otimes s(P_4 \oplus \boldsymbol{K_4}) \oplus 03 \otimes s(P_9 \oplus \boldsymbol{K_9}) \oplus s(P_{14} \oplus \boldsymbol{K_{14}}) \oplus \\ &\quad s(P_3 \oplus \boldsymbol{K_3})] \oplus [K_4 \oplus K_0 \oplus s(K_{13}) \oplus RC] \qquad (4.10) \\ R_8 &= [02 \otimes s(P_8 \oplus \boldsymbol{K_8}) \oplus 03 \otimes s(P_{13} \oplus \boldsymbol{K_{13}}) \oplus s(P_2 \oplus \boldsymbol{K_2}) \oplus \\ &\quad s(P_7 \oplus \boldsymbol{K_7})] \oplus [K_8 \oplus K_4 \oplus K_0 \oplus s(K_{13}) \oplus RC] \qquad (4.11) \\ R_{12} &= [02 \otimes s(P_{12} \oplus \boldsymbol{K_{12}}) \oplus 03 \otimes s(P_1 \oplus \boldsymbol{K_1}) \oplus s(P_6 \oplus \boldsymbol{K_6}) \oplus \\ &\quad s(P_{11} \oplus \boldsymbol{K_{11}})] \oplus [K_{12} \oplus K_8 \oplus K_4 \oplus K_0 \oplus s(K_{13}) \oplus RC] \qquad (4.12) \end{aligned}$$

The remaining key bytes can be determined iteratively using Equations 4.10-4.12 and Steps 1-5. Note that an adversary can also choose R_1, R_5, R_9 and R_{13} or a few other combinations to determine all 16 key bytes as well.

4.2.2 Proposed Fault-Injection Attack

The first attack is efficient in determining the key when an adversary can access the round registers that hold the output of each round. An adversary can use the scan architecture or exploit a faulty implementation for such a purpose. However, one cannot always assume access to registers. This motivates us to propose a fault injection attack that allows us to observe the internal state and utilize the previously presented brute-force attack. Note that the fault injection has become an effective means to launch an attack. It has been demonstrated that laser fault injection can successfully target a single register [78]. The same methodology and procedure in [78] is applicable to launching our proposed fault injection attack on AES. This attack leads to two possible scenarios, and both are elaborated below.

First, let us examine an example in OpenCores [82], which does not have the weakness of other implementations described in Section 4.2.1. Figure 4.4 shows this multicycle AES implementation where the ciphertext register receives the round register value at the last encryption round (e.g., when *done_d1 == 1* and *done_d2 == 0*). We also assume that an adversary does

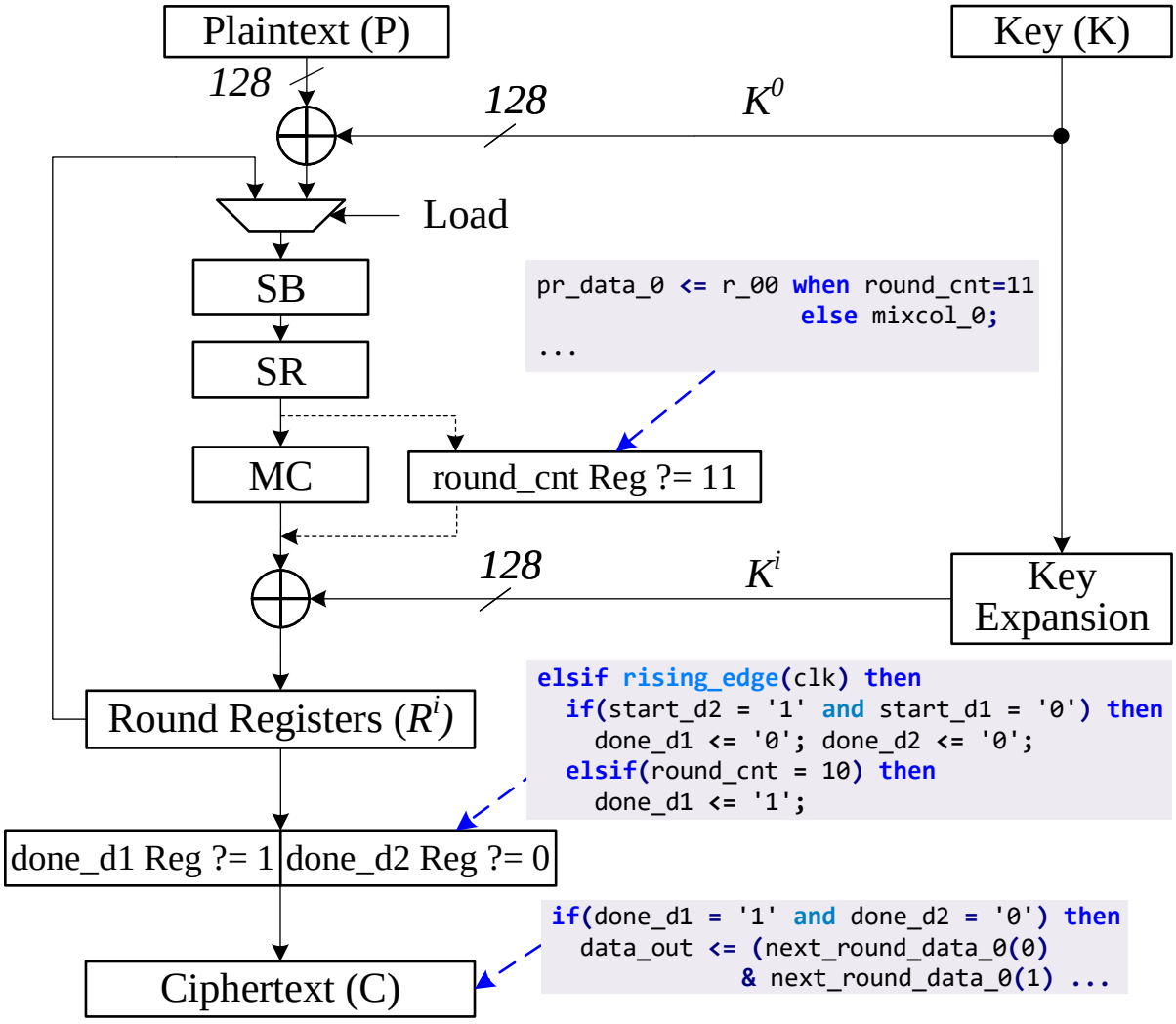

Figure 4.4: Structure of multicycle AES-128, where round result R^i is sent to ciphertext C only after the last round of encryption, code segment from Lines 319-326, 399-428.*

not have access to the internal scan chains and only observes the ciphertexts. The round operations are the same with Figure 4.2. We include the HDL code excerpt [82]. in Figure 4.4 which describes how the completion-indicator (CI) registers (e.g., *done_d1* and *done_d2*), and the ciphertext (i.e., *data_out*) are updated. As the *done_d2* register holds a logic *0* value during the round operations, the round registers values (e.g., *next_round_data_0*), are propagated to the ciphertext output (e.g., *data_out*) when *done_d1=1*. It is thus sufficient to inject only one logic *1* fault to *done_d1* register to extract round register value. Once the internal value is observed, an adversary can perform Steps 1-5 presented in Section 4.2.1 to retrieve the secret key completely.

Second, a hardware implementation can have the same logical condition applied to both skipping the MixColumns transformation and assigning the round register result to the ciphertext C, since both should happen at the last encryption round. In this way, if the attacker injects all the necessary fault to force the round result observable, the MixColumns step is also affected and bypassed. Alternatively, it may be possible for an adversary to inject faults on both CI register and round counter (e.g., *round_cnt*) so that he/she obtains the result from the first encryption round, but skips the MixColumns module. We briefly describe how an attack can be launched when bypassing the MixColumns operation.

• **Brute-force attack without MixColumns Operation.** Let us consider the first byte of round register R (i.e., R_0) after applying the first plaintext P, which can be computed as:

$$R_0 = s(P_0 \oplus \boldsymbol{K_0}) \oplus [K_0 \oplus s(K_{13}) \oplus RC].$$

After applying the second plaintext P' with $P_0 \neq P'_0$, we can write:

$$R'_0 = s(P'_0 \oplus \boldsymbol{K_0}) \oplus [K_0 \oplus s(K_{13}) \oplus RC].$$

Finally, we obtain

$$s(P_0 \oplus \boldsymbol{K_0}) \oplus s(P'_0 \oplus \boldsymbol{K_0}) = M_0$$

where, M_0 is a constant.

Then, we can follow the same procedure described in Steps 4-5 in Section 4.2.1 to recover key byte K_0.

4.2.3 Extending the Proposed Attacks to AES-192 and AES-256

The attacks presented in Sections 4.2.1-4.2.2 can retrieve the entire 16 bytes of the secret key for AES-128. It can also recover the first 16 bytes from AES-192 and AES-256, although the exact expression for round key in Key Expansion is different. However, it is necessary to extract the remaining 8 and 16 bytes for 192- and 256-bit keys, respectively. These key bytes belong to the second round key K^1, where they influence the result of the first encryption round

through AddRoundKey ($\bigoplus$). As a result, the adversary can decipher these key bytes from any one of the plaintext-ciphertext pairs obtained in Section 4.2.1 or 4.2.2, without the need to give additional plaintexts to the oracle or perform extra fault injections. For AES-192, the first eight bytes of the round key K^1, $K_0^1, ..., K_7^1$, are the last 8 bytes of the input key K, $K_{16}, ..., K_{23}$, respectively [50]. Likewise, the entire round key K^1 is nothing but the last 16 bytes of input key K for AES-256, $K^1 = \{K_{16}, ..., K_{31}\}$ [50]. We show how to determine key byte K_{16} of AES-192 from the observed R in the following:

$$\begin{aligned} R_0 &= \{MC(SR(SB(P \oplus K^0))) \oplus K^1\}_0 \\ &= \{MC(SR(SB(P \oplus K^0)))\}_0 \oplus K_0^1 \\ &= [02 \otimes s(P_0 \oplus K_0) \oplus 03 \otimes s(P_5 \oplus K_5) \oplus (P_{10} \oplus K_{10}) \oplus \\ &\quad s(P_{15} \oplus K_{15})] \oplus [\boldsymbol{K_{16}}] \\ &= \boldsymbol{K_{16}} \oplus Q_1, \end{aligned} \tag{4.13}$$

where $Q_1 = 02 \otimes s(P_0 \oplus K_0) \oplus 03 \otimes s(P_5 \oplus K_5) \oplus (P_{10} \oplus K_{10}) \oplus s(P_{15} \oplus K_{15})$ is a constant as $K_0 - K_{15}$ are known. One can directly compute K_{16} by XORing R_0 and Q_1.

It is also possible to determine K_{16} directly, if adversary chooses to bypass the Mix-Columns operation. We can also write:

$$R_0 = s(P_0 \oplus K_0) \oplus [\boldsymbol{K_{16}}], \tag{4.14}$$

where $s(P_0 \oplus K_0)$ is a constant as $K_0 - K_{15}$ are known. The key byte K_{16} can be computed by XORing R_0 and $s(P_0 \oplus K_0)$ like before.

The other key bytes (i.e., $K_{17} - K_{23}$) can be obtained similarly from $R_1 - R_7$. One can perform similar analysis to obtain the remaining $K_{16} - K_{31}$ key bytes from $R_0 - R_{15}$ for AES-256.

4.2.4 Number of Plaintext Requirement

In Section 4.2.1, three plaintexts are sufficient to derive one key byte. Note, in these three plaintexts, we only vary one byte of the same index and constrain the bytes at the three other indices to remain unchanged. Because we do not have constraints on all 15 remaining plaintext bytes, we are allowed with more flexibility on other plaintext bytes that do not belong to the same MixColumns computation as the key byte of interest. Instead of the apparent $3 * 16 = 48$ plaintexts to recover all 16-byte key, we can reduce this number by having four plaintext bytes, where no two bytes resides in the same MixColumns operation, vary concurrently in one plaintext, *e.g.,*, P_0, P_4, P_8, P_{12}. Hence, the minimum number of plaintexts needed for this attack is $1+2\times4 = 9$, where the minimal three plaintexts are satisfied by having one reference plaintext and its corresponding two variants. Once all nine ciphertexts are obtained, all sixteen key bytes can be determined in parallel, making the worst-case complexity of $\boldsymbol{O}(2^8)$. Suppose the round result skips MixColumns transformation, as of the second possible scenario in Section 4.2.2, each key byte is still recovered with three plaintext-ciphertext pairs. However, since we do not have the restriction on the other three plaintext bytes as for Section 4.2.1, it does not matter if other bytes stay the same or not. Hence, we can reduce the required number of plaintexts to break the entire key from 9, as for the attack in Section 4.2.1, down to 3, as long as the byte at the same index (*e.g.,*, index j) is different in all three plaintexts, $P_j \neq P'_j \neq P''_j$. The worst-case complexity of this attack is still $2^8 = \boldsymbol{O}(2^8)$, since all sixteen key bytes, K_j's, can be recovered concurrently, without the need to wait for any other bytes to be resolved first.

4.3 Theoretical Justification of Double-Solution for Equation 4.5

Aside from the exhaustive simulation we performed in Section 4.2.1, we present another perspective on the dual solutions, k^I and k^{II}, for Equation 4.5. Instead of the common approach [57] to expand the S-box to a system of equations in a bit-by-bit manner, we consider the polynomial in $\boldsymbol{GF}(2^8)$ as the fundamental unit. To differentiate matrix multiplication from polynomial multiplication under $\boldsymbol{GF}(2^8)$ with irreducible polynomial $IP = [1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1]$, we use $\cdot$ for matrix multiplication. S-box $s(x) = y$ contains two operations [50]. It first find

the inverse polynomial, x^{-1} of its input byte x under $\boldsymbol{GF}(2^8)$ and IP. Then, it applies the affine transformation on x^{-1} with a reversible matrix H of size 8×8 bits and an 8-bit column vector $c = [1\ 1\ 0\ 0\ 0\ 1\ 1\ 0]^T$ to get output byte $y = H \cdot x^{-1} \oplus c$ [50].

For our analysis, we can expand Equation 4.5 with the details of the internal construction of S-box as:

$$\big(H \cdot (p \oplus \boldsymbol{k})^{-1} \oplus c\big) \oplus \big(H \cdot (p' \oplus \boldsymbol{k})^{-1} \oplus c\big) = m.$$

Note that $(p \oplus \boldsymbol{k})^{-1}$ and $(p' \oplus \boldsymbol{k})^{-1}$ are the inverse of $(p \oplus \boldsymbol{k})$ and $(p' \oplus \boldsymbol{k})$, respectively. After rearranging terms, we get

$$H \cdot \big((p \oplus \boldsymbol{k})^{-1} \oplus (p' \oplus \boldsymbol{k})^{-1}\big) = m.$$

Since 8-by-8 bit matrix H has inverse, denoted as H^{-1}, we obtain

$$(p \oplus \boldsymbol{k})^{-1} \oplus (p' \oplus \boldsymbol{k})^{-1} = H^{-1} \cdot m.$$

Now, both sides of the equation are polynomials in $\boldsymbol{GF}(2^8)$. We use constant d to represent $H^{-1} \cdot m$, $d = H^{-1} \cdot m$, for clarity.

If we multiply both side with polynomial $(p \oplus \boldsymbol{k})$ and $(p' \oplus \boldsymbol{k})$, we get

$$(p' \oplus \boldsymbol{k}) \oplus (p \oplus \boldsymbol{k}) = d \otimes (p \oplus \boldsymbol{k}) \otimes (p' \oplus \boldsymbol{k}).$$

We can further simplify it as

$$p \oplus p' = d \otimes \big[(\boldsymbol{k} \otimes \boldsymbol{k}) \oplus (p \oplus p') \otimes \boldsymbol{k} \oplus (p \otimes p')\big].$$

Since d is a polynomial under $\boldsymbol{GF}(2^8)$ and it is not the zero polynomial, the inverse of d exists and we denote it as d^{-1}. Multiply both side with d^{-1} and abbreviate $k \otimes k$ as k^2 under $\boldsymbol{GF}(2^8)$, we have

$$\boldsymbol{k}^2 \oplus (p \oplus p') \otimes \boldsymbol{k} \oplus (p \otimes p') \oplus d^{-1} \otimes (p \oplus p') = 0.$$

Thus, we complete the derivation and it is clear that Equation 4.5 is a quadratic equation with respect to the unknown variable $\boldsymbol{k}$.

Under any quadratic equation $x^2 + ax + b = 0$ in $\mathbb{R}$, the two roots x_1, x_2 satisfy $x_1 + x_2 = -b, x_1 \times x_2 = c$. We made an interesting observation that these properties also hold true for Equation 4.3. The two solution for Equation 4.5, k^I, k^{II} uphold both

$$\begin{aligned} k^I + k^{II} &= p \oplus p', \text{ and} \\ k^I \otimes k^{II} &= (k^{II} \oplus p \oplus p') \otimes k^{II} \\ &= (k^{II})^2 \oplus (p \oplus p') \otimes k^{II} \\ &= (p \otimes p') \oplus d^{-1} \otimes (p \oplus p'). \end{aligned}$$

4.4 Summary

In this chapter, we presented two novel attacks targeting the hardware implementations of multicycle AES. In both attacks, each key byte requires only three plaintext-ciphertext pairs to retrieve its value. The entire secret key is recovered by solving all key bytes in parallel, resulting in a $O(2^8)$ worst-case complexity. If the internal round result is not observable in the output, we propose to inject fault on the completion-indicator register to reveal the internal state. Any traditional method can be applied to inject faults, and the protection against the fault injection attacks can be bypassed since no intermediate result is affected. We also showed the algebraic perspective on the dual solutions of Equation 4.5. Finally, we provide the theoretical extension of the properties of a regular quadratic equation to the finite field $\boldsymbol{GF}(2^8)$, which support Claim-1.

Chapter 5

Complexity Analysis of the SAT Attack on Logic Locking

The integrated circuits (ICs) are fundamental to virtually every technology in the Department of Defense (DoD), industrial and commercial spaces. Moore's Law has guided the microelectronics industry for decades to enhance the performance of ICs. The continuous addition of new functionalities in SoCs has forced design houses to adopt newer and lower technology nodes to increase operational speed, reduce power consumption, overall die area, and the resultant cost of a chip. This exponential growth becomes feasible due to the globalization of semiconductor design, manufacturing, and test processes. Building and maintaining a fabrication unit (foundry) requires a multi-billion dollar investment [14]. As a result, a system-on-a-chip (SoC) design house acquires intellectual properties (IPs) from many vendors and sends the design to a foundry for manufacturing, typically located offshore due to the horizontal integration in the semiconductor industry. At present, the majority of the SoC design houses no longer design the complete SoC and manufacture chips on their own. As a result, the trusted foundry model is no longer assumed to be valid for producing ICs, where the trustworthiness of microelectronic parts is often questioned.

Due to the outsourced IC design and fabrication, the underlying hardware in various information systems that were once trusted can no longer be so. The untrusted chip fabrication and test facilities represent security threats to the current horizontal integration. The security threats posed by these entities include: (i) overproduction of ICs, where an untrusted foundry fabricates more chips without the consent of the SoC design house to generate revenue by selling them in the market [84–90], and (ii) piracy of IPs, where an entity in the supply chain can use, modify and/or sell functional IPs illegally [21,22,91,92]. An untrusted foundry has access to all

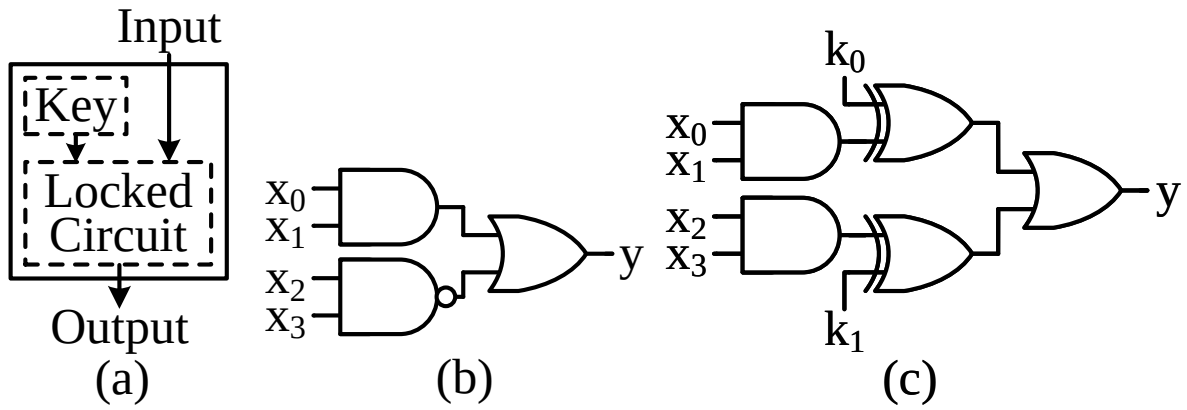

Figure 5.1: Overview of logic locking. (a) Architecture of a locked circuit. (b) Original design. (c) XOR-based locking, with secret key of $k_0k_1 = 01$.

the mask information constructed from the GDSII or OASIS files and then reconstructs all the layers and the complete netlist with advanced tools [46]. In addition, reverse engineering (RE) of ICs becomes feasible even for advanced technology nodes due to the advancement of the tools for the decapsulation of the ICs and imaging. RE is commonly used in the semiconductor industry to perform failure analysis, defect identification, and verify intellectual property (IP) infringement [93, 94]. Unfortunately, the same RE can be exploited by an adversary to reconstruct the gate-level netlist from a chip [45].

One of the best ways to prevent an adversary from cloning a netlist (either by an untrusted foundry or a reverse engineer) is to hide or obfuscate the circuit. The attacker cannot decode the original functionality even after extracting the netlist from RE. Logic locking promises to hide the inner details of a circuit by inserting a set of key gates. The only way to recover the original functionality is by applying a secret key stored in a tamper-proof memory of the chip. Figure 7.1 shows an abstract representation of logic locking. In addition to logic locking, hardware watermarking [95–97] could identify and prevent copying a netlist to a certain extent; however, it does not offer a proactive protection mechanism. The initial efforts in logic locking [84, 89, 98, 99], and hardware watermarking [95–97] were broken by Boolean Satisfiability (SAT) attack [100]. The distinguishing input patterns, obtained from SAT solver, combined with their corresponding responses from the oracle, are crucial for SAT attack [100] to uniquely determine the secret key. A DIP with its oracle response is denoted as an input-output (IO) pair, and we will use this terminology throughout the paper. The effectiveness of SAT attack propels

Table 5.1: Summary of post-SAT logic locking techniques and corresponding attacks.

Locking Type	**Techniques**	**Attacks**
Point function	[101–111]	[7, 150–166]
Cyclic	[112–117]	[167–169]
LUT	[118–125]	[125, 170, 171]
Scan	[126–131]	[172–174]
FSM	[132–137]	[175–180]
Timing	[138–143]	[181, 182]
HLS	[144–149]	[162, 163, 181]

the research community for new locking schemes in the post-SAT era, which are summarized in Table 5.1. These include point function-based lockings [101–111], cyclic-based [112–117], LUT/routing-based [118–125], scan and finite state machine (FSM)-based lockings [126–137], timing-based [138–143], and high-level synthesis (HLS)-based [144–149]. Concurrently, multiple attacks [7,125,150–162,162,163,163–181,181,182] against these logic locking techniques arise. In addition, various Machine Learning-based attacks [183–186], which are structural in nature and do not require the oracle, target the identification and recovery of keys that are obfuscated after synthesis process by commercial CAD tools.

The rest of the chapter is organized as follows. We introduce the background of SAT attack and various locking methods in Section 5.1. The inter-dependency between keys learned by SAT solver after each DIP is extensively explored in Section 5.2. The SAT attack complexity is further analyzed and explained in Section 5.3. Analysis of the point functions is shown in Section 5.4. The future directions are described in Section 5.5. Finally, we summarize this chapter in Section 5.6.

5.1 Background

5.1.1 SAT attack on Logic Locking

The entire series of attacks and the solutions thereafter originated from the SAT attack [100]. Subramanyan et al. [100] exploit the idea of combinational equivalence checking with miter

circuit and Boolean Satisfiability [187] to attack logic locking schemes. A miter circuit in SAT attack is the XOR of the output of two copies of the locked circuits sharing identical inputs, but not the secret keys. This oracle-guided attack successfully derives the secret key of various logic locking techniques [84,89,95–99] within a short time frame. The SAT attack requires two circuits, the original circuit, $C_O(X, Y)$, and its locked version, $C(X, K, Y)$, where X, Y, and K are the inputs, outputs, and key, respectively. The correct key K_c restores the original circuit functionality so that its output response is always consistent with the original circuit (e.g., the oracle) under every possible input combination, $C(X, K_c, Y) = C_O(X, Y)$. An incorrect key programmed in the tamper-proof memory leads to output mismatch under one or more input vectors. The output discrepancy between an incorrect key and the correct one is shown on the miter circuit's output. The SAT attack derives the key through the following steps:

Algorithm 1: SAT attack on logic locking [100].

Input : Unlocked circuit, oracle ($C_O(X, Y)$) and locked circuit ($C(X, K, Y)$)
Output: Correct Key (K_c)

```
1  i ← 0;
2  F ← [];
3  while (true) do
4  |   i ← i + 1 ;
5  |   [X_i, K_i, r] = sat[F ∧ (Y_{A_i} ≠ Y_{B_i})];
6  |   if (r == false) then
7  |   |   break;
8  |   end
9  |   Y_i = sim_eval(X_i);
10 |   F ← F ∧ C(X_i, K, Y_i);
11 end
12 K_c ← K_i;
13 return K_c ;
```

• *Finding the distinguishing input pattern (DIP) from the miter circuit:* It first constructs a miter with two copies of the locked circuit A and B. Both circuits ($C(X, K_A, Y_A)$ and $C(X, K_B, Y_B)$ in CNF) share the same input X except for the keys, K_A, K_B. Any output mismatch between the two locked circuits can be easily identified at the miter's output. In each round (*i.e.,* i^{th}), the tool finds the hypothesis key K_i, and reports a Boolean indicator r depending on whether a satisfiable assignment for the miter exists or not, Algorithm 1, Line 5. If SAT is returned, the

miter succeeded in amplifying the mismatched output, r is `true`, and the corresponding input pattern X_i is also recorded.

• *Deriving the correct key:* Upon obtaining a DIP X_i, SAT attack acquires the actual output Y_i from oracle simulation, $C_O(X_i, Y_i)$, Line 9. Input X_i and output response Y_i are used in updating the CNF formula F, Line 10. The clauses in F help narrow down the valid keyspace until it is left with only the correct key(s). If the UNSAT conclusion is generated, the differential output cannot be observed, r is assigned to `false`, and X_i is empty (Lines 6-8) and the program ends. Note that the last iteration of SAT attack returns UNSAT as all incorrect keys are pruned from the keyspace.

The SAT attack repeats the above two steps, where it iteratively checks for satisfiable assignment of the miter circuit. If r is `true` at the i^{th} iteration, we know that incorrect keys still exist in the search space. When the miter circuit becomes UNSAT with the clauses in F, the Boolean variable r becomes `false`, indicating no differential output exists. This means no more incorrect keys can be found as no discrepancy can be produced. If multiple keys remain in the search space, it must be true that multiple solutions are valid since they all give the same output response. This holds for a few locking designs [107, 188] and certain locking scenarios, *e.g.,* chained XOR key gates, where the correct key is not unique. Returning any one of them can restore the original circuit functionality. If only one key is left, it must be the right one. Then, the attack exits the `while` loop, Lines 6-8, and extracts the last round's hypothesis key as the correct one, Line 12. The attack finishes by reporting the correct key to the console, Line 13.

Note that the original SAT attack program [100] includes two preload vectors (all zeros and all ones) at the initial setup, before invoking SAT solver with the miter circuit. The number of IO pairs $|P|$ used to derive the correct key is one more than the number of total iterations TI, $|P| = 2 + (TI - 1) = TI + 1$. This is because the last iteration does not produce a DIP. It is clear that the IO pair count for determining the secret key of a locked circuit is in the same order as the iteration count, which only differs by a constant of 1. For better analyzing the iteration complexity of SAT attack on c6288 benchmark (see Section 5.5.1, we modify the original program [100] by disabling both preload vectors, resulting in $|P| = TI - 1$.

5.1.2 SAT Resistant Logic Locking Techniques and Attacks

As SAT attack [100] successfully breaks various logic locking techniques [84, 89, 95–99], it propels the research community to explore new locking schemes [101–110] that utilize point functions for achieving the minimal output corruptibility. SARLock [101] only perturbs one input pattern's output for each incorrect key. AntiSAT [109,110,188] and CAS-Lock [107] configure the point function with two complementary blocks g and $\overline{g}$. SFLL [105,106,108] flips the output for certain input patterns, where the correct key flips back the upset output and restores the original functionality. Although these techniques guarantee exponential iterations in SAT attack, various attacks [150–164] have been proposed to exploit the designs' vulnerabilities, *e.g.,* from structural and functional perspectives, and restore the original circuit. Nevertheless, SAT attack is still the backbone for the oracle-guided attacks [151–155, 159, 160, 165, 166].

5.2 SAT Attack Analysis: Pruning of Incorrect Key with CNF Update

This section presents a novel perspective of analyzing the SAT attack's effectiveness in breaking various locking schemes in deriving the secret key. We investigate the CNF clauses stored in the SAT solver and how it gets updated in every iteration with a DIP and its output response. The CNF consists of multiple clauses connected with AND ($\wedge$). One or more literals are joined by OR ($\vee$) inside each clause. We use literals, variables, and nodes interchangeably.

The SAT attack requires an unlocked circuit, $C_O(X, Y)$, and its locked version, $C(X, K, Y)$, where X, Y, and K are m, n and $|K|$ bit wide. The correct key K_c restores the original function so that its output response is always consistent with the unlocked circuit for all input combinations, *i.e.,* $C(X, K_c, Y) = C_O(X, Y)$. The SAT solver iteratively finds satisfiable assignments of the miter circuit whose inputs are denoted as DIPs. DIPs and the corresponding oracle outputs are denoted as IO pairs. As logic values for an IO pair $\{X, Y\}$ are known, $C(X, K, Y)$, shown in Figure 5.2(a), is transformed into the functions of keys $C(K_I, K_O)$, shown in Figure 5.2(b), where K_O can be derived from K_I. Further, $C(K_I, K_O)$ can be expanded further and is shown in Figure 5.2(c). Any key in K_O, *e.g.,* K_j^t, is dependent upon key bits K_i^t, *i.e.,* $K_j^t = f(K_i^t)$, where $K_i^t \subseteq K_I$. In addition, the combination of some key bits, *e.g.,* K_i^s,

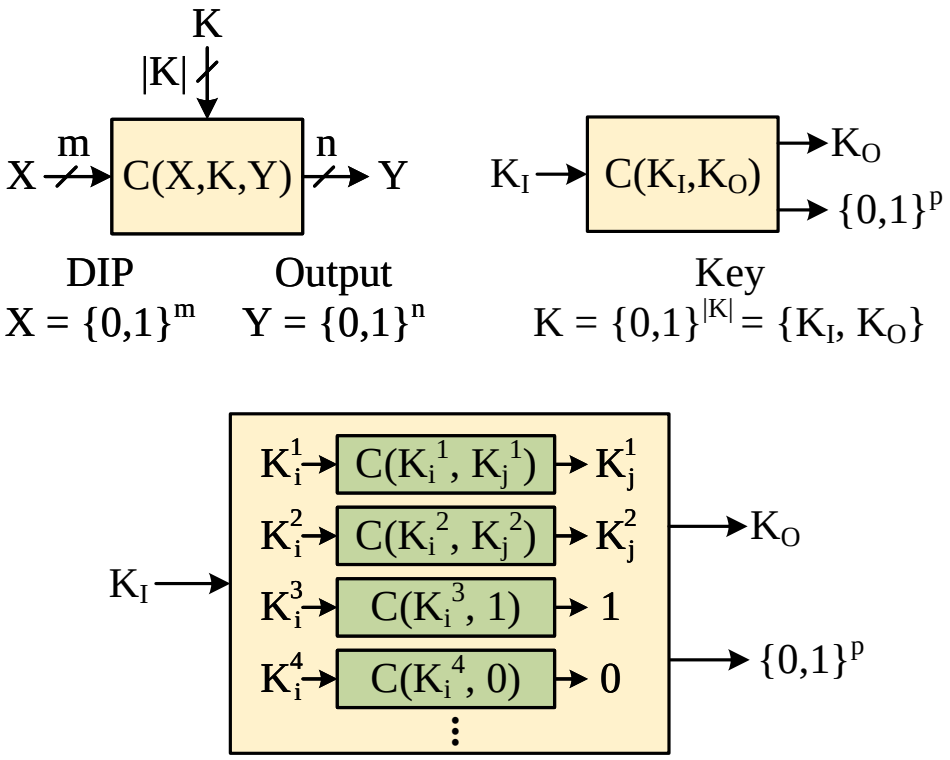

Figure 5.2: Abstract representation of functions of key bits derived from an IO pair. (a) Locked circuit with an IO pair, (b) function of keys, and (c) subfunctions.

$K_i^s \subseteq K_I$, produces a deterministic output, *i.e.,* either logic 0 or 1.

$$C(K_I, K_O) \iff \begin{cases} C(K_i^t, K_j^t), \text{where } K_j^t = f(K_i^t), K_j^t \notin K_i^t; i, j, t = 1, 2, \ldots \\ C(K_i^s, \{0,1\}), \text{where } \{0,1\} = f(K_i^s); i, s = 1, 2, \ldots \end{cases}$$

This key-dependent function $C(K_I, K_O)$ reveals additional information on the interdependency between key bits, *e.g.,* $K_j^t = f(K_i^t)$ and $\{0,1\} = f(K_i^s)$, crucial to the implicit removal of large incorrect key combinations.

The placement of the key gates inside a particular cone is crucial as overlapping cones may reduce the attack complexity. A logic cone can be described as a combinational logic unit that represents a Boolean function bounded by an output and all its inputs. An increased number of primary outputs usually leads to multiple key values propagating across different output bits simultaneously. We begin our analysis with an example circuit with a non-overlapping cone with a single output and show how the SAT attack decrypts the 3-bit key with 3 IO pairs. Then,

we describe how SAT attack can use fewer patterns to determine the secret key when key gates are placed under overlapping logic cones.

5.2.1 SAT Attack for a Locked Cone with One Output

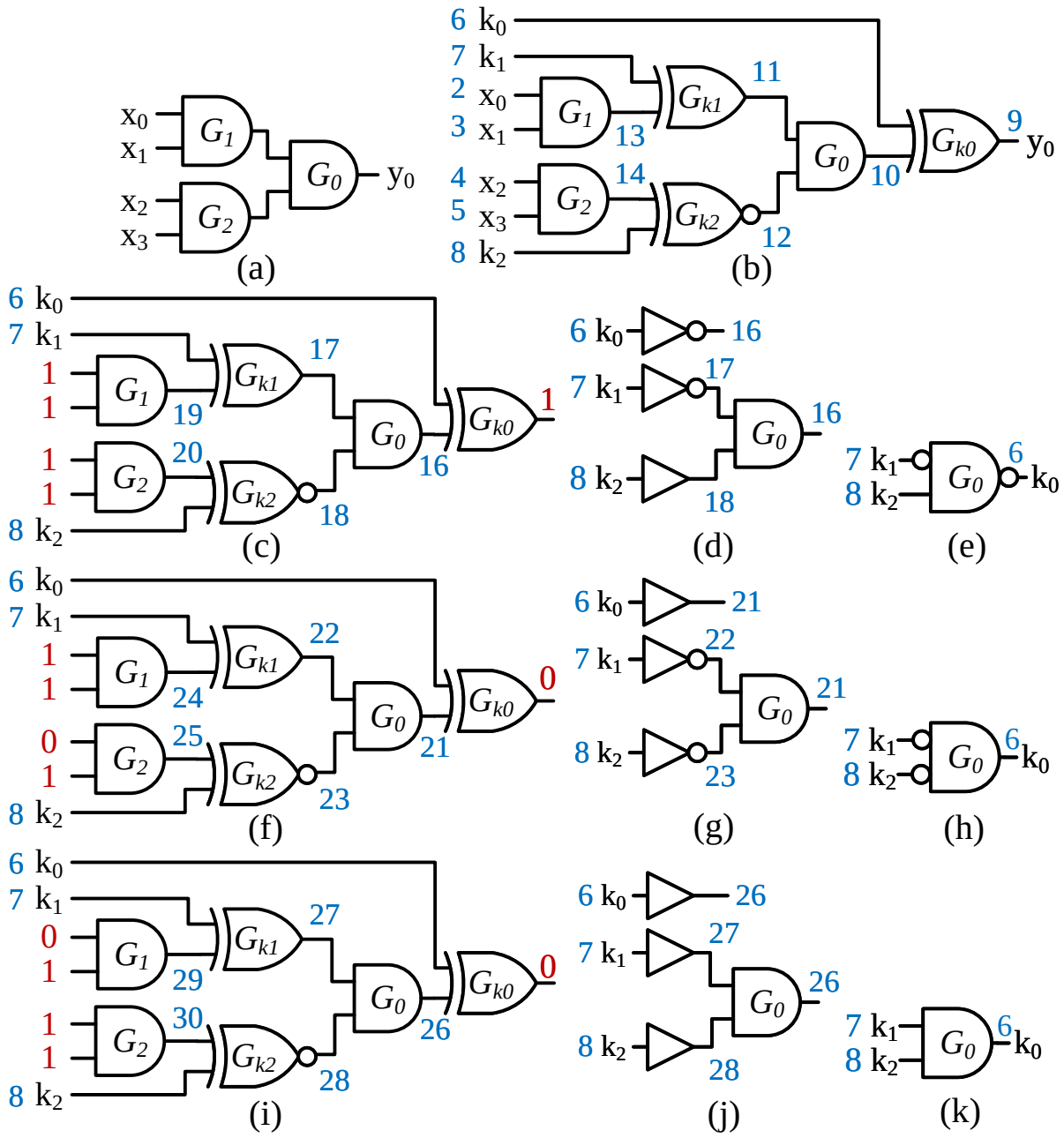

Figure 5.3: Step-by-step SAT attack analysis. (a) Original circuit. (b) Locked circuit with $K = \{001\}$. CNF update and key-pruning for (c-e) 1st IO pair $P_1 = \{1111; 1\}$; (f-h) 2nd IO pair $P_2 = \{1101; 0\}$, and (i-k) 3rd IO pair $P_3 = \{0111; 0\}$.

In this section, we examine how SAT attack implicitly removes the incorrect keys from the entire key search space. As described in Section 5.1.1, SAT solver finds a valid assignment to the miter circuit, and the tool records the extracted input vector, along with its output response obtained from the oracle simulation. The following example shows how SAT attack learns additional information on the secret keys from each IO pair from the miter circuit and oracle simulation.

Let us consider an example circuit with 4 inputs $x_0, ..., x_3$ and 1 output y_0 of Figure 5.3(a). Figure 5.3(b) is the locked circuit with a 3-bit key, k_0, k_1, k_2, using strong logic locking (SLL) scheme. Each node is assigned a unique literal (in blue) by SAT solver. Upon finding a valid assignment to the miter circuit in the first iteration, DIP X_1 is extracted, $\{X_1\} = \{x_0, x_1, ..., x_3\} = \{1111\}$. The output response $Y_1 = 1$ is obtained from oracle simulation with input X_1. SAT attack then records this IO pair $P_1 = \{X_1; Y_1\} = \{x_0, ..., x_3; y_0\} = \{1111; 1\}$. We show in detail how the locked circuit's CNF gets updated under P_1, where the search space is shrunk in half (eliminated 4 incorrect keys). The literal assignment for the locked circuit's original CNF remains unchanged, but the internal nodes for IO pair P_1 are labeled with new variables $\{16\text{-}20\}$ (Figure 5.3(c), consistent with the internal operations of SAT attack [100]. The CNF for $C(X_1, K, Y_1)$ (abbreviated as C_1) is:

$$C_1 = \overbrace{(\overline{17} \vee \overline{18} \vee 16) \wedge (17 \vee \overline{16}) \wedge (18 \vee \overline{16})}^{\text{AND gate } G_0} \wedge$$
$$\overbrace{(\overline{2} \vee \overline{3} \vee 19) \wedge (2 \vee \overline{19}) \wedge (3 \vee \overline{19})}^{\text{AND gate } G_1} \wedge$$
$$\overbrace{(\overline{4} \vee \overline{5} \vee 20) \wedge (4 \vee \overline{20}) \wedge (5 \vee \overline{20})}^{\text{AND gate } G_2} \wedge$$
$$\overbrace{(\overline{6} \vee \overline{16} \vee \overline{9}) \wedge (\overline{6} \vee 16 \vee 9) \wedge (6 \vee \overline{16} \vee 9) \wedge (6 \vee 16 \vee \overline{9})}^{\text{XOR gate } G_{k0}} \wedge$$
$$\overbrace{(\overline{7} \vee \overline{19} \vee \overline{17}) \wedge (\overline{7} \vee 19 \vee 17) \wedge (7 \vee \overline{19} \vee 17) \wedge (7 \vee 19 \vee \overline{17})}^{\text{XOR gate } G_{k1}} \wedge$$
$$\overbrace{(\overline{20} \vee \overline{8} \vee 18) \wedge (\overline{20} \vee 8 \vee \overline{18}) \wedge (20 \vee \overline{8} \vee \overline{18}) \wedge (20 \vee 8 \vee 18)}^{\text{XNOR gate } G_{k2}}$$

With IO pair P_1, we know the logic values for input/output, namely literals $2 = 1, 3 = 1, 4 = 1, 5 = 1, 9 = 1$, and the C_1 is updated as:

$$\begin{aligned} C_1 \;=\; & (\overline{17} \vee \overline{18} \vee 16) \wedge (17 \vee \overline{16}) \wedge (18 \vee \overline{16}) \wedge (19) \wedge (20) \\ & \wedge (\overline{6} \vee \overline{16}) \wedge (6 \vee 16) \wedge (\overline{7} \vee \overline{19} \vee \overline{17}) \wedge (\overline{7} \vee 19 \vee 17) \\ & \wedge (7 \vee \overline{19} \vee 17) \wedge (7 \vee 19 \vee \overline{17}) \wedge (\overline{20} \vee \overline{8} \vee 18) \\ & \wedge (\overline{20} \vee 8 \vee \overline{18}) \wedge (20 \vee \overline{8} \vee \overline{18}) \wedge (20 \vee 8 \vee 18) \end{aligned}$$

Both nodes 19, 20 are in logic 1, and the CNF is adjusted:

$$\begin{aligned} C_1 \;=\; & (\overline{17} \vee \overline{18} \vee 16) \wedge (17 \vee \overline{16}) \wedge (18 \vee \overline{16}) \wedge (\overline{6} \vee \overline{16}) \wedge \\ & (6 \vee 16) \wedge (\overline{7} \vee \overline{17}) \wedge (7 \vee 17) \wedge (\overline{8} \vee 18) \wedge (8 \vee \overline{18}) \end{aligned}$$

This equation reveals that literals 6 and 16 have the opposite logic values, so are 7 and 17, while 8 and 18 are identical. The circuit representation of C_1 is shown in Figure 5.3(d), still a function of k_0, k_1, k_2. These are the clauses appended in the formula F (Algorithm 1 Line 10). Equivalently, what SAT attack learned from the 1st IO pair P_1 is essentially a relation between 3 key bits, where $\overline{k_0} = \texttt{AND}(\overline{k_1}, k_2)$, as in Figure 5.3(e). The constraint shrinks the possible keyspace in half.

On the second iteration, SAT attack returns the 2nd IO pair $P_2 = \{X_2; Y_2\} = \{1101; 0\}$, as in Figure 5.3(d). Using the derivation we performed for the first iteration, the circuit representation of the added CNF clauses is shown in Figure 5.3(g), which again is a function between the 3 key bits, $k_0 = \texttt{AND}(\overline{k_1}, \overline{k_2})$ (Figure 5.3(h)). It further shrinks the remaining keyspace in half, with only two keys left valid. Figure 5.3(i) shows the 3^{rd} IO pair $P_3 = \{0111; 0\}$, whose CNF $C(X_3, K, Y_3)$ and its equivalent relation $k_0 = \texttt{AND}(k_1, k_2)$ are illustrated in Figure 5.3(j), (k), respectively. The combined effect of these three IO pairs, P_1, P_2, P_3, Figure 5.3(e, h, k), uniquely determine key $K = \{k_0, k_1, k_2\} = \{001\}$. On the 4th iteration, no more distinguishing input can be found for the miter circuit, where r is `false`, and the SAT attack is complete.

In short, each IO pair provides additional information on the unknown key bits, where $C(X_i, K, Y_i)$ essentially becomes an equation for the unknown keys. A new equation for key is obtained in every iteration from the corresponding IO pair, which is independent of the findings derived from the previous rounds. SAT attack derives the secret key once the accumulated system of equations can uniquely determine all key bits.

5.2.2 SAT Attack against Multiple Overlapping Logic Cones

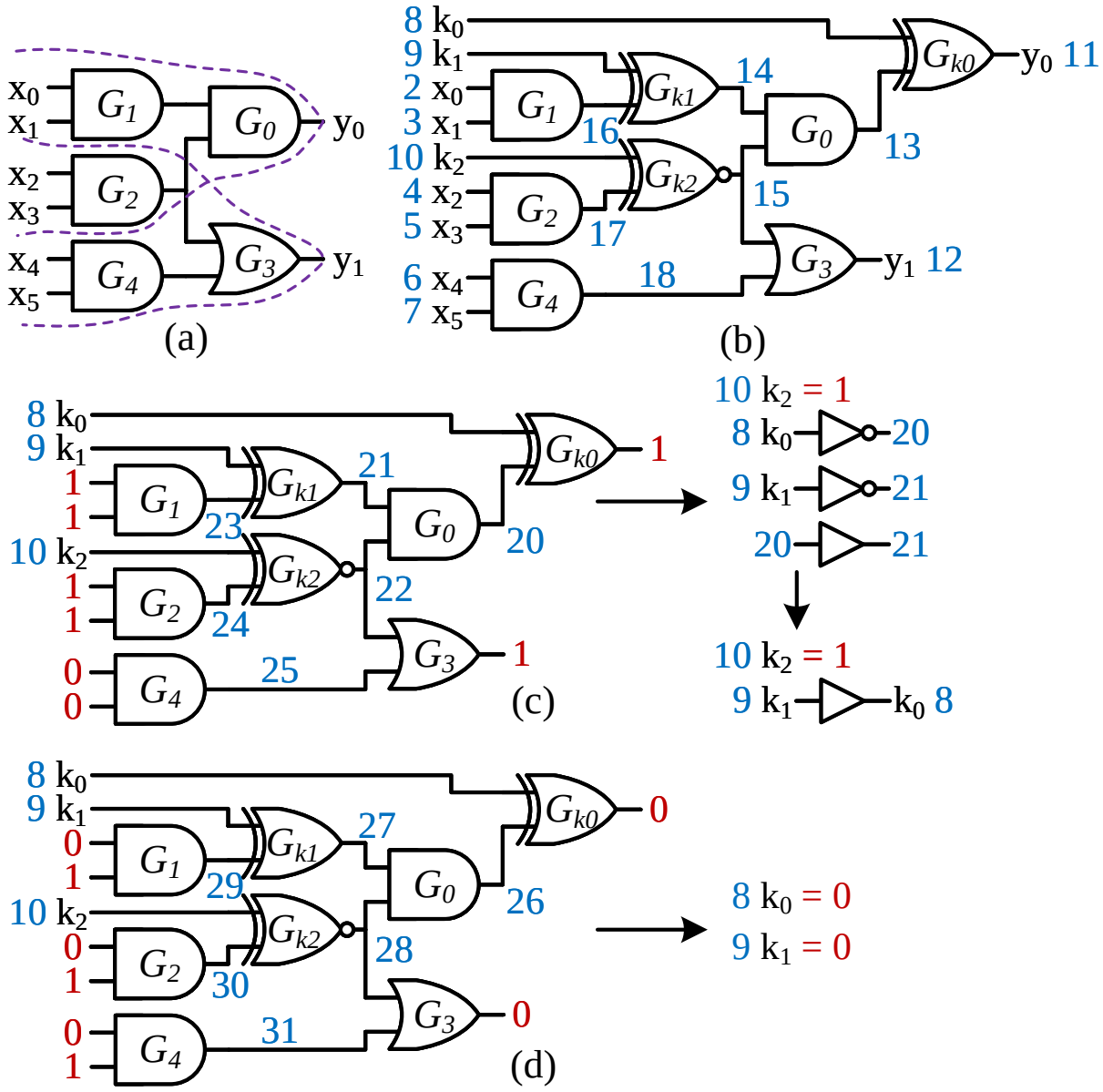

Figure 5.4: SAT attack on 2 intersecting cones with $K = \{001\}$. (a) Original circuit. (b) Locked circuit. CNF update and key-pruning for (c) 1st pair $P_1 = \{111100; 11\}$, (d) 2nd IO pair $P_2 = \{010101; 00\}$.

It is common for a circuit to have multiple outputs or fanouts. In other words, that circuit has multiple logic cones. With more fanouts, incorrect key responses are more likely to be observed than a single output. As the logic values for multiple keys can reach several outputs simultaneously, it accelerates and facilitates the removal of incorrect combinations to get the final key than the single logic cone where every key has to be observed from the same output pin. This is demonstrated by the example below. The following example shows that SAT attack needs fewer iterations to derive the secret key under multiple intersecting logic cones.

Let us consider a circuit with 2 outputs, y_0 and y_1, as shown in Figure 5.4(a). The locked circuit, as shown in Figure 5.4(b), has 3 key bits, k_0, k_1, k_2, with the same locations as in Figure 5.3(b). It differs from the locked circuit in Figure 5.3(b) with additional gates G_3, G_4, and output y_1. This circuit has two logic cones; one with output y_0, inputs x_0, x_1, x_2, x_3, keys k_0, k_1, k_2, and gates $G_0, G_1, G_2, G_{k0}, G_{k1}, G_{k2}$; the other with output y_1, inputs x_2, x_3, x_4, x_5, key k_2, and gates G_2, G_3, G_4, G_{k2}. The effect of k_2 can be observed from both outputs, y_0 and y_1. SAT attack only needs 2 IO pairs to solve the keys, as opposed to 3 IO observations for the locked cone with a single output y_0 in Figure 5.3(b). Figure 5.4(c) illustrates the 1st IO pair $P_1 = \{X_1; Y_1\} = \{x_0, ..., x_5; y_0, y_1\} = \{011001; 00\}$, Its equivalent CNF expression of $C(X_1, K, Y_1)$ (abbreviated as C_1) is expressed in:

$$
\begin{aligned}
C_1 = {} & (\overline{21} \vee \overline{22} \vee 20) \wedge (21 \vee \overline{20}) \wedge (22 \vee \overline{20}) \wedge (\overline{2} \vee \overline{3} \vee 23) \\
& \wedge (2 \vee \overline{23}) \wedge (3 \vee \overline{23}) \wedge (\overline{4} \vee \overline{5} \vee 24) \wedge (4 \vee \overline{24}) \wedge (5 \vee \overline{24}) \\
& \wedge (22 \vee 25 \vee \overline{12}) \wedge (\overline{22} \vee 12) \wedge (\overline{25} \vee 12) \wedge (\overline{6} \vee \overline{7} \vee 25) \\
& \wedge (6 \vee \overline{25}) \wedge (7 \vee \overline{25}) \wedge (\overline{8} \vee \overline{20} \vee \overline{11}) \wedge (\overline{8} \vee 20 \vee 11) \\
& \wedge (8 \vee \overline{20} \vee 11) \wedge (8 \vee 20 \vee \overline{11}) \wedge (\overline{9} \vee \overline{23} \vee \overline{21}) \\
& \wedge (\overline{9} \vee 23 \vee 21) \wedge (9 \vee \overline{23} \vee 21) \wedge (9 \vee 23 \vee \overline{21}) \\
& \wedge (\overline{10} \vee \overline{24} \vee 22) \wedge (\overline{10} \vee 24 \vee \overline{22}) \wedge (10 \vee \overline{24} \vee \overline{22}) \\
& \wedge (10 \vee 24 \vee 22)
\end{aligned}
$$

The 1st IO pair P_1 gives $2 = 1, 3 = 1, 4 = 1, 5 = 1, 6 = 0, 7 = 0, 15 = 1, 17 = 1$. The CNF for the locked circuit with P_1 is adjusted analogously to the previous example (Figure 5.3) by plugging in the logic value of these known literals. It is straightforward that node 23, the output of AND gate G_1, has logic 1, as its inputs are literals $2 = 1$ (x_0) and $3 = 1$ (x_1). Similarly, nodes $24 = 1$, and $25 = 0$, based on literals $4 - 7$ ($x_2, ..., x_5$). With an output 1 for OR gate G_5, its remaining input of node 22 must be 1, as $25 = 0$. Therefore, the CNF clauses added to SAT solver after the first iteration is:

$$C_1{=}(\overline{21}\vee 20)\wedge(21\vee\overline{20})\wedge(\overline{8}\vee\overline{20})\wedge(8\vee 20)\wedge(\overline{9}\vee\overline{21})\wedge(9\vee 21)\wedge(10)$$

With C_1, SAT attack determines key bit $k_2 = 1$, along with key-dependent equation $k_0 = k_1$, as shown in Figure 5.4(c). With the 2nd IO pair $P_2 = \{X_2; Y_2\} = \{010101; 00\}$, as in Figure 5.4(d), SAT atttack uniquely determines both key bits k_0 and k_1 as logic 0. In the third round, SAT attack returns UNSAT (as all key bits are solved), and the program finishes.

Table 5.2: Comparison of SAT attack iterations (TI) between multiple primary outputs ($|PO|$) and single cone.

Benchmark	**Locked Circuit (SLL) [100]**			**Locked Cone (Sec. 5.3)**		
	$\boldsymbol{\|PO\|}$	$\boldsymbol{\|K\|}$	$\boldsymbol{TI}$	$\boldsymbol{\|PO\|}$	$\boldsymbol{\|K\|}$	$\boldsymbol{TI}$
c432	7	80	24	1	21	27
c880	32	192	76	1	50	80
c1355	32	137	29	1	17	33
c1908	25	220	110	1	92	123
c3540	22	167	40	1	58	40
c5315	123	231	55	1	78	60

In addition to the example described above, we perform experiments to show a weaker attack resiliency for overlapping cones, as summarized in Table 5.2 with locked ISCAS'85 benchmarks. Table 5.2 compares the attack complexity with key sizes between the complete benchmark circuit, where multiple overlapping cones exist, and the extracted single cone from the same benchmark. Columns 2-4 and 5-7 list the number of primary outputs ($|PO|$), key size ($|K|$), and the total SAT attack iterations (TI) for breaking the SLL-based locked benchmarks and the corresponding largest cone, respectively. For example, SAT attack takes 76 iterations

to determine a 192-bit key for the c880 benchmark, whereas it takes 80 iterations to break the largest cone of c880 locked with a merely 50-bit key. We observe the same behavior for all other benchmarks as well. The SAT attack can only break fewer keys (or smaller key size) for a single-output logic cone than for the keys of the same benchmark circuit having multiple PO. This confirms a lower complexity for overlapping cones, which is due to the effect of incorrect keys manifested through multiple outputs where the interdependency between key bits is broken. Therefore, having multiple overlapping logic cones will reduce the iteration counts for SAT attack, making it easier to derive the final key when key bits can be observed at the outputs simultaneously. Since we are examining and analyzing the effectiveness of SAT attack, we henceforth focus on the analysis with a single logic cone only, as it is more complex than multiple cones and offers an upper bound to the iteration complexity. If we show the linear iteration complexity for a non-overlapping cone, then automatically, the same linear complexity will be preserved for overlapping cones.

5.3 SAT Attack Analysis: Iteration Complexity

In this section, we focus on the total iterations required for the SAT attack as the SAT attack complexity. We observe the linear iteration complexity for all ISCAS'85 benchmarks that agrees with the previously reported results. We, however, also observe the decrease in iteration complexity with increased key sizes for a large number of cases. To explain this phenomenon, we analyze how the output response from oracle, under certain DIPs, can trim more incorrect keys than other IO pairs. The complexity drop is caused by the multiple effective IO pairs selected by the tool. This explanation can also clarify the local peaks in iteration complexity due to the SAT attack selecting multiple less-effective IO pairs. We focus on the complexity trend for the iteratively increase in key sizes for any XOR-based locked circuits. The iterative insertion of keys (and key gates) ensures that the addition of one more key bit does not alter the locations of the already inserted key bits (and key gates). *To the best of our knowledge, this is the first study to report the reduction of iteration count with increased key size.*

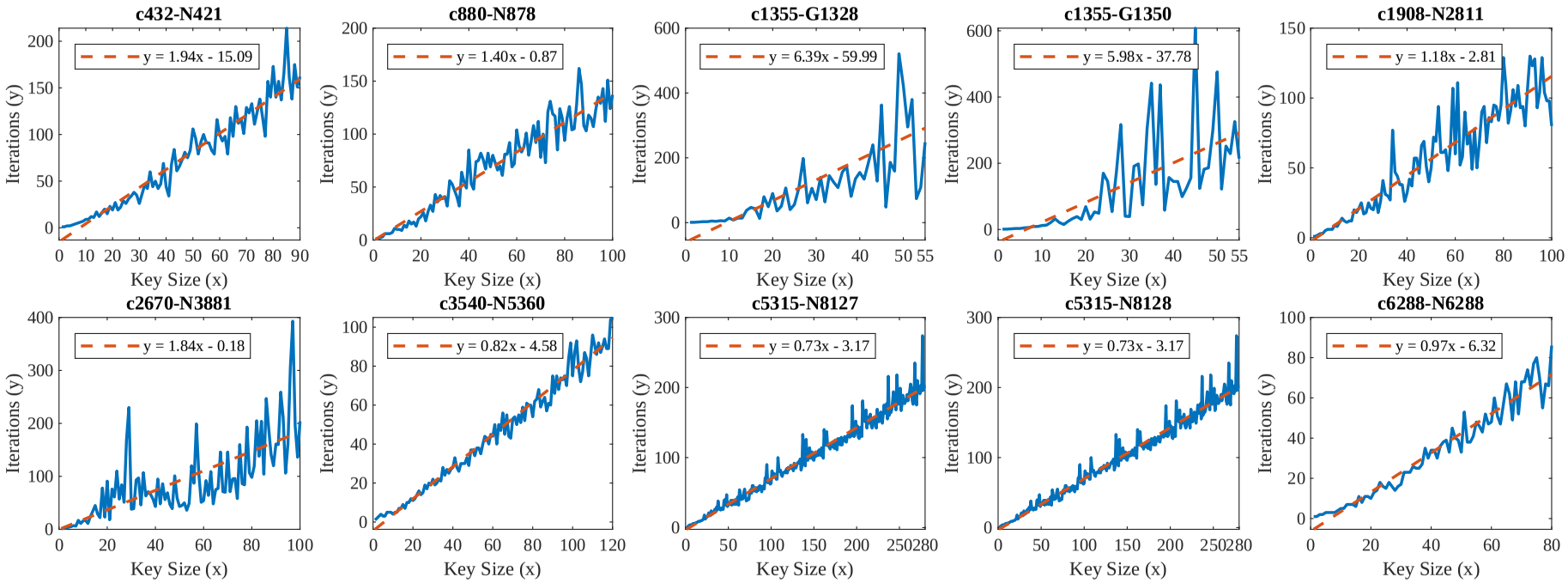

Figure 5.5: SAT attack total iterations for ISCAS'85 benchmarks.

The overview of SAT attack complexity analysis on the same logic cone is summarized in the following steps: (i) benchmark synthesis, (ii) cone analysis and the largest cone extraction, (iii) iterative insertion of key bits, and (iv) SAT attack iteration complexity aggregation. Synthesis is performed under 32nm technology libraries in Synopsys Design Compiler [189]. Figure 5.5 shows an overall linear trend in total attack iterations under increased key sizes for non-overlapping cones. The best-fit lines are drawn in dashed lines with equations. To avoid the complexity reduction under multiple logic cones, the largest cone from each synthesized ISCAS'85 benchmark is extracted so that the response of any incorrect key combinations is observed through the sole output only. Each circuit is mapped to a directed graph with inputs pointing toward gates' output and, ultimately, the primary output. Logic cones are extracted by reversal of edge directions [190] and breadth-search [191] from each primary output. The ordered node list obtained in breadth-first search is used for determining (i) the largest cone (or cones if a tie) by node count and (ii) key gate insertion sequence as breadth-first search traverses all gates (nodes) within the same layer (same distance from output nodes) first before reaching gates at further layers. Following the same node order as in breadth-first search, we successively add one more XOR/XNOR key gate at a time, starting from gates closest to the primary output with increasing proximity. The original cone and its locked designs are all converted to the *bench* format. SAT attack runs through all key sizes for every locked cone, and the total iterations are recorded. Figure 5.5 shows the SAT attack iteration complexity on 9 benchmark cones with increasing key sizes. For example, c432-N421 is the logic cone from c432 benchmark with output N421. Cone c5315-N8127 and c5315-N8128 both contain the same gate count, but a significant overlap of gates exists. Please note that these logic cones all have reconvergent fanouts [192].

There are two observations from Figure 5.5. First, the overall complexity increase is not exponential, but linear. This means that, on average, the attack removes an exponential (or sub-exponential) number of incorrect keys per iteration. Second, all 9 benchmark cones exhibit the local non-monotonically complexity increase when additional keys are inserted. Note that a monotonic function (f) is either an entirely nonincreasing or nondecreasing function, where its first derivative does not change sign [193]. Now, f is called monotonically increasing if $\forall x, y$,

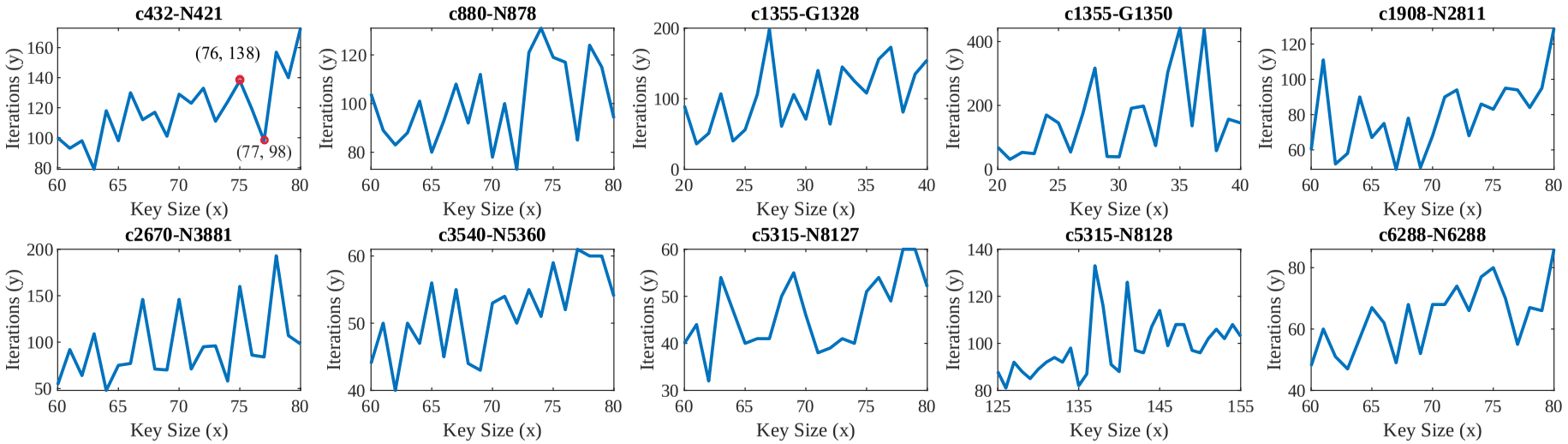

Figure 5.6: The zoomed-in view of SAT attack Iterations for ISCAS'85 benchmarks.

it satisfies $f(x) \leq f(y)$ for $x \leq y$. We denote a function as non-monotonically increasing if it increases globally (on average), but not monotonic.

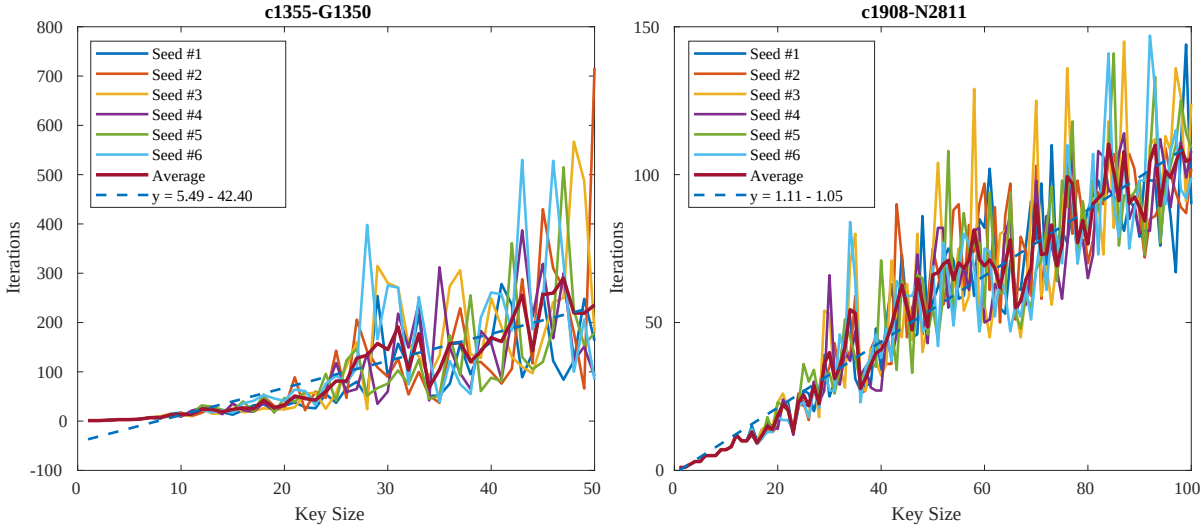

Figure 5.7: The iteration complexity of SAT attack on locked benchmarks c1355-G1350 and c1908-N2811 with 6 randomized seed setups for SAT solver Lingeling.

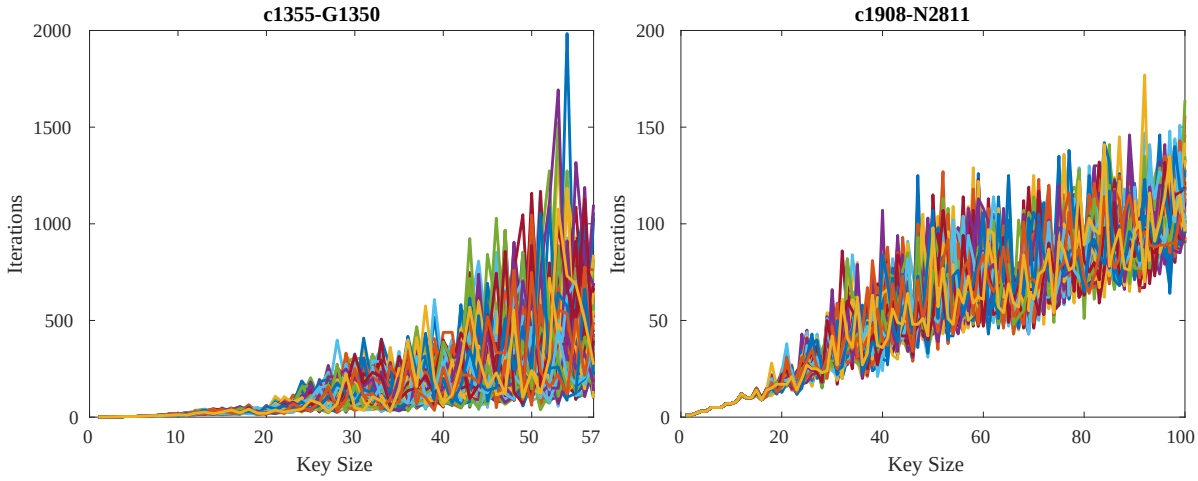

Figure 5.8: The iteration complexity of SAT attack on locked benchmarks c1355-G1350 and c1908-N2811 with 100 randomized seed setups for SAT solver Lingeling.

Figure 5.6 shows the zoomed-in view of SAT attack complexity for benchmark cones. For example, for cone c432-N421, it takes 138 iterations to break the key size of 76, but only needs 98 iterations when one more key bit is added. A non-monotonically increase in complexity is also observed in all the other benchmark cones. Note that the same non-monotonical behavior for the iteration complexity can still be observed under a different initialization seed setup.

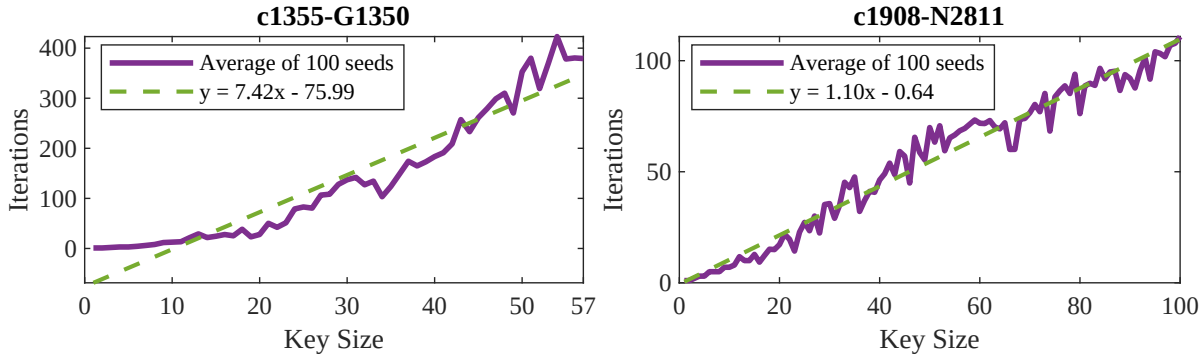

Figure 5.9: The average iteration complexity of SAT attack on locked benchmarks c1355-G1350 and c1908-N2811 with 100 different seed setups for SAT solver Lingeling.

In addition, a non-monotonic linear increase can be observed in the averaged linear iteration complexity with as little as 6 randomized seeds as well as 100 different initialization seeds, as shown in Figures 5.7, 5.8, and 5.9 for c1355-G1350 and c1908-N2811. However, it does not suggest or infer that the minimum DIP count for solving each locked circuit with increased key sizes, a problem spans in PSPACE complexity instead. It is evident that the insertion of more key gates does not always lead to an increase in attack complexity. The non-monotonically increasing behavior in iteration complexity is observed in all cones. The question is, *what causes the SAT attack to have such complexity drops when more keys are present in a locked design?*

To describe the non-monotonically increasing behavior, we consider another example shown in Figure 5.10 where the effectiveness of individual IO pairs in eliminating incorrect keys is explored. The purpose here is to demonstrate that all the IO pairs are not equally effective in eliminating incorrect keys, some are better than others. As the SAT tool finds a DIP, which typically depends on the circuit topology, it is possible that the tool selects a more efficient DIP in earlier iterations for a locked circuit with a larger key that eliminates a large number of incorrect keys which results in a reduction in iteration. Figure 5.10(a) shows the circuit, where four keys ($k_0 - k_3$) are added (Figure 5.10(b)). Note that an OR gate (G_7) is located at the cone output. The 1^{st} IO pair $P_1 = \{X_1; Y_1\} = \{x_0, ..., x_6; y\} = \{0000000;1\}$ returned by SAT attack has $y = 1$ as the output. As logic 1 is the output of G_7, its two inputs could be any of the 3 combinations $\{01/10/11\}$. As the correct key cannot be determined uniquely, we focus on finding the incorrect ones, which are unique and result from $\{00\}$. One

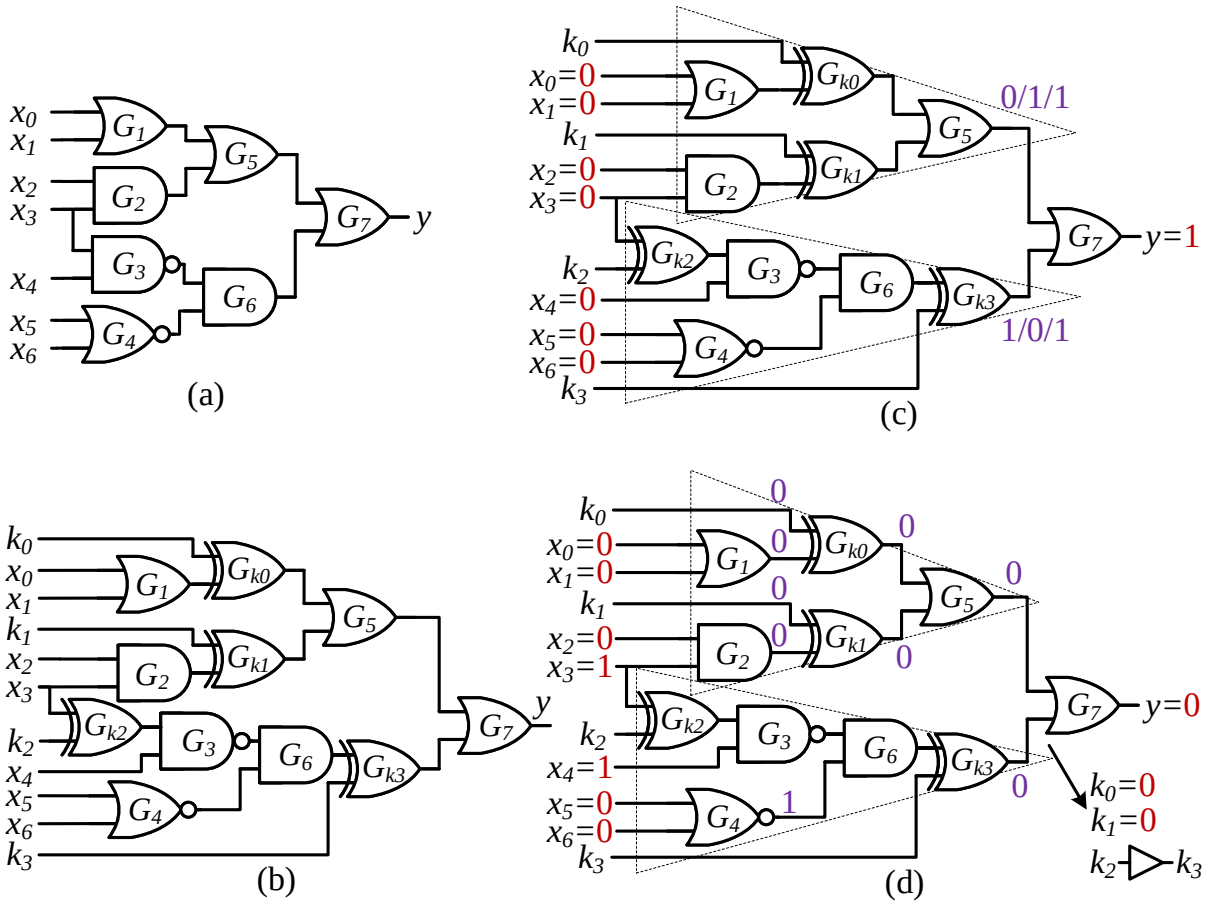

Figure 5.10: Key elimination. (a) original circuit, (b) locked circuit with 4 keys, (c) the 1^{st} IO pair $P_1 = \{0000000;1\}$ from SAT attack (d) the second IO pair $P_2 = \{0001100;0\}$ and equivalent relation of k_0, k_1, k_2, k_3 under P_2 only, where k_0 and k_1 are determined.

can simply find these incorrect ones using logic propagation, and it can be shown that there exist only 2 incorrect keys, Table 5.3 Column 2. Any key combinations that cause an output mismatch with the oracle's are marked with ✗, indicating an incorrect key value implicitly removed from keyspace; key value(s) which produces the same output as the oracle's is noted with ✓. From the 1st iteration, we observe fewer incorrect keys (i.e., $2 \ll \frac{2^4}{2}$) are removed than the 2nd iteration (i.e., $14 \gg \frac{2^4}{2}$) due to the properties of OR gate, where no unique conclusion can be made regarding its inputs (i.e., 10, 01 or 11) if the output is 1.

On the second iteration, the tool obtains another IO pair, $P_2 = \{0001100;0\}$, with 0 at the output of the OR gate. The rest 13 incorrect key combinations are identified from P_2, as listed in Table 5.3, Column 3. Here, we are interested in how P_2 trims more than half of the keys in the search space. The locked circuit with the IO pair P_2 is shown in Figure 5.10(d). With the same derivation for CNF $C(X_2, K, Y_2)$, we know the outputs of gates G_1, G_2, G_4 are 0, 0, 1,

Table 5.3: SAT attack uses 2 patterns to eliminate all 15 incorrect keys from keypace. If output differs from the oracle's, ✗ is placed, else ✓. The correct key is highlighted.

4-bit key $\{k_0, ..., k_3\}$	**IO Pair 1** (P_1) {0000000;1}	**IO Pair 2** (P_2) {0001100;0}
0000	✓	✓
0001	✗	✗
0010	✓	✗
0011	✗	✓
0100	✓	✗
0101	✓	✗
0110	✓	✗
0111	✓	✗
1000	✓	✗
1001	✓	✗
1010	✓	✗
1011	✓	✗
1100	✓	✗
1101	✓	✗
1110	✓	✗
1111	✓	✗

once we have the input X_2. These gates' outputs can be similarly decided with X_1. With output $y = 0$ at OR gate G_7, both inputs from this OR gate must be 0. This means both outputs of OR gate G_5 and XOR gate G_{k3} are 0; and subsequently, the inputs of OR gate G_5 must be 0 as well, which are the output of both XOR gates G_{k0} and G_{k1}. This results in the unique solution for 2 key bits k_0 and k_1 with $k_0 = 0, k_1 = 0$.

A similar analysis can be performed on AND gates, whose inputs are uniquely defined under a logic 1 output. In summary, having a response of 0 at OR gates, or 1 at AND gates effectively splits the cone into two halves, where keys in one half are independent of the keys in the other half. This is equivalent to splitting the logic cone into two subcones based on input ports of OR/AND gates, where keys in both subcones can be evaluated and trimmed simultaneously. Therefore, the efficiency of removing incorrect keys depends on IO pairs, where the selection of an IO pair depends on the locked circuit topology that changes when adding more key bits. The effective IO pairs help remove more incorrect keys than the others. If a few effective IO pairs are selected in earlier iterations of the SAT attack, the iteration count

can go down significantly. This leads to a non-monotonically increasing iteration complexity with the key size.

5.4 Case Study: Locking with Point Functions

As the efficiency of SAT attack is indisputable, the subsequent logic locking proposals shift the focus toward building an exponential complexity in total iterations against SAT attack. One of the common approaches is to embed a point function right before the output of a logic cone, where the circuit's output response is perturbed based on the designer's chosen input combinations. This section presents a theoretical analysis of point functions of AntiSAT [188], CAS-Lock [107], TTLock [104], and SFLL [105], and explains why they can also be broken by SAT-based attacks. In this section, we present a case study on how our proposed SAT attack analysis (see Sections III, IV) can be used to analyze the attack complexity of KBM & SAT [160], a modified version of SAT attack with key constraints. The following analysis clarifies how and why SAT attack can still be effective in breaking AntiSAT and CAS-Lock under proper key constraints. Note that we are not proposing any new attacks but rather providing explanations to demonstrate that (i) AntiSAT with fixed K_g requires only a single IO pair to determine the secret key, and (ii) the linear complexity for CAS-Lock under the same constraint. We also show that adding additional key constraints on $K_{\overline{g}}$ would not yield any extra benefits on the complexity reduction to an adversary for breaking AntiSAT-based locking designs. Note that the adversarial model for logic locking follows the same Kerckhoffs's principle as in modern cryptography, where the security of the system is based on the secret key and not on the obscurity of the algorithm used [161, 184, 186]. For point function-based locking techniques such as AntiSAT and CAS-Lock, we assume that the attacker has full knowledge of the locking scheme and the existence of a comparator logic inside the locked netlist.

5.4.1 Deterministic Property of SAT Attack with Constraints

This section analyzes the SAT attack complexity on point function-based locking schemes with complementary blocks, g and $\overline{g}$, where K_g and $K_{\overline{g}}$ are inside g and $\overline{g}$, respectively. Sengupta et al. [160] have shown an effective approach to reducing AntiSAT and CAS-Lock to polynomial

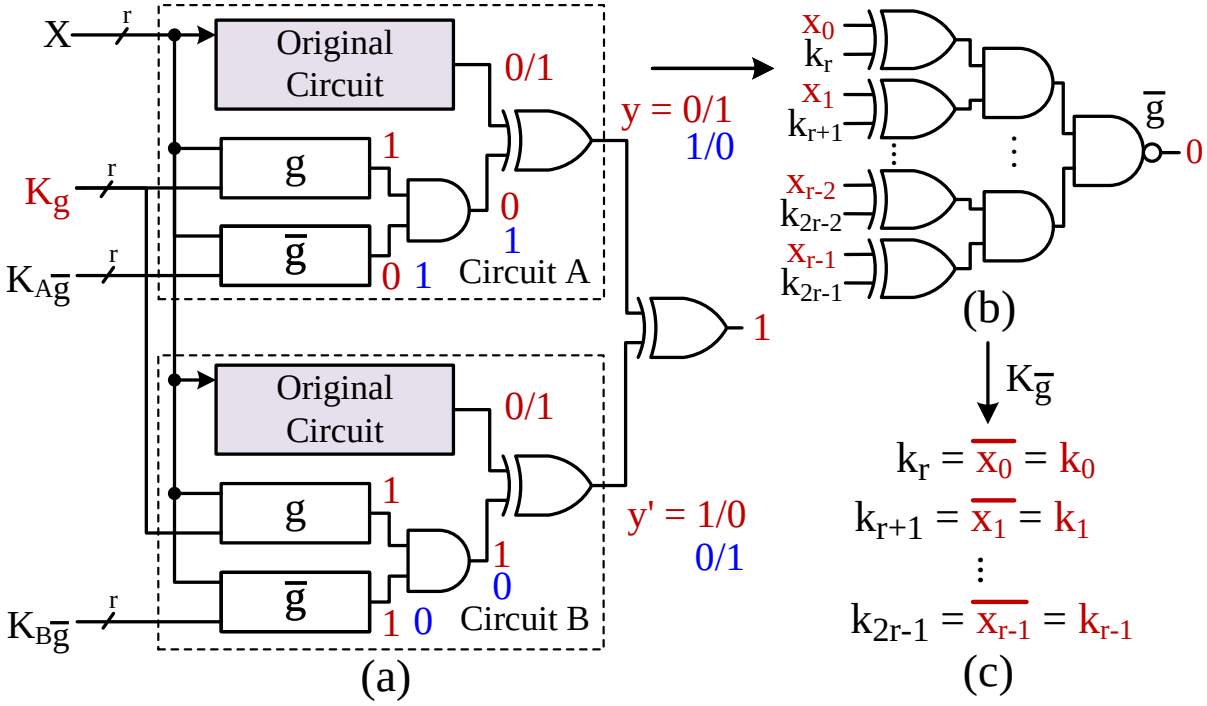

Figure 5.11: SAT attack on AntiSAT with fixed key K_g. (a) Miter construction. (b) CNF update. (c)$K_{\bar{g}} = \{k_r - k_{2r-1}\}$ is determined.

complexity with key-bit mapping (KBM) & SAT, where KBM separates K_g and $K_{\bar{g}}$ and SAT attack is applied with a fixed K_g. The following analysis explains how and why SAT attack is still effective in breaking AntiSAT and CAS-Lock under proper key constraints. Our proposed SAT attack analysis also explains the same attack complexity of linear in iterations. In addition, we provide explanations to show (i) AntiSAT with fixed K_g requires only one IO pair to determine the secret key, and (ii) linear complexity for CAS-Lock under the same constraint. However, constraining $K_{\bar{g}}$ would not give any extra benefits on the complexity reduction to an adversary on breaking AntiSAT.

SAT attack analysis on AntiSAT under key constraints

The two sets of keys, K_g and $K_{\bar{g}}$, in AntiSAT offer two choices for the attacker, fixing one or the other. Using our key pruning analysis of Section 5.2, we explain how an adversary can determine the key with single IO pair only when setting K_g constant. Yet, he/she will be less fortunate in breaking the secret key if $K_{\bar{g}}$ is kept constant instead.

• Key constraint on K_g: Let us consider a circuit with r-bit input $X = \{x_0, ..., x_{r-1}\}$, 1-bit output y, locked with $2r$-bit keys of $K_g = \{k_0, ..., k_{r-1}\}$ and $K_{\overline{g}} = \{k_r, ..., k_{2r-1}\}$ of r-bit each. We assume the attacker already knows the bit locations for K_g using the KBM of [160]. Furthermore, the r-bit $K_g = \{k_0, ..., k_{r-1}\}$ is set to a constant vector. SAT attack is able to find an IO pair and uniquely determines all bits of $K_{\overline{g}}$. Figure 5.11(a) shows the miter construction, where K_g are highlighted in red to indicate a fixed value. As miter creates differential output between two copies of the locked circuit A and B, without loss of generality, suppose the point function of circuit A has output 0 and circuit B's has output 1, shown in red (and vice versa in blue). Since both circuits have the same original cone, their output is identical to both A and B as they share the same input X. Without loss of generality, we assume the output of the original cone under the DIP found by the miter is logic 0. One can also assume with logic 1 instead. Hence, the output of AntiSAT block in circuit A is 0 while 1 for B's. As AntiSAT block has AND at the output, both inputs of this AND gate in B are 1, where $g = 1$, $\overline{g} = 1$. Then, we know that the DIP X obtained from the miter must be complementary to the fixed K_g, $X = \overline{K_g}$ to ensure all ones for g's AND tree of B. Following the analysis in Section 5.2, the solver updates its CNF clauses with X and the oracle's output (logic 0 from the assumption). When this IO pair is applied to the locked circuit, the AntiSAT block gets a logic 0 output. Since DIP X gives $g = 1$ for circuit B, we still have $g = 1$ during CNF update. Then, $\overline{g} = 0$, as shown in Figure 5.11(b). As $\overline{g}$ is the NAND gate's output, all its inputs have logic 1. This uniquely determines all r-bit key $K_{\overline{g}}$, which is the complement of DIP X, $K_{\overline{g}} = X$, and identical to K_g, $K_{\overline{g}} = X = K_g$. SAT attack completes on the 2[nd] iteration since key $K_{\overline{g}}$ is already resolved. Therefore, the constraint on K_g helps SAT attack finish within one IO pair.

• Key constraint on $K_{\overline{g}}$: If the adversary decides to set key $K_{\overline{g}}$ constant instead, he/she will not get the same efficiency for key derivation as in fixing K_g. Suppose we constrain $K_{\overline{g}} = \{k_r, k_{r+1}, ..., k_{2r-1}\}$ to a constant r-bit vector. When SAT solver tries to find a satisfiable assignment to the miter circuit, following the same assumptions as before, we can derive that both A and B have the same logic 1 for $\overline{g}$ blocks. Having an output 1 at the NAND gate is equivalent to putting logic 0 to an AND gate. There are $2^r - 1$ possible solutions for the r-bit input to produce a logic 1 at $\overline{g}$'s output. Equivalently, there are $2^r - 1$ choices of DIP, satisfying the

criterion of miter construction. When the tool updates SAT solver's CNF with DIP and output response, we get $\overline{g} = 1$ for the NAND gate and $g = 0$ for AND gate. Since unknown key bits are in K_g of block g, a specific IO pair can prune only 1 incorrect key combination that results in $g = 1 (\neq 0)$. The total IO pairs required to remove all incorrect keys of the r-bit keyspace for K_g is $2^r - 1$. The total iterations required for SAT attack is 2^r. Therefore, by constraining $K_{\overline{g}}$, the adversary removes only one incorrect key and the overall SAT attack complexity remains exponential.

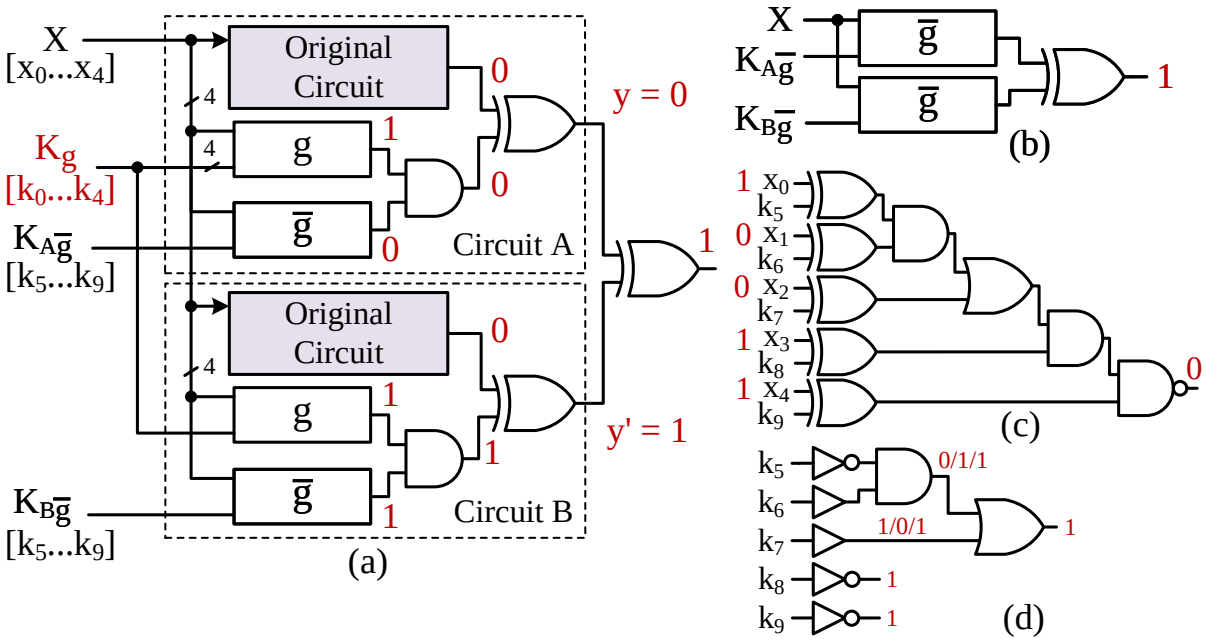

Figure 5.12: SAT attack on CAS-Lock with fixed K_g. (a) Miter construction and (b) Equivalent representation. (c) CNF update at SAT attack iteration 1. (d) Key pruning after iteration 1.

SAT attack analysis on CAS-Lock under key constraints

The same analysis on key constraints in AntiSAT can be applied to CAS-Lock, where the constraining of K_g or $K_{\overline{g}}$ leads to linear complexity in solving $K_{\overline{g}}$ or K_g, respectively. We illustrate with a $(2r = 10)$-bit CAS-Lock example, where block g and $\overline{g}$ have one OR gate each, as shown in Figure 5.12. Our analysis can be generalized and applied to any OR gate replacement inside the cascaded AND chain of g and $\overline{g}$. When SAT attack searches for a DIP X for miter, as shown in Figure 5.12(a), the CAS-Lock block of one copy (*i.e.,* A) has logic 0 while the other (*i.e.,* B) has logic 1. As K_g is fixed, the miter is essentially solving a differential

output for $K_{\overline{g}}$ (Figure 5.12(b)). Suppose B's CAS-Lock block is 1, then it has $g = 1$ and $\overline{g} = 1$. The DIP X obtained by SAT solver must satisfy $g = 1$ as K_g is constant. The oracle response, identical to the analysis for AntiSAT, helps to determine a logic 0 for the CAS-Lock block, as no alteration of output logic occurred. The CNF update implicitly eliminates the wrong keys in $\overline{g}$ with $\overline{g} = 0$ under $g = 1$ and output 0 for the combined blocks. After the 1st iteration, k_8, k_9 are uniquely determined, which reduces the key space from 2^5 to 2^3 and is shown in Figure 5.12(d). Note that some of the keys k_5, k_6, k_7 will be determined in the same way in the 2nd iteration of the SAT attack. The attack will continue iterating until all key bits are uniquely determined.

5.4.2 Extending the point function analysis to TTLock and SFLL

TTLock [104] and SFLL [105, 106] do not have two sets of keys like AntiSAT and CAS-Lock. Both perturb unit (*PU*) and restore unit (*RU*) are serially XORed with the original circuit, e.g., a logic cone (*LC*) of interest, as shown in Figure 5.13. Keys are in the restore unit (*RU*) only, where the perturb function (F^*) is key-free. The same analysis can be performed as PU and RU with the correct key implementing the same function even though different versions of SFLL have different output corruptibility. The output of PU, F^*, is logic 1 for only one input combination, where it alters the circuit behavior. The correct key helps flip back the perturbed logic and restores the original functionality as LC. Therefore, it must be true that the functional behavior for PU and RU are identical under the correct key so that LC's output is preserved. In other words, PU is the oracle for RU. If we can extract both PU and RU, we can then apply SAT attack on both circuits only, without requiring an oracle LC. As TTLock and SFLL perform logic synthesis after insertions of PU and RU, the adversary needs an accurate identification of PU and RU under logic optimization. The extraction of RU during post-synthesis is straightforward because commercial CAD tools cannot merge it inside LC or PU when the key is unknown; however, the challenging part is to retrieve PU since CAD tools may partially merge PU inside LC. Using the directed acyclic graph analysis [155], there are multiple candidates for PU with full input X. One only needs to apply SAT attack to all possible PUs with the extracted RU and perform key validation in the end. *Note that we do*

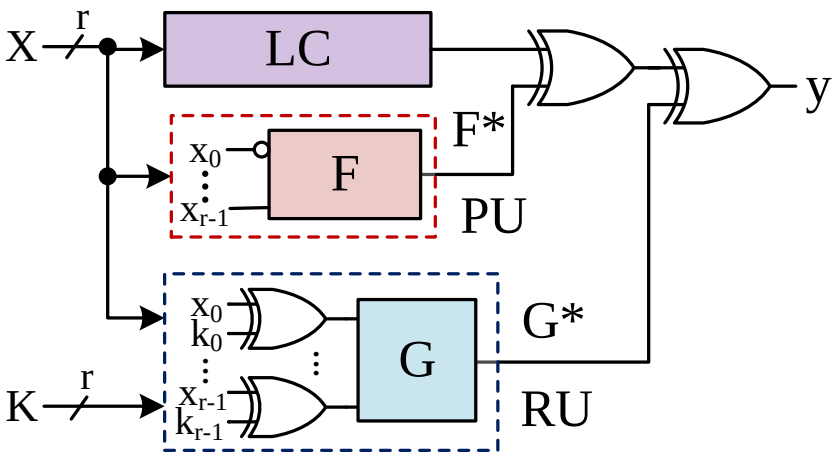

Figure 5.13: Generalized architecture of stripped functionality logic locking (SFLL). The functions F and G can be configured to implement TTLock and SFLL-HD^h.

not need an unlocked chip (serves as the oracle for traditional SAT attack) as the oracle is already present in the synthesized LC & PU circuit. Our future work is to find an efficient way to determine the valid oracle and identify the wrong ones from all extracted PUs.

5.5 Future Directions

5.5.1 Achieving Higher Time Complexity Against SAT Solvers

Even though point functions have demonstrated exponential iteration complexity, the adversary can formulate the attack with structural and functional analysis so that the complexity drops significantly. Although SAT attack has demonstrated linear trends in solving the secret key, it is still possible, in our opinion, for a logic design to achieve SAT resiliency. Here, we discuss how future locking schemes should consider a drastic increase in the hardness of their design against SAT attack from the example of c6288 benchmark. Besides targeting exponential iterations required for SAT attack, it may be feasible to significantly increase the overall time for SAT solver to find each satisfiable assignment for the miter circuit, which result in a longer computation time within each iteration.

As described in SAT attack [100], Subramanyan et al. stated that the multiplier benchmark c6288 is inherently challenging to SAT solvers, and was excluded from analysis. We locked its largest cone, N6288, in the same way as we did for other benchmarks in ISCAS'85, as described in Section 5.3. From the perspective of SAT attack iteration complexity, it remains

Table 5.4: Anatomy of SAT attack time on c6288_N6288.

$\|K\|$	$\|P\|$	CPU time (s)				$\frac{\text{UNSAT}}{\text{Total}}(\%)$
		Total	**IO Pairs**	**Average**	**UNSAT**	
1	1	86.351	0.09108	0.09108	86.25948	99.894
2	2	84.439	0.10289	0.05145	84.33634	99.878
3	3	86.551	0.11019	0.03673	86.44092	99.872
4	4	88.804	0.11963	0.02991	88.68458	99.865
5	4	79.614	0.12705	0.03176	79.48717	99.840
6	4	62.048	0.11630	0.02908	61.93153	99.812
7	4	88.088	0.11822	0.02955	87.97006	99.865
8	4	66.762	0.11330	0.02832	66.64887	99.830
9	5	78.434	0.12385	0.02477	78.31049	99.842
10	7	62.018	0.14788	0.02113	61.87004	99.761
11	8	72.615	0.15925	0.01991	72.45534	99.780
12	6	66.560	0.19532	0.03255	66.36468	99.706
13	9	74.612	0.22130	0.02459	74.39026	99.703
14	8	78.492	0.14760	0.01845	78.34455	99.811
15	10	77.133	0.17205	0.01721	76.96051	99.776
16	11	83.077	0.23765	0.02161	82.83926	99.713
17	11	85.083	5.70418	0.51856	79.37841	93.295
18	15	72.317	0.30082	0.02006	72.01650	99.584
19	15	89.654	0.34619	0.02308	89.30831	99.613
20	14	92.588	0.32586	0.02328	92.26268	99.648
21	15	67.431	0.45250	0.03017	66.97835	99.328
22	12	80.299	0.26642	0.02220	80.03259	99.668
23	19	88.228	0.63912	0.03364	87.58904	99.275
24	15	76.825	0.42104	0.02807	76.40369	99.451
25	20	88.295	2.48402	0.12420	85.81125	97.186
30	16	73.065	0.84954	0.05310	72.21507	98.837
35	29	86.737	14.53748	0.50129	72.19920	83.239
40	27	149.097	13.34636	0.49431	135.7502	91.048
45	41	1130.466	18.31241	0.44664	1112.154	98.380
50	37	84.404	6.16717	0.16668	78.23738	92.693
55	45	1188.844	57.14645	1.26992	1131.698	95.193

linear with key size $|K|$, as shown on the bottom-right plot in Figure 5.5 and Column 2 of Table 5.4. This suggests that c6288 behaves identically to other ISCAS'85 benchmarks. In addition, one can also observe that the complexity can decrease when more keys are inserted, as shown in Figure 5.6 and Column 2 of Table 5.4. The question is, *what makes the circuit structure of a multiplier challenging to SAT solver?* To better analyze the SAT complexity in breaking c6288_N6288, we record the CPU time spent for each iteration, including the very last

UNSAT round. We exclude the pre-processing time, *i.e.,* setting up arrays of literals, initializing solver, *etc*. The post-processing time is also excluded from the CPU time, *i.e.,* displaying the correct keys and the overall status, *etc*. Table 5.4 lists the time duration for SAT solver to derive the correct keys. The 1st and 2nd columns list key sizes $|K|$ and IO pair count $|P|$. Column 3 is the total time the SAT solver spent, which consists of two parts, (i) time used for generating all IO pairs, Column 4, and (ii) time checking that no more DIP exists (UNSAT), Column 6, where the averaged time it takes to find each IO pair is in Column 5. Column 7 reports the time ratio of the UNSAT decision over the total time spent on SAT solver. The interesting observation is that the major time spent was not on finding DIPs to prune off keyspace, but was on the last iteration, where SAT solver tries various backtracking before getting the UNSAT decision. The total time devoted to generating the IO pairs is negligible compared to the time spent in the very last iteration (UNSAT). In particular, the time duration for UNSAT in solving c6288 benchmark cones with respect to the total time span is generally over 90%.

In summary, it can be possible to achieve SAT attack resiliency by using hard-to-find DIPs rather than SAT iteration count. We convey this message by presenting two different case studies of post-SAT locking with point functions. We showed the key pruning analysis on a modified version of the SAT attack, which employs the identification of key gates and their inputs, can eliminate an exponential number of incorrect key combinations with respect to total key space. In order to achieve SAT resiliency, one may need to incorporate the same SAT attack time complexity for each DIP as the last iteration of UNSAT in the c6288 multiplier benchmark. The objective is to considerably increase the total backtracks and logic reassignment required for SAT solvers to find a DIP in every iteration so that a longer time duration can be achieved. We conjecture that future locking schemes can provide sufficient difficulty for the present-day SAT solvers with conflict-driven clause learning (CDCL) algorithm [194].

5.5.2 Controllability Analysis

Controllability analysis can be incorporated prior to the key insertion to achieve a strictly monotonically increasing linear iteration complexity. Controllability, widely used in VLSI testing, is defined as the difficulty of assigning the target signal to a logic 0 or a logic 1 [77]. A high value

indicates the easiness of setting a node to that desired logic value from the inputs. Figure 5.14 shows an example of how controllability analysis can help analyze SAT complexity. If the output of AND gate is logic 1 with high probability (see Figure 5.14(a)), its inputs can be uniquely determined, and the SAT attack can evaluate keys in parallel with the corresponding IO pair. This location is not a preferred location for inserting a key if the controllability of 1 is very high as many of the random input patterns will set this value. Instead, a very high probability of setting the same node to logic 0, as shown in Figure 5.14(b), could be a desired location for placing the key gate.

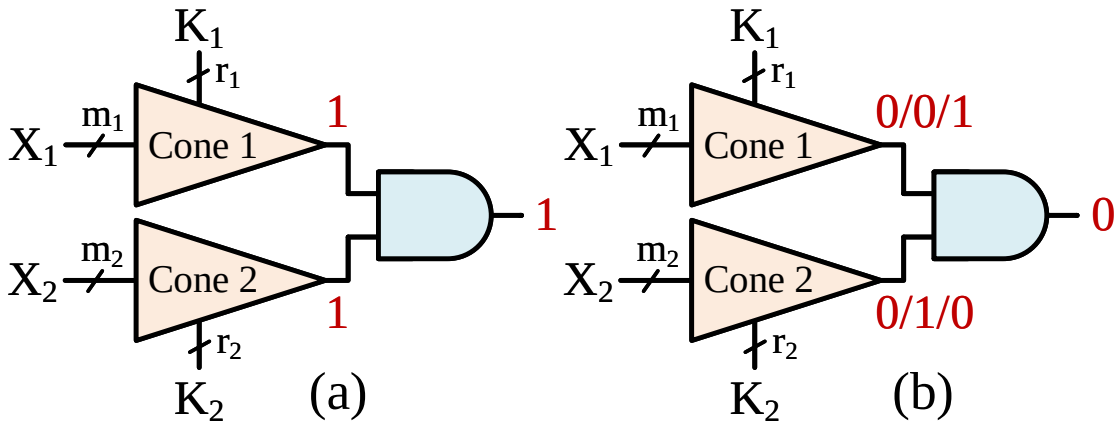

Figure 5.14: SAT attack key evaluation of circuit with an AND gate at the output. Oracle response of (a) logic 1, (b) logic 0.

Note that the defender's objective is to ensure DIPs are less effective in removing an exponential or sub-exponential number of incorrect keys (or finding a DIP). The goal is to keep the complexity in the order of $O(2^{K_1+K_2})$ (Figure 5.14(b)), not $\texttt{max}(O(2^{K_1}), O(2^{K_2}))$, equivalent to a logic 0 output at the AND gate in Figure 5.14(a). We envision, with controllability analysis, nodes with different output probabilities for logic 0 and 1 under different gate types could be a good indicator for the adaptive key insertion strategy.

5.5.3 Extension of the Proposed Complexity Analysis to the SMT Attacks

Our complexity analysis can be extended to the SMT attacks proposed in [181] due to the similarities between SMT and SAT. Satisfiability modulo theories (SMT), which consider the satisfiability of formulas under non-binary variables, offer more flexibility in the input space

than the binary space for SAT. SMT attacks expand the capability of the SAT attack to target non-functional-based attacks such as delay and timing-based logic locking [138]. Azar et al. [181] proposed four approaches: (i) Reduced SAT Attack, (ii) Eager SMT Attack, (iii) Lazy SMT Attack, and (iv) Accelerated Lazy SMT Attack. Our future work will explore and extend the proposed complexity analysis approach to these SMT attacks.

5.6 Summary

In this chapter, we provide a new perspective to analyze the efficiency of the SAT attack based on the CNF clause updates inside the SAT solver. In each iteration, SAT attack records the interdependencies between key bits from a distinguishing input pattern and its output response. Any locked circuit with multiple logic cones facilitates incorrect key removal as the effect of keys is propagated to multiple outputs. We further investigate the SAT attack complexity with the same cone of increasing key sizes. A non-monotonically increase in SAT complexity under increased key sizes is reported for the first time, where the insertion of additional key bits does not guarantee a strict linear growth in the SAT attack iteration complexity. Instead, this phenomenon of complexity drop happens to all ISCAS'85 benchmark cones. We subsequently provided an explanation of this observation from the oracle's response and logic gate types. It explains why more incorrect keys are eliminated from the keyspace with a particular IO pair. In addition, we give analytical reasoning to show how the constraining of key bits for post-SAT solutions like AntiSAT and CAS-Lock would aggressively reduce the key search down to constant or linear complexity. Finally, we furnish our discussions on SAT attack complexity analysis with novel observations on breaking the multiplier benchmark c6288, along with future directions.

Chapter 6

AFIA: ATPG-Guided Fault Injection Attack on Secure Logic Locking

Over the last few decades, the impact of globalization has transformed the integrated circuit (IC) manufacturing and testing industry from vertical to horizontal integration. The continuous trend of device scaling has enabled the designer to incorporate more functionality in a system-on-chip (SoC) by adopting lower technology nodes to increase performance and reduce the overall area and cost of a system. Currently, most SoC design companies or design houses no longer manufacture chips and maintain a foundry (fab) of their own. This is largely due to the increased complexity in the fabrication process as new technology development is being adopted. The cost for building and maintaining such foundries is estimated to be a multi-million dollar investment [195]. As modern integrated circuits (ICs) are becoming more complex, parts of the design are reused instead of designing the whole from scratch. As a result, the design house integrates intellectual properties (IP) obtained from different third-party IP vendors and outsources the manufacturing to an offshore foundry. Due to this distributed design and manufacturing flow, which includes designing SoCs using third-party IPs, manufacturing, testing, and distribution of chips, various threats have emerged in recent years [21, 22, 85]. The research community has also been extensively involved in proposing countermeasures against these threats [23, 84, 95–98, 196].

Logic locking has emerged as the most prominent method to address the threats from untrusted manufacturing [84–86, 197]. In logic locking, the netlist of a circuit is locked with a secret key so that the circuit produces incorrect results in regular operation unless the same key is programmed into the chip. Figure 6.1(a) shows an abstract view of logic locking where the

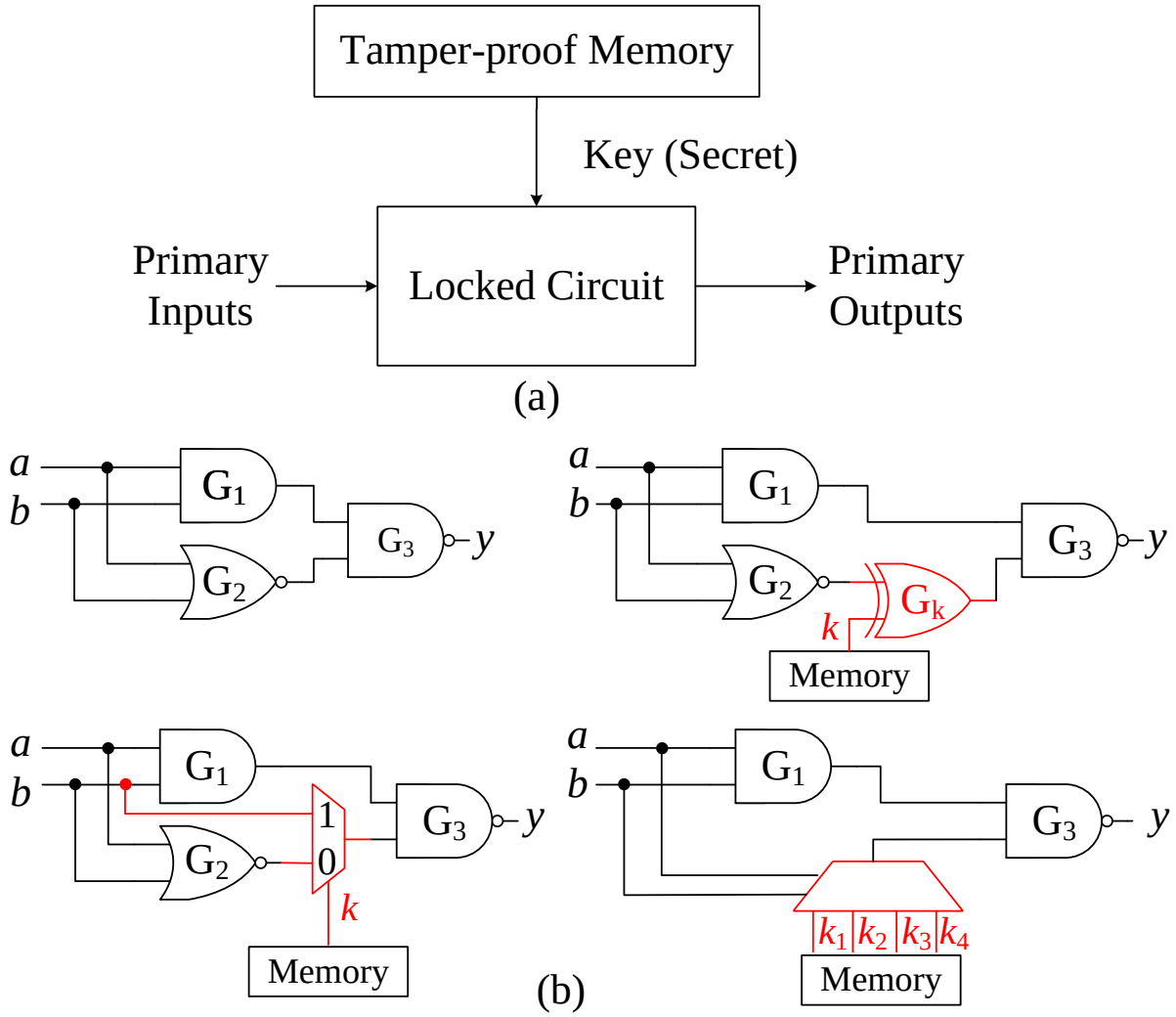

Figure 6.1: Logic Locking: (a) An abstract view of the logic locking. (b) Different types of logic locking techniques with XOR/XNOR, MUX and LUT.

key is stored in a tamper-proof memory and is applied to the locked circuit to unlock its functionality. The key needs to be kept secret, and care must be taken during the design process so that this secret key is not leaked to the primary output directly during the operation. The common logic locking techniques insert additional logic elements like XOR gates [84], multiplexers (MUXs) [99], and look-up tables (LUTs) [89] to lock the circuit functionality, and are shown in Figure 6.1(b). SAT attack, by Subramanyan et al. [100], was among the first ones to efficiently attack a range of locking schemes. With SAT analysis et al. [100], the key of a locked circuit is determined in a short period of time. The SAT attack requires the locked netlist, recovered through reverse engineering, and a functional working chip. Since then, several SAT-resistant locking techniques have emerged [105, 113, 118, 119, 138, 141–143, 169, 198–201] and many of

them were broken soon after they have been proposed [6,125,131,164,173,175,202–204]. The majority of the research has been directed towards SAT attack resiliency. *However, can we reliably state that a logic locking technique is completely secure even if we achieve complete SAT resistivity?* An untrusted foundry can be treated as an adversary as logic locking is proposed to protect designs from untrusted manufacturing. The adversary has many more effective means to determine the secret key without performing SAT analysis. A few of these attacks can be in the form of probing [78, 205], inserting a hardware Trojan in the design [158], and analyzing the circuit topology [156, 206, 207]. Countermeasures are also developed to partially prevent these attacks [125, 156, 206, 208–212].

Unlike cryptosystems, not all input patterns for a locked circuit are valid for propagating the incorrect key values to the primary outputs. Instead, only a few patterns may exist to carry the values of key bits to the output, similar to the identification of hard-to-detect faults. This is especially true for Post-SAT solutions [101, 102, 105–107], where they minimize the output corruptibility for incorrect keys. For logic locking, some key bits can block the propagation of the target key bit, *i.e.,* SLL [98] and Post-SAT designs. This is different from the fault injection attack in cryptography, where an entirely new output can be observed under any input pattern even though there is a single bit change in the key as the plaintext goes through many transformations (e.g., Shift Rows, Mix Columns and key addition for AES) [61, 213, 214]. It is trivial for a cryptosystem to change one key bit and apply a random pattern. Unfortunately, this is not the case for a circuit locked with a secret key. It is hard to observe the output change with the change of a single key bit by applying a random pattern. The novelty of this paper is that we apply the methodology in ATPG to efficiently derive the desired input pattern, which guarantees the change in output under different keys and helps launch the fault injection attack.

This chapter is organized as follows: An overview of different logic locking techniques and existing attacks along with fault injection techniques is provided in Section 6.1. We describe the previously published attack [157] in Section 6.2. The proposed attack and its methodology to extract the secret key from any locked circuit are described in Section 6.3. We present the results for the implementation of the proposed attack on different locked benchmark circuits in section 6.4. Finally, we conclude the chapter in Section 6.5.

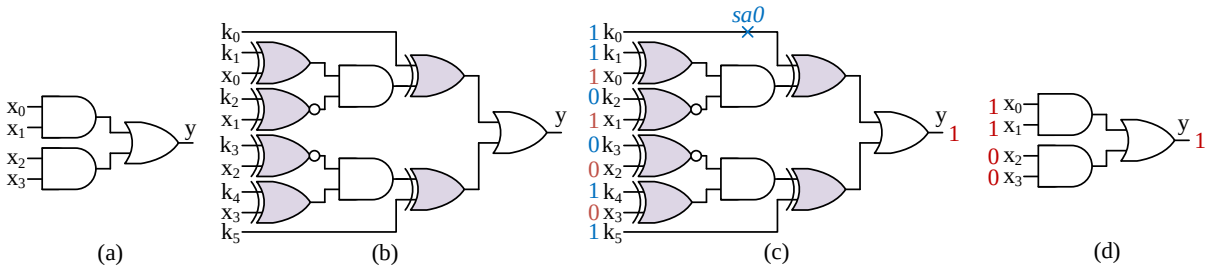

Figure 6.2: The inefficiency of CLIC-A attack. (a) Original circuit. (b) Locked circuit with 6 dependent key gates, where correct key $\{k_0, ..., k_5\} = \{001100\}$. (c) Key assignment and input pattern returned by Constraint-based CLIC-A. (d) The oracle response $y = 1$.

6.1 Prior Work in existing logic locking and attacks

As mentioned in Chapter 2, the objective of logic locking is to obfuscate the functionality of the original circuit by inserting a lock (secret key). The key-dependent circuit makes it difficult for the adversary to pirate or analyze the original circuit directly. In this context, various traditional logic locking techniques were based on different location selection algorithms for key gate placement, such as random (RLL) [215], fault analysis-based (FLL) [98], and strong interference-based logic locking (SLL) [99]. To demonstrate the capabilities of an adversary, Subramanyan et al. [100] developed a technique using Boolean Satisfiability (SAT) analysis to obtain the secret key from a locked chip. This oracle-guided SAT attack iteratively rules out incorrect key values from the key space by using distinguishing input patterns and the corresponding oracle responses.

In post-SAT era, resiliency against the SAT attack became one of the crucial metrics to demonstrate the effectiveness of newly proposed schemes [204]. Sengupta et al. proposed stuck-at-fault based stripping of original netlist and reconstruction to form the locked netlist, where incorrect results are produced only for chosen input patterns [105, 201]. Simultaneously, researchers have adopted a different direction to tackle the SAT attack, including restricting access to the internal states of a circuit through scan-chains. Guin et al. proposed a design that prevents scanning out the internal states of a design after a chip is activated and the keys are programmed/stored in the circuit [200, 216]. The concept of scan locking gained significant interest from the researchers, which led to the development of various scan-chain-based

locking schemes [127, 131, 199]. Alrahis et al. attack scan-chain-based locking schemes by unrolling the sequential circuit to a combinational one, which is then provided to the SAT solver to extract the secret key [217]. Sisejkovic et al. [211] proposed an oracle-less structural analysis attack on MUX-based (SAAM) logic locking to exploit insertion flaws in MUX-based key gates. Deceptive multiplexer-based (D-MUX) logic locking is proposed to achieve functional secrecy [165] against both SAAM and oracle-less machine learning-based attacks. As combinational feedback loops are not translatable to SAT problems, cyclic-based locking [113] is resistant to the initial SAT attack [100]. In addition, there has been extensive efforts in the proposal of non-functional logic locking techniques, such as scan-chain-based [131, 218], timing-based locking [138, 141–143], and routing-based locking [118, 119, 125].

As the research community explores new directions to understand an attacker's latent qualities, new attacks on logic locking have been proposed. An adversary may perform direct or indirect probing on the key interconnects or registers [78]. An attacker is not required to understand the complete functionality of the circuit to perform these attacks. In this, Rahman et al. demonstrated how an attacker could target the key registers and perform optical probing to gain knowledge regarding the fixed value for those registers. Following this, tampering attacks can also become an attacker's primary choice. Jain et al. exploited this notion to extract the secret key by implementing hardware Trojans inside the locked netlist [158]. Without an oracle, the attribute of repeated functionality in the circuit can also be used to compare the locked unit functions and their unlocked version to predict the secret key [156]. This makes it necessary to lock all instances of unit functions in the entire netlist to achieve a secured logic locking scheme. CLIC-A [164, 203], an ATPG-based attack, can break keys by applying constraint-based ATPG to propagate the target key bit to the primary output but suffers scalability in the dependent key count. Cyclic-based locking has suffered from modified SAT-based attack [169], where cyclic-based constraints are placed to avoid infinite loops. Several non-functional-based locking can be broken by sequential-based attacks with limited scan access [173, 175, 204] or SMT attack [181].

6.1.1 Comparison of AFIA with CLIC-A

There is a major difference between our proposed *AFIA* and *CLIC-A* [164]. First, the worst-time complexity for test generation regarding the total test pattern count for solving the key-dependent faults between *AFIA* and *CLIC-A* differs significantly. Our worst-case complexity of solving an n-bit key of non-mutable convergent key gates inside a single logic cone is at most n test patterns with $\frac{n\cdot(n-1)}{2}$ injected faults (see Theorem 6.4). This is because AFIA determines each key bit by directly comparing the output response with the generated test pattern. On the other hand, *CLIC-A* applies constraint-based ATPG by assigning constraint on the $(n-1)$-bit key and setting a stuck-at 0 at the target key line since placing don't cares (*X*) on other key bits will not produce the desired test patterns for non-mutable convergent key gates. However, the simulated output from the constraint-based ATPG likely agrees with the oracle simulation under the same test pattern hardness of logic locking. Note that it does not mean that the constraints placed on the $(n-1)$-bit key is the correct key values, and it only indicates that, under the particular test pattern, the output from the netlist with constraints and the stuck-at fault matches with the oracle output. Instead, *CLIC-A* has to perform additional constraints in ATPG and check the output against the oracle to ensure that key values are correct. The worst-case complexity in the total test pattern count for *CLIC-A* is exponential $\boldsymbol{O}(2^{(n-1)})$.

Let us consider an example of an unlocked circuit in Figure 6.2(a) locked with 6 dependent XOR/XNOR key gates, as shown in Figure 6.2, whose correct key $\{k_0, ..., k_5\} = \{001100\}$. As none of the keys can be sensitized to the output without knowing the correct value for the other five, CLIC-A runs constraint-based ATPG and sets *sa0* to k_0. Suppose ATPG returns a test pattern with key vector $\{k_0, ..., k_5\} = \{110011\}$ and input vector $\{x_0, ..., x_3\} = \{1100\}$, along with the simulated fault-free output $\texttt{atpg-sim}(x_0, ..., x_3, k_0, ..., k_5) = 1$, as shown in Figure 6.2(c). Although the output of the simulated netlist matches with the oracle response $y = 1$, the key value returned by constraint-based ATPG is incorrect. There is no method for CLIC-A to check whether the key vector is the actual key other than appending it as a constraint to ATPG so that the test pattern returned at the next iteration would be different from the current one. The worst-case complexity for CLIC-A to fully determine the 6-bit key is to

iterate through all possible combinations of the remaining 5-bit key (excluding k_0 with *sa0*), resulting in a 2^5 test pattern count to break the locking scheme with dependent keys. On the other hand, AFIA only requires 6 test patterns to determine all 6-bit keys, which is much more efficient than CLIC-A. In summary, test pattern generation for CLIC-A becomes infeasible if there are a large number of dependent keys in a logic cone.

6.1.2 Comparison of AFIA with Key Sensitization Attack

There is a similarity between our proposed AFIA and sensitization attack [98]. The similarity between these approaches is the sensitization, i.e., the propagation, of the key to the output. However, our approach is more general for the following reason. First, the sensitization attack does not need fault activation as the key gates are XOR/XNOR gates, and the key can propagate to the key gate output for both input 0 and 1. However, this may not hold for non-XOR-based locking techniques. For example, MUX-based locking has keys connected to the input of AND gate instead of the XOR gate, where one needs to set the other AND input to the non-controlling value 1 for fault activation. Besides, it is common practice for recent locking techniques to synthesize the locked benchmark after key insertion. The synthesis tool can optimize the key gate with other gate types, which results in keys directly connected to non-XOR gates like AOI, NAND, etc. To propagate the key value to the primary output, having only key sensitization without the activation would not work for synthesized locked circuits. For example, we can break *SFLL-hd* [105], *SFLL-flex* [105], and *SFLL-rem* [219] with n patterns for a n-bit key (see Section 6.3.7), where sensitization attack requires brute force attack ($\boldsymbol{O}(2^n)$) to all the non-mutable keys in the *SFLL* restoration circuitry. Second, our proposed fault injection can break non-mutable convergent key gates from strong logic locking, which is the countermeasure proposed in a sensitization attack. AFIA only needs at most n (see Theorem 6.2) test patterns for a n-bit pairwise non-mutable convergent keys, but it would take $\boldsymbol{O}(2^n)$ in the worst case to brute force the correct key under sensitization attack [98].

6.1.3 Dissimilarities between Logic Locking and Cryptosystems

There has been considerable efforts [105, 220] in the proposal of formal analysis on logic locking through introducing similar concepts used in cryptography. However, logic locking techniques differ from various cryptosystems in two aspects. First, the objective for logic locking and cryptosystem is different. The cryptographic algorithm ensures that the secret key is fully integrated with the input plaintext (*i.e.,*, the addRoundKey in all ten rounds of AES encryption). Logic locking, however, focuses on perturbing the output, commonly by XORing a 1-bit key with a wire in the circuit, under certain input patterns, where no repeated insertion of the same key bit or its derived value to elsewhere. Second, the output of a locked circuit and the ciphertext of a cryptosystem behaves differently under input combinations. A locked circuit under an incorrect key may behave identically to the oracle (or locked circuit with the correct key) under multiple input patterns. This is particularly true for Post-SAT locking solutions, *i.e.,*, SARLock [101], Anti-SAT [102], SFLL [105, 106], CAS-Lock [107], where the output corruptibility for incorrect keys is reduced to the bare minimum. This means that a locked circuit with an incorrect key behaves exactly as an unlocked circuit under an exponential number of input combinations.

The cryptographic algorithms, especially the block ciphers, are built on confusion and diffusion properties recommended by Claude Shannon in his classic 1949 paper [221]. This results in a large number of output bit changes in the output (ciphertext) even for a single bit change in the input (plaintext) [213, 214]. For example, AES has 10/12/14 rounds of diffusion and confusion operations depending on the key size of 128/192/256 bits. It is thus trivial to launch differential fault analysis as it will guarantee the change in the output, where one can compare the faulty and fault-free responses by injecting a fault into a key register, one at a time. On the contrary, digital circuits generally do not have repeated layers of operations like block ciphers. Digital circuits, except crypto accelerators, are designed to meet the user specification of speed, power, and area, and the functionality (change in output) depends on the user's needs. It is well understood and verified that digital circuits have lots of don't cares (Xs) in the inputs. The VLSI test community adopted test compression [222, 223] to reduce the test pins and

resultant test times. As there exists a large number of Xs in the test pattern, it is infeasible to apply a random pattern and expect it to propagate the target key bit (e.g., a stuck-at fault at the key line) to output. For example, if there are 70% Xs in a test pattern with a 100 input cone [which is very common], the probability of a random pattern propagating the key to the output is $2^{30}/2^{100} \approx 0$. The effect of some keys in a locked circuit can even be muted due to the circuit's structural and functional behavior [98], which is in direct contrast to cryptosystems, where every output is influenced by all key bits [214].

6.1.4 Fault Injection Methods

Over the years, several threats and methods have emerged to break a cryptosystem without performing mathematical analysis or brute force attacks. Using these attacks, an adversary can subvert the security of protection schemes, primarily through extracting or estimating the secret key using physical attacks. Fault injection attacks intentionally disturb the computation of cryptosystems in order to induce errors in the output response. To achieve this, external fault injection is performed through invasive or non-invasive techniques. This is followed by the exploitation of erroneous output to extract information from the device.

Fault-based analysis on cryptosystems was first presented theoretically by Boneh et al. on RSA [224]. This contribution initiated a new research direction to study the effect of fault attacks on cryptographic devices. The comparison between the correct and faulty encryption results has been demonstrated as an effective attack to obtain information regarding the secret key [225–227]. These can be realized into different categories:

- *Clock Glitch*: The devices under attack are supplied with an altered clock signal which contains a shorter clock pulse than the normal operating clock pulse. For successfully inducing a fault, these clock glitches applied are much shorter than the circuit's tolerable variation limit for the clock pulse. This results in setup time violations in the circuit and skipping instructions from the correct order of execution [228, 229].

- *Power Variation*: This technique can be further bifurcated into two subcategories: either the malicious entity may choose to provide a low power supply to the system (also

abbreviated as underfeeding), or the adversary may choose to influence the power line with spikes. This adversely affects the set-up time and influences the normal execution of operations. The state elements in the circuit are triggered without the input reaching any stable value, causing a state transition to skip operations or altering the sequence of execution [230–232].

- *Electromagnetic Pulses/Radiation*: The eddy current generated by an active coil can be used to precisely inject faults at a specific location in the chip. This method does not require the chip to be decapsulated in order to inject the fault. However, the adversary is required to possess information regarding specific modules and their location inside the chip [233, 234].

- *Laser*: Fault injection using lasers is also regarded as a very efficient method because it can precisely induce a fault at an individual register to change its value [69]. For optical fault injection, the laser can be focused on a specific region of the chip from the backside or front side. However, due to the metal layers on the front side, it is preferred to perform the attack on the backside of the chip. Skorobogatov et al. [235] first demonstrated the effectiveness of this method by using a flashgun to inject fault to flip a bit in the SRAM cell. Several other research groups also utilized and proposed different variants of this method to study the security of cryptographic primitives [236–239].

- *Focused-ion Beam (FIB)*: The most effective and expensive fault injection technique is devised with focused ion beam (FIB) [46]. This method enables cutting/connecting wires and even operates through various layers of the IC fabricated in the latest technology nodes [240].

- *Software-based Fault Injection*: This technique produces errors through software that would have been produced when a fault targeted the hardware. It involves the modification of programs running on the target system to provide the ability to perform the fault injection. It does not require dedicated complex hardware, a gate-level netlist, or RTL models that are described in hardware description languages. The faults are injected into

accessible memory cells such as registers and memories through software that represent the most sensitive zones of the chip [241–243].

6.2 Differential Fault Analysis (DFA) attack

In this section, we present a differential fault analysis (DFA) attack introduced in [157]. Our attack method is inspired by VLSI test pattern generation. One test pattern is able to detect a single stuck-at fault with the propagation of this fault to the primary output. Since key values from tamper-proof non-volatile memory are loaded to key registers, these registers are the potential locations for stuck-at-faults. With an active chip at hand, the adversary could target these registers and extract the secret key.

6.2.1 Threat Model

The threat model defines the capabilities of an adversary and its standing in the IC manufacturing and supply chain. It is very important to know an attacker's ability and the available resources/tools to estimate its potential to launch the attack. The design house or entity designing the chip is assumed to be trusted. The attacker is assumed to be the untrusted foundry or a reverse engineer having access to the following:

- The locked netlist of a circuit. An untrusted foundry has access to all the layout information, which can be extracted from the GDSII or OASIS file. Also, this locked netlist can be reconstructed by reverse engineering the fabricated chip in a layer-by-layer manner with advanced technological tools [46].

- An unlocked and fully functional chip is accessible to the adversary since the chip is publicly available from the market.

- A fault injection equipment is essential to launch the attack. It is not mandatory to use high-end fault injection equipment. The main operation is to inject faults at the locations of key registers (all the flip-flops) on a de-packaged/packaged chip. Precise control is not necessary as we target all the flip-flops simultaneously. An adversary can also choose

the software methods to inject faults at these flip-flops. Once the register is at the faulty state, the scan enable (*SE*) signal needs to be assigned to put the chip in test mode.

- The attacker has the know-how to determine the location of the tamper-proof memory. Then, it will be trivial for an adversary to find the location of the key register in a netlist, as it can easily trace the route from the tamper-proof memory.

Notations: To maintain uniformity across the entire paper, we represent frequently used terms with the defined notations, and they will be referred to with these notations in the following subsections.

- $\mathcal{K}$ denotes key length or key size, i.e., the number of bits in the key.
- K denotes the keyspace; $K = \{k_0, k_1, \ldots k_{\mathcal{K}-1}\}$.
- The locked netlist of a circuit is abbreviated as C_L. The unlocked and fully functional chip/circuit, whose tamper-proof memory has been programmed with the correct key, is denoted by C_O. The two versions of fault-injected circuits are described as follows:
  - C_F represents a locked circuit where all the key lines ($\mathcal{K}$) are injected with logic 1 (or logic 0) faults. We call it the circuit with faulty key registers for differential fault analysis (DFA).
  - C_A represents the same locked circuit in which $(\mathcal{K} - 1)$ key lines are injected with the same logic 1 (or logic 0) faults, leaving one key line fault-free. We denote this circuit as a fault-free circuit for DFA.

  For any given circuit, we assume the primary inputs (PI) of size $|PI|$, primary outputs (PO) of size $|PO|$, and secret key (K) size of $\mathcal{K}$. We also use key lines or key registers alternatively throughout this paper as their effects are the same on a circuit.
- Stuck-at fault (*saf*): For any circuit modeled as a combination of Boolean gates, stuck-at fault is defined by permanently setting an interconnect to either 1 or 0 in order to generate a test vector to propagate the fault value at the output. Each connecting line can

have two types of faults, namely, stuck-at-0 (*sa0*) and stuck-at-1 (*sa1*). Stuck-at faults can be present at the input or output of any logic gates [77].

- Injected fault: A fault is injected at the key register using a fault injection method (see details in Section 6.1).

Note that *saf* is an abstract representation of a defect to generate test patterns, whereas an injected fault is the manifestation of a faulty logic state due to fault injection.

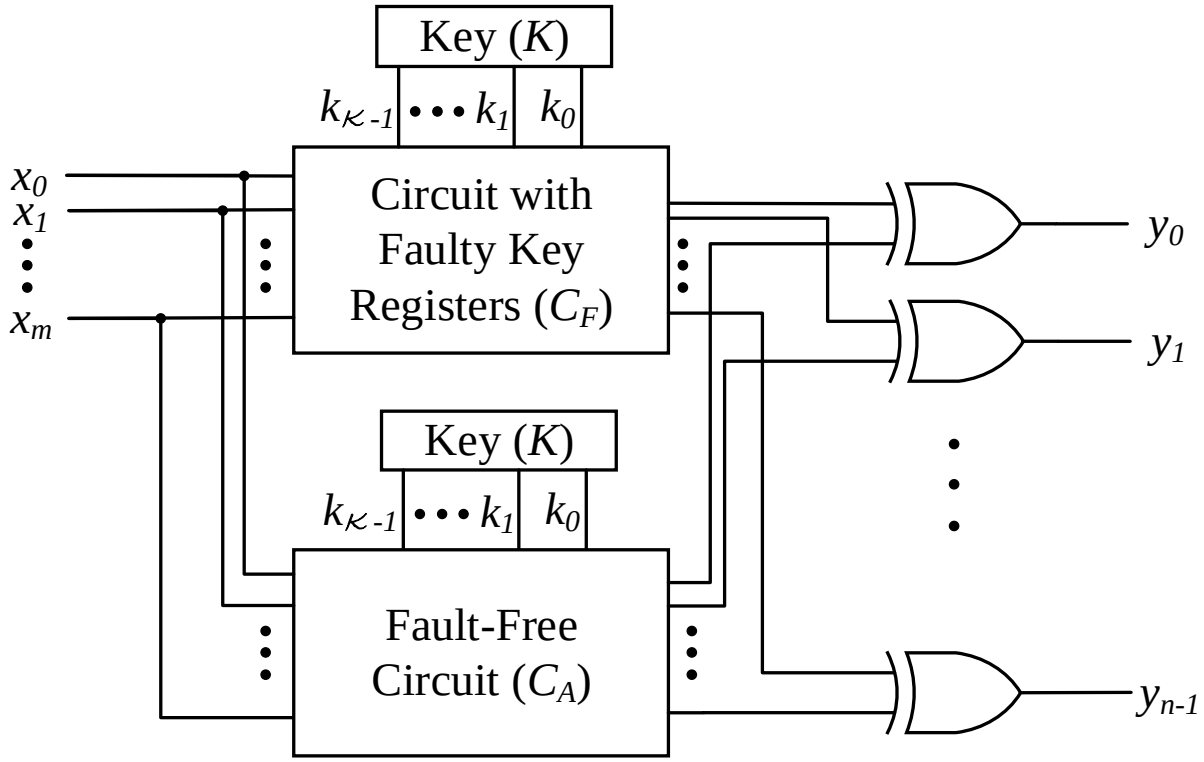

Figure 6.3: The abstract representation of our DFA attack.

6.2.2 Differential Fault Analysis (DFA) Attack Methodology

This fault injection attack relies on differential fault analysis. The captured output response of the circuit with faulty key registers with the corresponding fault-free circuits can reveal the key. Applying any fault injection methods (see the details in Section 6.1.4), the attacker can create the faulty chip/circuit. Figure 6.3 shows an abstract representation of DFA. The fault-free circuit (C_A) is an unlocked chip (C_O) bought from the market whose key bits need to be retrieved. Except for the key-bit targeted to be extracted, all remaining key registers are fixed to

a particular faulty value of either 0 or 1 corresponding to the selected fault. A circuit with faulty key registers (C_F) uses the same chip, and it is injected with a particular fault to keep all the key registers or interconnects to a faulty value of logic 1 or 0. One input pattern is first applied to C_A, and its response is collected. The same input pattern is then applied to the C_F to collect the faulty response. By XORing the corresponding circuit response, any output discrepancy between fault-free circuit (C_A) and the circuit with faulty key registers (C_F) is revealed. If both the circuits differ in their responses, the XORed output will be 1; otherwise, it will be 0. If we find an input pattern that produces a conflicting result for both C_A and C_F only for one key bit, the key value can be predicted. The key value is the same as the injected fault value if the XORed output is of logic 0; otherwise, the key value is a complement to the injected fault.

The attack can be described as follows:

- *Step-1*: The first step is to select an input pattern that produces complementary results for the fault-free (C_A) and faulty (C_F) circuits. The input pattern needs to satisfy the following property – it must sensitize only one key bit to the primary output(s). In other words, only the response of one key bit is visible at the *PO*, keeping all other key bits at logic 1s (or 0s). If this property is not satisfied, it will be impractical to reach a conclusion regarding the value of a key bit. *Now the question is, how can we find if such a pattern exists in the entire input space (ξ).*

  To meet this requirement, our method relies on stuck-at faults (*saf*) based constrained ATPG to obtain the specific input test patterns (see details in Section 6.2.4). Considering the fact that the adversary has access to the locked netlist, it can generate test patterns to detect *sa1* or *sa0* at any key lines and add constraints to other key lines (logic 1 and 0 for *sa1* and *sa0*, respectively). A single fault, either *sa0* or *sa1* on a key line, is sufficient to determine the value of that key bit. Therefore, we have selected *sa1*, and the following subsections are explained considering this fault only. This process is iterated over all the key bits to obtain $\mathcal{K}$ test patterns. The algorithm to generate the complete test pattern set is provided in Algorithm Section 6.2.4.

- *Step-2*: The complete set of generated test patterns is applied to the fault-induced functional circuit with faulty key registers (C_F). The circuit is obtained by injecting logic 1 fault on the key registers if *sa1* is selected in the previous step; else, the circuit is injected with logic 0 faults for *sa0*. The responses are collected for later comparison with fault-free responses. For C_A, test patterns are applied such that it matches the fault modifications in the circuit. For example, the test pattern for the first key is applied to the circuit when the circuit instance does not pertain to any fault on its corresponding key register and holds the correct key value while the remaining key registers are set to logic 1 (for *sa1*) or 0 (for *sa0*). For the next key-bit, (C_A) instance is created by excluding this selected key bit from any fault while keeping all other key registers to logic 1 (for *sa1*) or 0 (for *sa0*). This process is repeated for all key bits, and their responses are collected for comparison in the subsequent step.

- *Step-3*: The adversary will make the decision regarding the key value from the observed differences in the output responses of (C_A) and (C_F). For any test pattern corresponding to a particular key bit, when the outputs from both circuits are the same, it implies that the injected fault on the key lines in a C_F circuit is the same as the correct key bit; only then will the outputs of both ICs be same. Otherwise, when C_F and C_A differ in their output response, it concludes the correct key bit is a complement to the induced fault. This process is repeated for all key bits. In this manner, the key value can be extracted by comparing the output responses of both circuits for the same primary input pattern.

6.2.3 Example

We choose a combinational circuit as an example for simplicity to demonstrate the attack. The attack is valid for sequential circuits, as well, as it can be transformed into a combinational circuit in the scan mode, where all the internal flip-flops can be reached directly through the scan-chains [77].

Figure 6.4 shows the test pattern generation on a circuit locked with a 3-bit secret key, where the propagation of k_0 is dependent on k_1 and vice versa. First, we target to find out the

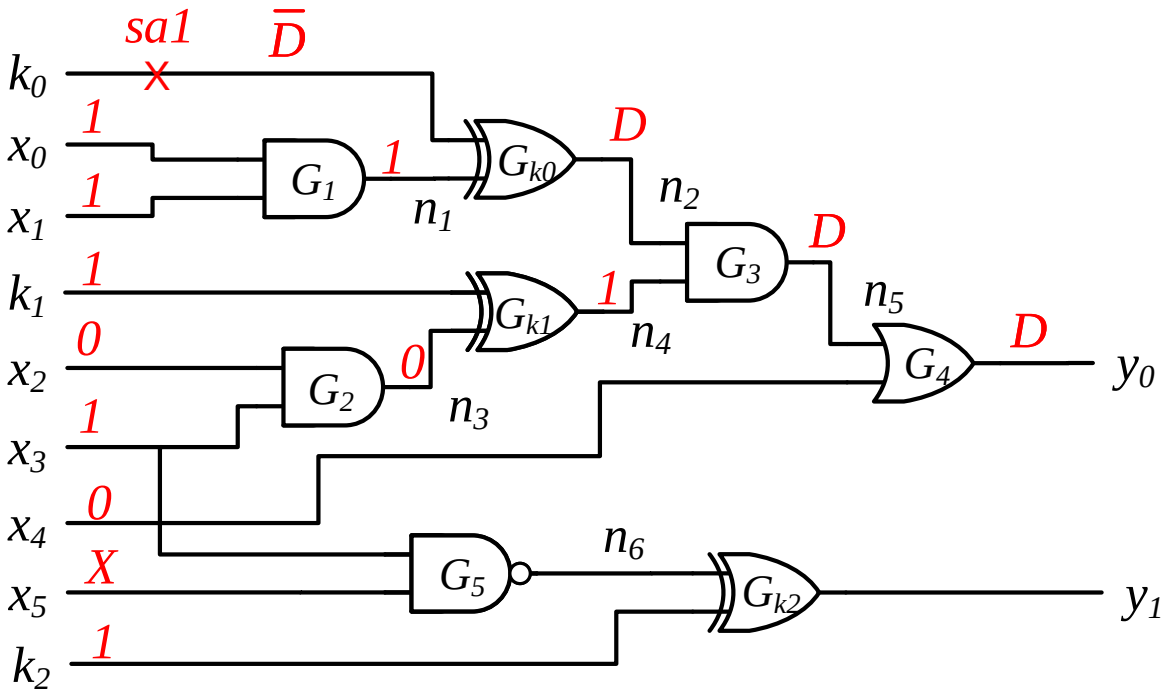

Figure 6.4: Test pattern generation considering a *sa1* at key line k_0 with constraint $k_1 = 1$ and $k_2 = 1$. Test pattern, $P_1 = [11010X]$ can detect a *sa1* at k_0.

value of k_0. A test pattern P_1 is generated to detect a *sa1* fault at k_0 with constraint $k_1 = 1$ and $k_2 = 1$ (adding faults on all the key lines except the target key bit). As the value of k_1 is known during the pattern generation, the effect of the *sa1* at k_0 will be propagated to the primary output y_0. For a fault value $\overline{D}$ at k_0, if $[x_0\ x_1] = [1\ 1]$ then D propagates to n_2. To propagate the value at n_2 to the output of G_3, its other input (n_4) needs to attain logic 1. Since $k_1 = 1$ due to injected fault which is set as a constraint in ATPG tool, $n_4 = 1$ for $n_3 = 0$ which implies $[x_2\ x_3] = [0\ 1]$. At last, $x_4 = 0$ propagates D propagates the value at n_5 to the primary output y_0. The output y_0 can be observed as D for the test pattern $P_1 = [1\ 1\ 0\ 1\ 0\ X]$. Finally, to perform the DFA, this pattern P_1 needs to be applied to both C_A and C_F to determine the value of k_0. Similar analysis can be performed for the other two key bits, k_1 and k_2.

6.2.4 Test Pattern Generation

To generate the test pattern set, an automated process relying on constrained ATPG is performed. The detailed steps to be followed are provided in Algorithm 2. Synopsys Design Compiler [189] is utilized to generate the technology-dependent gate level netlist and its test

Algorithm 2: Test pattern generation for constrained ATPG in DFA

Input : Locked gate-level netlist (C_L), test protocol (T), and standard cell library
Output: Test pattern (*P*) set

```
1  Read the locked netlist (C_L) ;
2  Read standard cell library ;
3  Run design rule check with test protocol generated from design compiler ;
4  Determine key size K from C_L ;
5  for i ← 0 to (K − 1) do
6      Add a sa1 fault at key line k_i ;
7      for j ← 0 to (K − 1) do
8          if i ≠ j then
9              Add constraint at k_j to logic 1 ;
10         end
11     end
12     Run ATPG to detect the fault ;
13     Add the test pattern, P_i to the pattern set, P ;
14     Remove all faults ;
15     Remove all constraints ;
16 end
17 Report the test pattern set, P ;
```

protocol from the RTL design. A test protocol is required for specifying signals and initialization requirements associated with design rule checking in Synopsys TetraMAX [244]. Automatic test generation tool TetraMAX generates the test patterns for the respective faults along with constraints for the locked gate level netlist.

The inputs to the algorithm are the locked gate-level netlist (C_L), Design Compiler generated test protocol (T), and the standard cell library. The algorithm starts with reading the locked netlist and standard cell library (Lines 1-2). The ATPG tool runs the design rule check with the test protocol obtained from the Design Compiler to check for any violation (Line 3). Only upon the completion of this step is the fault model environment set up in the tool. The size of the key ($\mathcal{K}$) is determined by analyzing C_L (Line 4). The remaining key lines are selected one by one to generate test patterns (Line 5). A stuck-at-1 fault is added at the i^{th} key line to generate P_i (Line 6). The ATPG constraints (logic 1) are added to other key lines (Lines 7-11). A test pattern P_i is generated to detect the *sa1* at the i^{th} key line (Lines 12-13) and added to the pattern set, P. All the added constraints and faults are removed to generate the $(i+1)^{th}$ test pattern (Lines 14-15). Finally, the algorithm reports all the test patterns, P (Line 17).

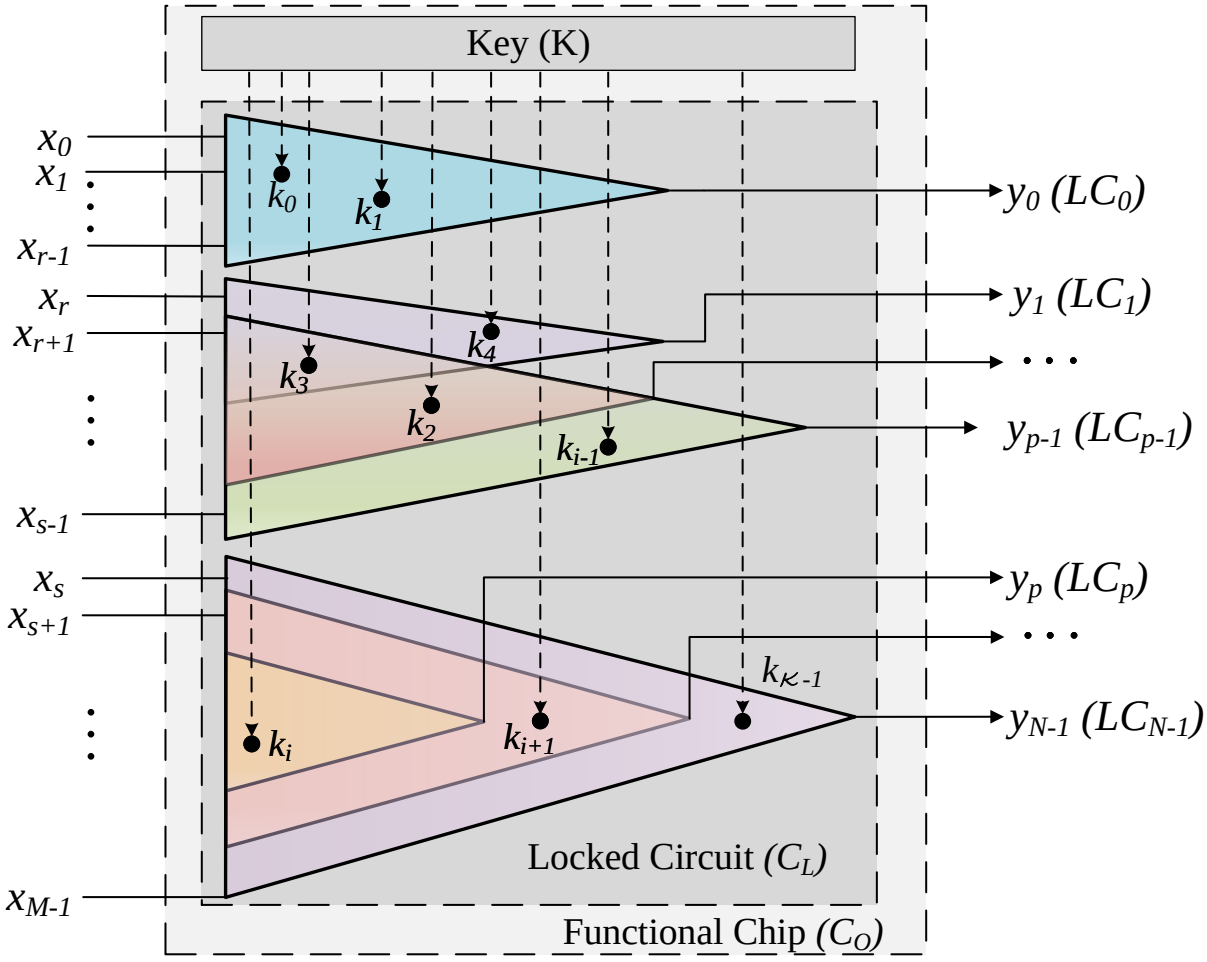

Figure 6.5: An abstract view of a locked circuit.

6.3 AFIA: ATPG-guided Fault Injection Attack

The objective of an adversary is to reduce the number of injected faults to launch an efficient attack. The DFA presented in Section 6.2.2 requires $2\mathcal{K}-1$ faults to determine a single key bit, where $\mathcal{K}$ denotes the secret key size. This severely limits the adversary's capability as injecting a large number of faults is challenging from the fault injection equipment's perspective. All these faults need to be injected when applying the test pattern to evaluate one key bit. In this section, we present an efficient attack and denoted as ***AFIA***, an ***A***TPG-guided ***F***ault ***I***njection ***A***ttack based on key sensitization. This new attack only requires injecting the fault on a key register if there is a dependency among keys. The threat model remains the same as DFA. We consider an untrusted foundry to have access to the gate-level netlist and can generate manufacturing test patterns.

6.3.1 Overall Approach

The proposed attack AFIA evaluates one key bit at a time iteratively and can be summarized by the following steps:

- *Step-1*: First, AFIA analyzes the locked circuit C_L and its logic cones. Some cones are completely independent (e.g., LC_0 in Figure 6.5), some cones share few inputs (e.g., LC_1, ..., LC_{p-1}), and the others share the same inputs (e.g., LC_p, ..., LC_{N-1}). It is necessary to determine keys from cones that are a subset of other larger cones (if any) first during the test pattern generation in order to reduce the number of injected faults. For an independent logic cone (say LC_0), we can propagate the keys one at a time without injecting faults at keys of other cones. If the two cones are overlapped, it is beneficial to sensitize keys to a cone with fewer unknown keys.

- *Step-2*: Similar to DFA, it requires an input pattern to derive a correct key bit. We denote this key bit as the target key bit. Constraints are set on the recovered key lines, where no fault injections are needed. The attacker performs fault injection (*Step-3*) solely on keys (in the same cone) that block the propagation of the targeted key bit. The blocking key set is determined by the returned test patterns from ATPG TetraMAX [244]. Once a key bit is determined, AFIA targets the next key bit of the same cone by putting the previously obtained keys as constraints during the test pattern generation.

- *Step-3*: The last step applies fault injections on functional chip C_O using the generated test patterns of *Step-2*. The targeted key value can be extracted by comparing the fault-injected output against the output pattern computed by ATPG. When the value of all the targeted key bits in one text pattern has been identified, we can constrain these bits with their actual values in ATPG in the subsequent pattern.

AFIA is an iterative method, where *Step-2* is performed to generate test patterns, and *Step-3* injects fault and applies that pattern to determine the targeted key bit. Once this targeted key is determined, it will be used as a constraint in *Step-2*. The following subsections present these three steps in detail.

6.3.2 Cone Analysis

The goal of this proposed attack is to apply minimal fault injections to recover the complete key set. It is ideal for the adversary to inject faults at key registers only when necessary. In general, not all keys prevent the propagation of the target key bit, as many of the keys are often distributed across the netlist and reside in different logic cones. A logic cone is a part of the combinational logic of a digital circuit that represents a Boolean function and is generally bordered by an output and multiple inputs [77]. Thus, cone analysis can effectively separate the dependence of different groups of key bits, where one group does not block the propagation of the key bits in other groups. We propose to analyze the internal structure of the locked netlist C_L by creating a directed graph G from it. We denote that both the inputs and logic gates' outputs are nodes. A directed edge exists from Node n_1 to Node n_2 if and only if they are associated with a logic gate. Intuitively, a circuit with N outputs has N logic cones, as in Figure 6.5. Note that the number of cones can be only primary outputs (POs) for a combinational circuit or the sum of POs and pseudo primary outputs (PPOs) for a sequential circuit [77]. All the inputs and logic gates whose logical values affect y_j belong to logic cone LC_j. The graph representation of logic cone LC_j with sink y_j is a subgraph of G.

Two possible scenarios might occur during the locking of a netlist. Key bit(s) can be placed uniquely in a logic cone and cannot be sensitized to any other POs/PPOs except the cone's output. Other key bits can be placed in the intersection of multiple cones and can be sensitized through any of these. We observe that the majority of the key bits are inside the intersections with multiple cones. What should be the best strategy to propagate a key bit to one of the POs/PPOs when there exist multiple sensitization paths? Our objective is to reduce the number of faults to sensitize a key bit to a PO/PPO, and it is beneficial to select a cone with the minimum number of keys. Note that the keys in a cone can block the propagation of a targeted key in that same cone only and requires fault injection to set a specific value to these blocking keys. It is, thus, necessary to construct a key-cone association matrix *A* to capture the correlation between the logic cones and the key bits. The matrix A not only provides insight on which keys (and how many of them) are inside a logic cone but also offers a structured view of

whether a key belongs to multiple logic cones, and is presented as follows:

$$
\begin{aligned}
A &= [a_{i,j}]_{\mathcal{K}\times N} \\
&= \begin{array}{c} \\ k_0 \\ k_1 \\ k_2 \\ \vdots \\ k_{\mathcal{K}-1} \end{array}
\begin{array}{c}
\begin{array}{cccc} LC_0 & LC_1 & \ldots & LC_{N-1} \end{array} \\
\left[\begin{array}{cccc}
a_{0,0} & a_{0,1} & \ldots & a_{0,N-1} \\
a_{1,0} & a_{1,1} & \ldots & a_{1,N-1} \\
a_{2,0} & a_{2,1} & \ldots & a_{2,N-1} \\
\vdots & \vdots & \ddots & \vdots \\
a_{\mathcal{K}-1,0} & a_{\mathcal{K}-1,1} & \ldots & a_{\mathcal{K}-1,N-1}
\end{array}\right]
\end{array},
\end{aligned}
$$

where, $a_{i,j} \in \{0, 1\}$, and $a_{i,j} = 1$ if key k_i is present in cone LC_j, otherwise, $a_{i,j} = 0$.

It is straightforward for the attacker that, if he/she picks cone LC_j and key bit k_i (if its value is still unknown) in this cone, only keys (other than k_i) residing in LC_j could potentially impede the propagation of k_i to the output y_j. This is advantageous to the attacker because the keys outside of cone LC_j would not, by any means, affect the propagation of k_i to y_j. Thus, he/she can safely ignore these keys, and it does not matter whether he/she already has the correct logical values for them or not.

For example, the directed graph representation of locked netlist c432-RN320 with a 32-bit key [245] is shown in Figure 6.6. Output nodes are in red, key registers in green (at the left-most level), key gates in cyan, remaining input (at the left-most level), and gates in blue. The top two logic cones with the fewest keys are LC_{N223} of output $N223$ and LC_{N329} of output $N329$. Logic cone LC_{N223} has only one key (*keyIn_0_4*, with key gate highlighted) (all other nodes and edges are in magenta and light green). Logic cone LC_{N329} is the superset of LC_{N223}, and it contains additional thirteen keys (all other nodes and edges exclusively in LC_{N329} are in purple and orange). With AFIA, the only key in LC_{N223} is determined first, followed by the remaining thirteen keys in LC_{N329}. Because of the only key in LC_{N223}, no fault injection is necessary for this key's propagation to $N223$.

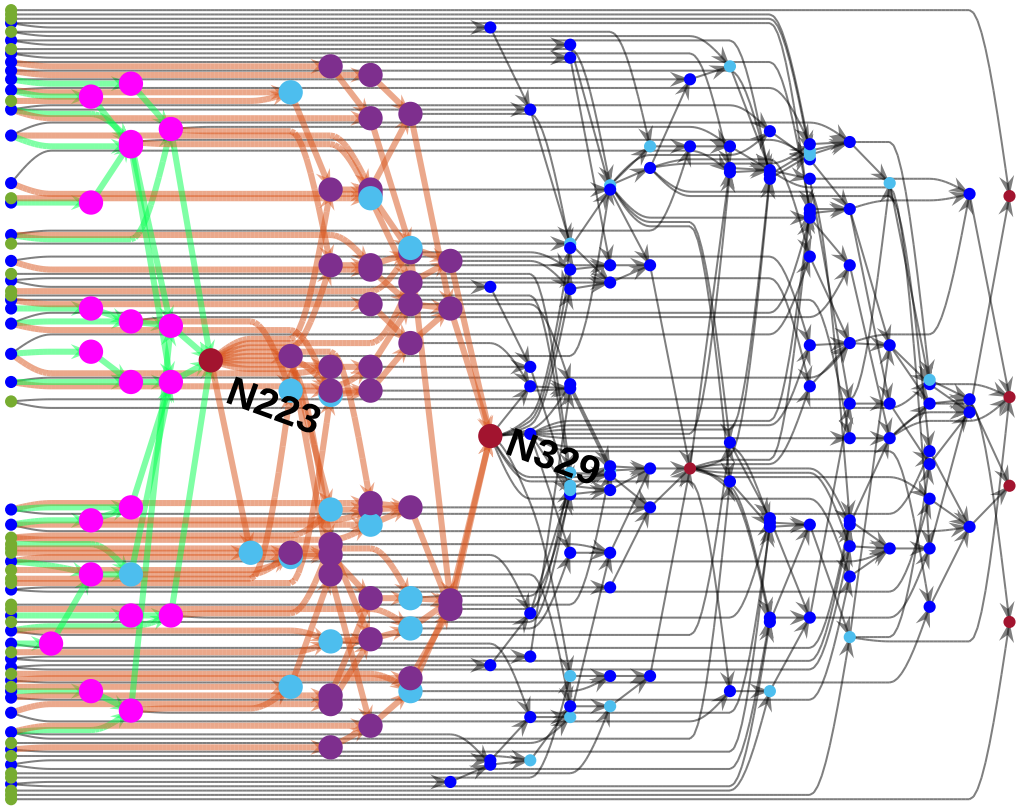

Figure 6.6: Directed graph of locked c432-RN320 netlist with a 32-bit key.

6.3.3 Test Pattern Generation

Once the cone analysis is performed, it is required to generate test patterns so that a targeted key can be sensitized to one of the PO/PPO. The test pattern generation process is similar to the DFA presented in Section 6.2.2 except with a much lesser number of ATPG constraints. We treat undetermined keys as inputs during the test pattern generation and the recovered keys as ATPG constraints. As the secret key remains the same in an unlocked chip, it is unnecessary to inject faults at the recovered key bits as their values are known during the test pattern generation. On the other hand, we need to inject faults at unknown and yet to be determined key lines. However, it is not necessary to inject faults at all of them. We use the ATPG tool to determine whether one or more unknown key bits do not block the propagation of the targeted key bit. As we treat unknown keys as inputs, the ATPG tool can generate a pattern that might contain $X's$ at some of the key lines (using *set_atpg -fill X* [244]), and we do not need to inject faults at these bits. This allows an adversary to reduce the number of fault injections further. Similar to DFA, a stuck-at fault, *sa1* (or *sa0*), is placed on the target key bit with constraints on recovered

key bits during the ATPG. When TetraMAX [244] returns a test pattern, the attacker applies the pattern and injects faults (presented in Section 6.3.4) to sensitize the target key bit at the PO/PPO. After recovering one key bit, AFIA sets ATPG constraints on the recovered key lines, generates another test pattern, and applies it to sensitize the next key.

6.3.4 Fault Injection

The final step applies fault injections on functional chip C_O using generated test patterns from Section 6.3.3. Faults are injected at the key registers with any appropriate fault injection techniques described in Section 6.1.4. No fault injection is necessary at the key bits whose values are already determined as their values are no different from those already programmed in the chip C_O. If we receive a faulty response by applying the test pattern developed in *Step-2*, the value of the secret key will be 1 as we have sensitized a *sa1* fault during the ATPG; otherwise, the secret key is 0. If we generate a test pattern considering a *sa0* fault, the faulty response results in the secret key of 0, and vice versa. *Step-2* in Section 6.3.3 and *Step-3* in Section 6.3.4 are repeated until the entire secret key is found. Consequently, fewer faults are injected compared with the DFA since injections happen only at key locations (of the same logic cone) that block the propagation of the to-be-determined key bits.

6.3.5 Proposed Algorithm for AFIA

Algorithm 3 describes the implementation details of AFIA. The adversary first constructs a directed graph G from the locked netlist C_L (Line 1), as elaborated in Section 6.3.2. Aside from converting netlist to graph, `netlist2Graph(.)` returns the key list K and output list Y. By exploiting directed graph structure, logic cone LC_j can be easily extracted by flipping all edges in graph G (Line 2) and run breadth-first-search (BFS) or depth-first-search, (DFS) [246], on output nodes y_j. The key-cone association matrix *A* is declared as an empty array, where the cone and key information will be added (Line 3). Function `extractCone(.)` is implemented with BFS. It returns the directed subgraph of logic cone LC_j and a logical (**true/false**) vector LK_j of dimension $\mathcal{K} \times 1$. If key bit k_q is inside cone LC_j, $LK_j[q] =$ **true**; else,

Algorithm 3: AFIA: ATPG-guided Fault Injection Attack.

Input : Locked gate-level netlist (C_L)
Output: Secret key (*KEY*)

```
1  // ——————————- Cone Analysis ——————————————
2  [K, Y, G] ← netlist2Graph(C_L) ;
3  Gflip ← flipEdges(G) ;
4  A ← [ ] ;
5  for each y_j in Y do
6  |   [LK_j, LC_j] ← extractCone(Gflip, K, y_j) ;
7  |   A ← append vector LK_j as the last column ;
8  end
   // ——————- ATPG test pattern generation ——————————
9  Recovered key bits from Step-3 of AFIA, K^R ← ∅ ;
10 while (A != false) do
11 |   [K^U_LC] ← fConeWMinKeys(A, K) ;
12 |   if K^U_LC != ∅ then
13 |   |   for l ← 0 to (|K^U_LC| − 1) do
14 |   |   |   Add a sa1 fault at key line K^U_LC[l] ;
15 |   |   |   Add constraints at recovered key bits to K^R;
16 |   |   |   Test pattern P_l ← run ATPG (set_atpg -fill X);
17 |   |   |   Remove all faults ;
18 |   |   |   Remove all constraints ;
   |   |   |   // ——————- Fault Injection ——————————
19 |   |   |   Invoke Step-3 of AFIA with P_l;
20 |   |   |   Add recovered key K^U_LC[l] to K^R ;
21 |   |   |   Assign false to all entries in key K^U_LC[l]'s row in A ;
22 |   |   end
23 |   end
24 end
25 Report the secret key, KEY ← {K, K^R} ;
```

$LK_j[q] =$ **false**. Matrix A is updated by concatenating all vectors LK_j's together (Line 6) so that the complete A has $\mathcal{K}$ rows and N columns, as explained in Section 6.3.2.

AFIA invokes `fConeWMinKeys(.)` (Line10) and obtains a vector K^U_{LC} of all unknown keys in the logic cone with the fewest (positive) unknown keys. For simplicity, K^U_{LC} records the row indices of the unknown keys, as in matrix A. For every key bit in K^U_{LC}, the *sa1* is set on the to-be-determined key (Line 13). The recovered key values in K^R are appended as constraints (Line 14). Test pattern P_l (Line 15) is generated after invoking ATPG. All the stuck-at faults (Line 16) and constraints (Line 17) are removed. When P_l and fault injections (Line 18) are applied on the working chip C_O, $K^U_{LC}[l]$ bit is recovered by referencing the ATPG's

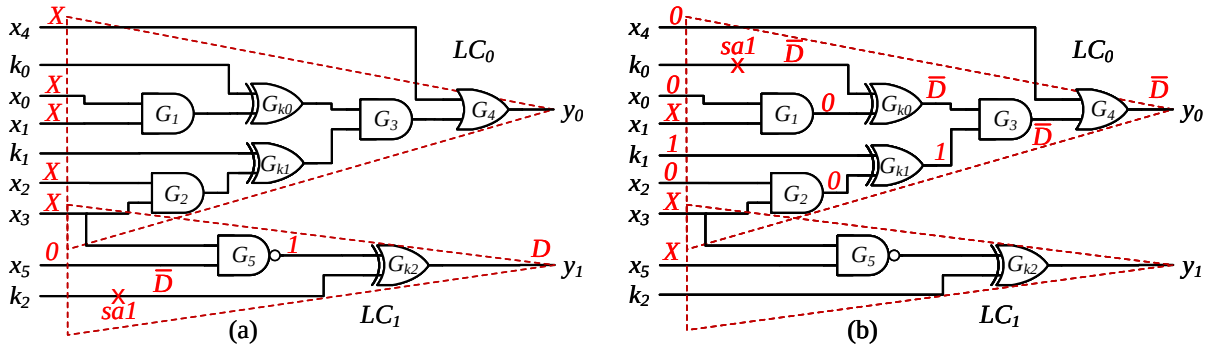

Figure 6.7: Test Pattern Generations for AFIA. (a) Test Pattern $P_0 = [XXXXX0]$ for *sa1* at k_2. (b) Test Pattern $P_1 = [0X0X0X]$ for *sa1* at k_0 with injected fault $k_1 = 1$.

predicted output of the corresponding P_l. Afterward, the correct bit value is added to the recovered key list K^R (Line 19). Since this bit is recovered, it is no longer an unknown key, and AFIA updates the association matrix *A* to assign logical zero to all entries on key $K^U_{LC}[l]$'s row (Line 20). This is conceptually equivalent to deleting $K^U_{LC}[l]$ from the unknown key list as `fConeWMinKeys(.)` will only count the number of non-zero entries per column. When all key bits in K^U_{LC} are determined, the adversary moves on to the subsequent logic cone (Line 10). Finally, when all cones are covered, the secret key *KEY* is returned (Line 24).

6.3.6 Example

Here, we use the same circuit as in Figure 6.4 as an example to illustrate how AFIA works. The circuit has six inputs, two outputs, and three key bits. With two outputs, this circuit has two logic cones, as in Figure 6.7. The same D-Algorithm [77] is applied to show the propagation of stuck-at-faults. Based on cone analysis in Section 6.3.2, logic cone LC_0 contains two key bits, k_0, k_1, cone LC_1 has only one key k_2. Thus, the association matrix *A* can be represented as:

$$A = \begin{array}{c} \\ k_0 \\ k_1 \\ k_2 \end{array} \begin{array}{c} \begin{array}{cc} LC_0 & LC_1 \end{array} \\ \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \end{array}.$$

AFIA picks a logic cone with the fewest number of unknown keys to solve (Line 10, Algorithm 3). Since all keys are unknown at this time, `fConeWMinKeys(.)` function selects

logic cone LC_1 and returns $K_{LC}^U = [2]$. This cone has one key bit k_2, to which we assign *sa1*. Using D-Algorithm, fault value $\overline{D}$ is marked on this key line. Here, the output y_1 is directly connected to XOR key gate G_{k_2}, and we can propagate this fault $\overline{D}$ to output $y_1 = D$ with logic 1 for the other input of this XOR gate, as in Figure 6.7(a). Test pattern $P_0 = [x_0 x_1 \dots x_5] =$ [*XXXXX0*] can detect *sa1* for key k_2. Here, the value of the recovered key is 1 when the output is faulty. Otherwise, the recovered key is 0 as we have sensitized a *sa1* fault during the ATPG. Note that no fault injection is necessary to determine this key. Matrix *A* is updated with all zeros on the k_2's row,

$$A = \begin{array}{c} \\ k_0 \\ k_1 \\ k_2 \end{array} \begin{array}{c} \begin{array}{cc} LC_0 & LC_1 \end{array} \\ \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} \end{array}.$$

In the next iteration (Line 10), there is only one logic cone (also the cone with the least unknown keys), LC_0, left in matrix A that has unknown keys. Function `fConeWMinKeys(.)` identifies LC_0 and yields $K_{LC}^U = [0\ 1]^T$, which captured the indices of unknown keys k_0, k_1. With two keys k_0 and k_1, AFIA chooses k_0 first randomly (Algorithm 3 Line 13). By adding *sa1* at k_0, test pattern $P_1 = [x_0 x_1 \dots x_5] =$ [*0X0X0X*] with logic *1* fault on k_1 can propagate the faulty response $\overline{D}$ in k_0 to y_0, as shown in Figure 6.7(b). Fault injection is performed at k_1 by setting its value to 1, and apply P_1 to determine k_0. AFIA, then, flushes out all the entries on row k_0 of matrix *A*,

$$A = \begin{array}{c} \\ k_0 \\ k_1 \\ k_2 \end{array} \begin{array}{c} \begin{array}{cc} LC_0 & LC_1 \end{array} \\ \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} \end{array}.$$

After k_0 is recovered, AFIA moves on to determining the other key in LC_0, k_1, (Line 12). We add a *sa1* at k_1 (Line 13), along with constraining on k_0, k_2 to their determined values (Line 14). If the correct logical value for k_0 is *0* (i.e., the stored key), test pattern $P_2 = [x_0 x_1 \dots x_5] =$ [*110X0X*] can sensitize the *sa1* of k_1 to the output y_0. If the stored secret

key bit is $k_0 = 1$, the test pattern P_2 will be different, and its value will be $[0X0X0X]$, which one can verify using the same D-Algorithm. Note that no fault injection is necessary to determine k_1.

Finally, the matrix *A* will be updated to all zeros and the AFIA recovers the entire key.

6.3.7 AFIA Complexity Analysis

The average complexity of the AFIA attack is linear with the key size (K). In this section, we show that AFIA is very effective at breaking any logic locking technique. However, the fault injection time may vary depending on the effectiveness of the equipment. It is practically instantaneous to obtain the secret key once the responses are collected from C_O.

Lemma 6.1. *One input pattern is sufficient to recover one key bit.*

Proof. A single test pattern is sufficient to detect a *saf* if such a fault is not redundant [77]. A redundant fault results from a redundant logic that cannot be exercised from the inputs. As the key gates are placed to modify the functionality, it cannot be a redundant logic. As there exists one test pattern to detect a *saf* at the key line, it can be used to recover one key bit. □

Theorem 6.2. *AFIA recovers the entire secret key, K using at most $\mathcal{K}$ number of test patterns, i.e.,,*

$$TP_{AFIA}[f_K(C_L) = f(C_O)] \leq \mathcal{K}. \tag{6.1}$$

where $f_K()$ represents the functionality with K as the key.

Proof. A C_L with a $\mathcal{K}$-bit key is injected with a *saf* fault on every key line. As AFIA requires one test pattern to obtain one key bit (see Lemma 6.1), the upper bound of the number of test patterns is $\mathcal{K}$. However, a single pattern can detect two or more stuck-at faults on the key lines if their effect is visible in different logic cones (*e.g.,*, different outputs). As a result, the required number of test patterns to recover the entire key (K) can be less than $\mathcal{K}$. □

Theorem 6.3. *AFIA is applicable to strong logic locking [98], where pairwise key gates are inserted to block the propagation of one key by the other.*

Proof. In strong logic locking, the propagation of one key is blocked due to the other key. However, $(\mathcal{K}-1)$ faults are injected at $(\mathcal{K}-1)$ key lines, worst-case scenario, except for the one whose value needs to be determined. Once an external fault is injected into the functional chip, the key value is fixed and no longer remains unknown. Hence, AFIA is applicable to strong logic locking. □

Theorem 6.4. *The worst-case complexity for the total number of faults injected in AFIA is* $\boldsymbol{O}(\mathcal{K}^2)$.

Proof. Let us consider a circuit with a single logic cone locked with a secret key vector $\{k_0, \ldots, k_{\mathcal{K}-1}\}$. Suppose all key bits are pairwise non-mutable convergent, i.e., the propagation of one key bit depends on all the other keys. To sensitize the 1^{st} key bit, we need to add $\mathcal{K}-1$ faults during the fault injection process. The 2^{nd} key bit requires $\mathcal{K}-2$ faults as the value of the 1^{st} key bit is known. Similarly, the 3^{rd} key bit requires $\mathcal{K}-3$ faults, and so on. Thus, the total number of faults is:

$$\sum_{i=1}^{\mathcal{K}}(\mathcal{K}-i) = \frac{\mathcal{K}\cdot(\mathcal{K}-1)}{2}.$$

Thus, the worst-case complexity for the total number of faults injected is $\boldsymbol{O}(\mathcal{K}^2)$. □

Theorem 6.5. *The average-case complexity for the total number of faults injected in AFIA is* $\boldsymbol{O}(\mathcal{K})$.

Proof. Consider a circuit with N logic cones, each cone LC_j has negligible or no overlap with its neighboring cones, LC_{j-1} and LC_{j+1}, and $\mathcal{K}$ keys are evenly distributed (amortized) among the N cones. For each cone, it has an average $a = \frac{\mathcal{K}}{N}$ keys). Since negligible overlap between cones, there is no preference between the order of execution on deciphering keys in logic cones, and each cone needs to inject $\frac{\mathcal{K}/N\cdot(\mathcal{K}/N-1)}{2}$ faults. Overall, by summing up all faults for every logic cone, the required number of fault injections is $N \cdot \frac{\mathcal{K}/N\cdot(\mathcal{K}/N-1)}{2}$.

Thus, the average-case complexity is $N \cdot \frac{\mathcal{K}/N\cdot(\mathcal{K}/N-1)}{2} = \frac{a-1}{2} \cdot \mathcal{K} = \boldsymbol{O}(a\mathcal{K}) = \boldsymbol{O}(\mathcal{K})$. □

6.3.8 AFIA on Fault-Tolerant Circuit

Fault-tolerant circuits and circuits with redundancy may prevent the injected faults from being revealed at the output. However, it does not affect our proposed AFIA. As the objective of logic

locking is to produce incorrect output for wrong key combinations under certain input patterns, these input patterns ensure the differential output behavior for keys. Thus, the key cannot be inserted inside the region of redundancy, where no input pattern can ever produce differential output. Any key bit placed at these locations cannot corrupt the output so that either logic 0 or logic 1 is its correct value. The SoC designer would not place a key bit in such a way that both logic values gives the correct output since it contradicts the principle of logic locking. In summary, redundancies are not a countermeasure against AFIA attack for a well-designed locked circuit.

6.3.9 AFIA on Non-Functional-Based Locking Techniques

Our fault injection-based attack can also be extended to non-functional logic locking techniques [119, 131]. The dynamically obfuscated scan-chain (DOSC) technique [131] has three secrets stored in the tamper-proof memory, which are the functional obfuscation key, the LFSR seed, and the control vector. AFIA can break the functional obfuscation key if the obfuscated scan-chain becomes transparent to the attacker. To achieve that, the attacker needs to inject faults at all the *Scan Obfuscation Key* registers directly to get a known shift out state from the functional IP. For the routing-based locking technique [119], our proposed attack is applicable to breaking the key-configurable logarithmic-based network (CLN) as the switch-boxes (SwB) consist of MUX-based key gates. Once a fault is injected into a key register, the selection path for the corresponding MUX is determined. We can target these keys one at a time with test patterns generated from the ATPG tool and inject faults on dependent key registers.

6.4 Experimental Results

This section provides the feasibility of fault injection to break secure logic locking. Extensive simulations are performed on different benchmarks with different locking techniques to demonstrate the effectiveness of the proposed fault injection attack for breaking a secure locking technique. We have shown a significant reduction of total fault count for AFIA compared to DFA, presented in our conference paper, in breaking the same locked benchmark.

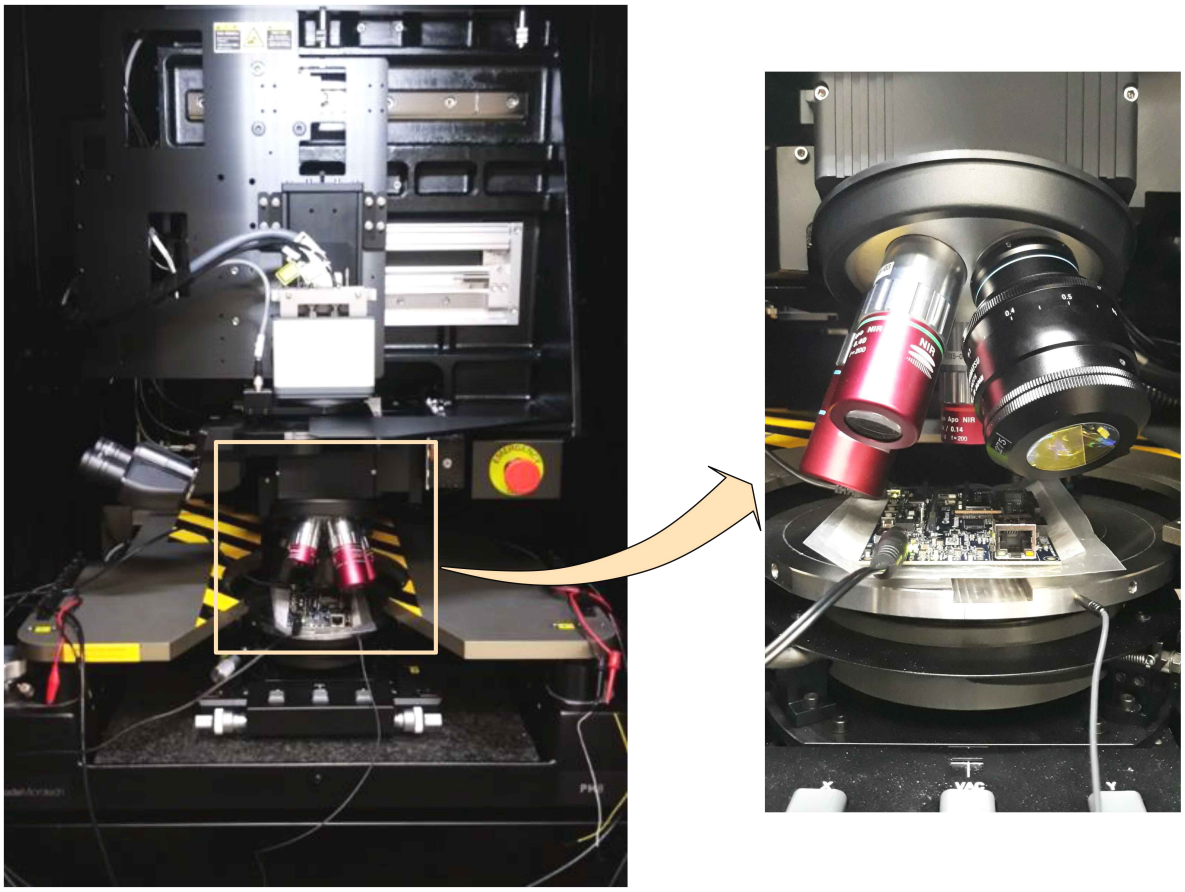

Figure 6.8: The FPGA board placed under the lens for laser-fault injection at the target registers.

6.4.1 Laser Fault Injection

To demonstrate the laser fault injection attack, we selected a Kintex-7 FPGA [247], which is used as the device-under-test (DUT). Locked benchmark circuits are implemented in the Kintex-7 FPGA, where faults are injected into key registers. Figure 8.2 shows the laser fault injection (LFI) setup with a Hamamatsu PHEMOS-1000 FA microscope [248]. The equipment consists of a diode pulse laser source (Hamamatsu C9215-06) with a wavelength of 1064 nm. Three objective lenses were used during this work: 5x/0:14 NA, 20x/0:4 NA, 50x/0:76 NA. The 50x lens is equipped with a correction ring for silicon substrate thickness. The laser diode has two operation modes – a) low power (200 mW) pulse mode, and b) high power (800 mW) impulse mode. The high power impulse mode can be used for laser fault injection. The laser power can be adjusted from 2% to 100% in 0.5% steps.

Photon emission analysis [249] can be used to localize the implemented locked circuitry in the DUT. Thereafter, the DUT is placed under the laser source for LFI. A trigger signal is

fed to the PHEMOS-1000 to synchronize the LFI with the DUT operation. Once the device reaches a stable state after power-on, the laser is triggered on the target key registers. After the fault injection, we need to guarantee that the device is still functioning as expected and has not entered into a completely dysfunctional state. The laser triggering timing can be checked by a digital oscilloscope for greater precision.

6.4.2 Fault Count Comparison

The differential attack methodology (DFA) introduced in Section 6.2 and in [157] requires $\mathcal{K}-1$ number of constraints per test pattern. The total number of faults that need to be injected to determine one key bit is $2\mathcal{K}-1$, as C_A and C_F require $\mathcal{K}-1$ and $\mathcal{K}$ faults, respectively. The total number of faults required to decipher $\mathcal{K}$ key bits is $(2\mathcal{K}-1)\cdot\mathcal{K} = 2\mathcal{K}^2-\mathcal{K}$. Compared to DFA, AFIA only requires injecting faults to key registers if these key bits are interdependent, where the propagation of one key is dependent on others.

Table 6.1 shows the number of faults to be injected for both the DFA (Algorithm 2) and AFIA (Algorithm 3). To demonstrate the feasibility of the fault injection attack on logic locking, we computed the number of faults after generating test patterns using constrained ATPG using the Synopsys TetraMAX tool [244]. Note that the successful generation of test patterns using constrained ATPG guarantees the successful attack on locking. We choose benchmark circuits with random logic locking (added '-RL' after the benchmark name) and strong logic locking (added '-SL') from TrustHub [245], SFLL-hd (added 'SFLL-hd'), SFLL-flex (added 'SFLL-flex'), and SFLL-rem (added 'SFLL_rem') benchmarks from [105], and GitHub [219]. Column 2 represents the secret key size, whereas Columns 3 and 4 represent the number of faults to determine the entire key for DFA and AFIA, respectively. Data in Column 4 is collected under *sa1* fault in test pattern generation (Algorithm 3). Finally, Column 5 shows the average number of faults to evaluate one key bit under AFIA. For example, with locked benchmark c432-RN320, the number of faults required for DFA is 2016, whereas AFIA requires only 48 faults to extract the 32 key bits, leading to 1.5 faults per key bit. For c1355-SL1280, the number of faults increased significantly to 32,640 for DFA. AFIA only requires 1,419 faults to determine the 128 key bits, or 11.09 faults per key bit.

Table 6.1: Comparison of Number of Injected Faults

Locked Benchmark	Key Size ($\mathcal{K}$)	DFA	AFIA	
		F_T	F_T	$F_T/\mathcal{K}$
c432-RN320	32	2016	48	1.5
c432-RN640	64	8128	165	2.58
c432-RN1280	128	32640	1085	8.48
c2670-RN1280	128	32640	520	4.06
c3540-RN1280	128	32640	268	2.09
c5315-RN1280	128	32640	282	2.20
c6288-RN1280	128	32640	268	2.09
c7552-RN1280	128	32640	334	2.61
c1355-SL1280	128	32640	1419	11.09
c1908-SL1280	128	32640	654	5.11
c5315-SL1280	128	32640	3469	27.10
c6288-SL1280	128	32640	368	2.88
c7552-SL1280	128	32640	188	1.47
b14_C_k8_SFLL-hd	8	120	28	3.5
b14_C_k16_SFLL-flex	16	496	120	7.5
b14_C_k32_SFLL-flex	32	2016	496	15.5
b14_C_k64_SFLL-flex	64	8128	2016	31.5
b14_C_k128_SFLL-flex	128	32640	8128	63.5
c432_k8_SFLL-hd	8	120	28	3.5
c432_k16_SFLL-flex	16	496	120	7.5
c432_k32_SFLL-flex	32	2016	496	15.5
c880_k8_SFLL-hd	8	120	28	3.5
c880_k16_SFLL-flex	16	496	120	7.5
c880_k32_SFLL-flex	32	2016	496	15.5
SFLL_rem_k128 [219]	128	32640	8128	63.5

Based on Theorem 6.5, if keys are uniformly distributed among logic cones, the number of fault injections for AFIA is linear with respect to key size, $\boldsymbol{O}(a\mathcal{K}) = \boldsymbol{O}(\mathcal{K})$, with variable a indicating the average key size per logic cone. If having the same key size, an RLL circuit with more logic cones, or a smaller a, (provided that the size of all logic cones are about the same), should, generally, has fewer fault injections than one with fewer logic cones. This is equivalent to having fewer injected faults in an RLL-based circuit that contains more output than the ones without (see definition of the number of logic cones in 6.3.2). Benchmark c432-RN1280 has a larger a than other 128-bit RLL circuits, for c432 has only seven outputs, while c2670 has 140 outputs, c3540 has 22, c5315 has 123, c6288 has 32, c7552 has 108 outputs respectively. (Note, not all logic cones will have keys inside, but the circuit with more output usually has

more key-embedded cones than those with fewer outputs.) This is the reason that c432-RN1280 requires considerably more fault injections in total, 1085, than other locked netlist with same key size, where c2670-RN1280 needs 520 faults, c3540-RN1280 has 268, c5315-RN1280 has 282, c6288-RN1280 has 268, c7552-RN1280 has 334, see Table 6.1.

RLL randomly picks a location in the original unlocked circuit for key gate insertion, while SLL produces more blocking keys. In terms of theoretical complexity analysis, as long as the key gates in RLL locked circuit are distributed uniformly, the number of fault injections for SLL should be larger than RLL, under the same original unlocked benchmark and the same key size, *e.g.,*, c5315-RN1280 and c5315-SL1280, c6288-RN1280 and c6288-SL1280. For SFLL-hd and SFLL-flex, each locked circuit has a perturbation unit and a restoration unit. All keys reside in the functionality restoration unit, where every key passes through the output of the restoration subcircuit to reach the primary output [161, 207]. Because of this restoration unit, all key bits are interdependent. Hence, all SFLL-flex and SFLL-hd circuits belong to the worst-case scenario as in Theorem 6.4, in which the number of injected faults is $\frac{\mathcal{K}\cdot(\mathcal{K}-1)}{2}$. We also evaluated our proposed attack on the latest SFLL variant, SFLL-rem [106, 250]. Although SFLL-rem does not have the added perturb unit, the keys are present in the restoration unit only, and our attack can still break it.

6.5 Summary

This chapter presents AFIA, a novel stuck-at fault-based fault injection attack that undermines the security of any logic locking technique. AFIA utilizes cone analysis to analyze the dependency of keys. Faults are injected only at the interdependent key bits, which is a significant improvement from the previously published attack DFA [157], dropping the total number of faults to the linear multiple of key size. With the automatic test pattern generation (ATPG) tool, we constructed a pattern set, which is used to apply to an unlocked chip. Each pattern is sufficient to determine a one-bit key. All key bits are derived by comparing collected responses from fault injections and the predicted response from test pattern generation. We performed

laser fault injections on Kintex-7 FPGA with various locked benchmark circuits and state-of-the-art locking techniques, and our results have demonstrated the effectiveness of the proposed AFIA scheme. Our future work will focus on developing a locking technique to prevent AFIA.

Part III

Hardware-Based Solutions

Chapter 7

A Comprehensive Test Pattern Generation Approach Exploiting the SAT Attack for Logic Locking

The exponential growth of integrated circuits (ICs) in our critical infrastructure requires aggressive testing as system failure has severe safety consequences. As a result, it is critical that the escape of manufacturing defects to the next stage approaches zero. For example, multiple safety standards, like AEC-Q100 and ISO 26262 [251], are defined to meet the zero-defective-parts-per-million goal for safety-critical automotive chips. Testing plays a vital role in detecting all defects in the manufactured chips to avoid the potentially devastating effects of defective ones slipping from the testing facility. Today's commercial automatic test pattern generation (ATPG) can generate test patterns for stuck-at, delay, bridging, and a few other fault models [244]. However, achieving a fault coverage that leads to zero-defect escape is still an open problem. For example, it is challenging to reach 100% stuck-at fault coverage using commercial ATPG tools. It can be extremely difficult to sensitize a hard-to-detect fault and propagate the faulty response at the outputs of a large circuit, limiting the desired goal of achieving perfect fault coverage. It is also challenging to identify all the redundant faults as their effects cannot be propagated to the output. These faults can be ignored for determining a meaningful fault coverage, as they do not impact the function of a circuit.

Over the past few decades, we have seen a steady increase in the fault coverage for digital circuits using the continued advancement in combinational ATPG techniques, from Roth's D-Algorithm [252, 253] to PODEM [254], FAN [255], SOCRATES [256], TRAN [257], *etc*. Along with these techniques, SAT-based test pattern generation has also been proposed as a

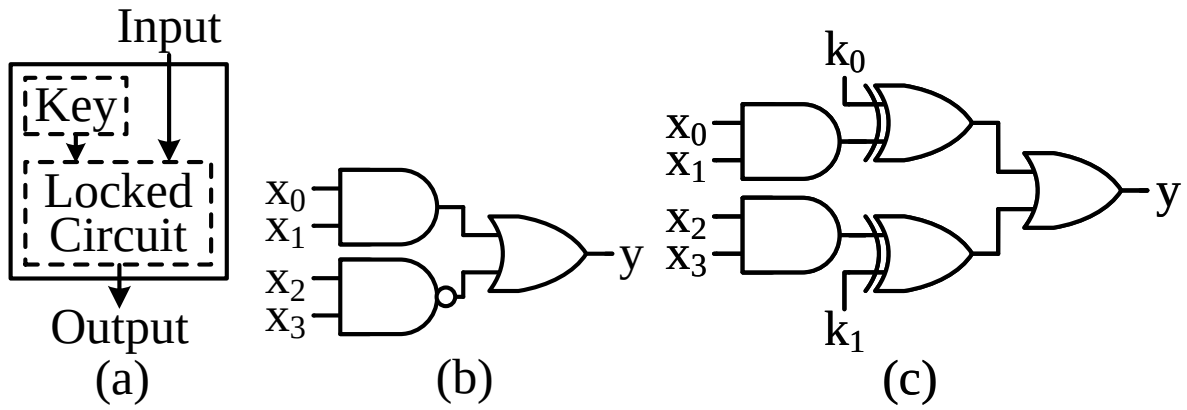

Figure 7.1: Logic Locking. (a) Overview of logic locking (b) Original circuit. (c) XOR-based locking with $\{k_0 k_1\} = \{01\}$.

solution to achieve higher fault coverage [258–265]. For SAT-based techniques, the miter construction between the fault-free circuit and faulty circuit with a stuck-at fault (*saf*) is the core for generating a test pattern for detecting that *saf*. The increased number of conflicts for larger circuits resulting from the miter circuits and getting resolved at a later stage makes hard-to-detect faults undetectable. As these SAT-based prior works were concluded nearly a decade ago, it is fair to assume that the Industry has already assimilated the state-of-the-art research. However, we still observe several hard-to-detect and redundant faults that a commercial ATPG tool, *e.g.,* Synopsys TetraMAX II [244] fails to identify even with the maximum abort limit (see Section 7.3). Some undetected faults are redundant faults in the circuit, where no pattern could propagate the faulty effect to the primary output. Others are the hard-to-detect faults, where the ATPG tools fall short in finding the appropriate test patterns even if such tests exist to detect these faults. Therefore, the main bottleneck from reaching high fault coverage for IC testing is in the undetected faults, specifically, the classification of redundant faults and test pattern generation for hard-to-detect faults. *The focus of this paper is to analyze and classify these undetected faults, not identified by commercial ATPG tools, so that (i) we can accurately distinguish any redundant faults from non-redundant ones; (ii) generate the test patterns for each hard-to-detect faults; and (iii) generate tests for combining multiple hard-to-detect faults to reduce the total pattern count.*

The hardware security community has been actively involved in solving the threat of intellectual property (IP) piracy [15, 21, 22, 92, 266] and IC overproduction [23, 84–89], originating from the horizontal integration of semiconductor design, manufacturing, and test. Due to the

increased chip design complexity and manufacturing processes, it is practically infeasible for many design houses to manufacture chips on their own. An untrusted entity in the semiconductor supply chain can pirate the design details and cause irreparable damage. Logic locking [23, 84, 89, 98, 99] was proposed to counter IP piracy, where a circuit design is obfuscated using a secret key. Figure 7.1 shows an abstract representation of logic locking with a simple example. The secret key (K) is programmed into the tamper-proof memory, as shown in Figure 7.1(a). Figure 7.1(b) shows an example of the original netlist with function $y = x_0x_1+x_2x_3$. A lock is inserted using two XOR gates with key $\{k_0k_1\}$, as shown in Figure 7.1(c). This modifies the original functionality y to $y' = (x_0x_1 \oplus k_0) + (\overline{x_2x_3} \oplus k_1)$. The secret key value $\{k_0k_1\} = \{01\}$ maps y' to y for all possible input combinations. The security of any locking scheme relies on the secrecy of the key. The original netlist can be recovered if an adversary obtains the correct key values. Subramanyan et al. showed that Boolean Satisfiability (SAT) could be used to break traditional locking schemes effectively [100]. The attack constructs a miter circuit and asks SAT solver to find an input pattern that produces differentiating output behavior between incorrect keys and the right one, similar to revealing the faulty state to the output [7, 99, 157]. The attack is very effective in determining the key (i.e., the value of k, the key-input of XOR shown in Figure 7.1(c)) no matter where the key gate (XOR) is placed in the netlist. This motivates us to develop a novel test pattern generation scheme using this powerful SAT attack. The question is, can we model a stuck-at fault to its key-dependent locked circuit counterpart so that the SAT attack can find a test pattern to determine the key, and, thus, a test for the same stuck-at fault?

The chapter is organized as follows. We begin with a brief introduction to test pattern generation and the SAT attack on logic locking in Section 7.1. Our proposed approach to increase fault coverage is presented in Section 7.2. The result and analysis for the proposed approach are described in Section 7.3. Finally, we conclude this chapter in Section 7.4.

In parallel, the research community explored SAT-based test pattern generation. The initial SAT-based technique, proposed by Larrabee [258], constructs the Boolean difference between the faulty and fault-free circuits to detect single stuck-at faults. Stephan et al. proposed TEGUS that uses gate characteristic functions added in depth-first search order from inputs to outputs [259]. Over the years, different SAT-based test pattern generation techniques have been proposed to obtain a high fault coverage [260–265]. Eggersgluss et al. explored SAT-based test compaction with a large number of unspecified bits [260–262]. Drechsler et al. includes the modeling of tristate elements with additional unknown (U) and high-impedance (Z) states [263]. Balcarek et al. [265] filters the unexcitable faults based on the static and dynamic implications. Fujita et al. [264] targets test pattern generation with multiple faults. However, to find a test pattern for a hard-to-detect fault, the SAT solver encounters a large number of conflicting assignments and requires an increased number of backtracks, which makes test generation time excessively high.

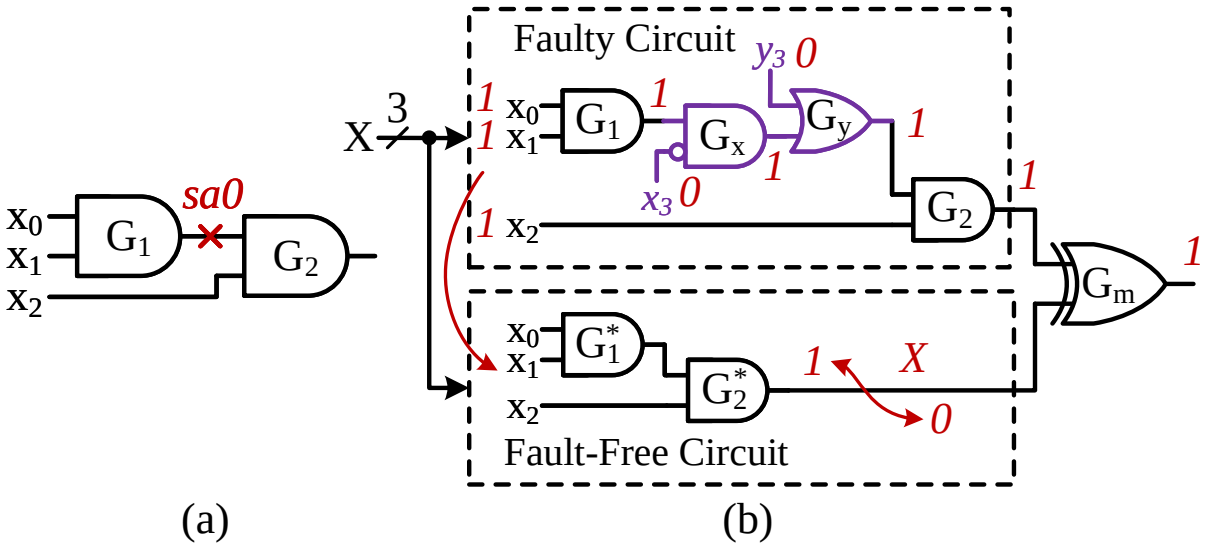

Figure 7.2: Conflicts in solving miter circuit presented in Fujita et al.. (a) A simple circuit with a *sa0* fault. (b) Conflict during SAT assignment.

Fujita et al. [264] constructs the miter with a faulty circuit with modeled faults and fault-free circuits. Figure 7.2(a) shows a simple circuit with a *sa0* fault. Figure 7.2(b) shows the miter circuit used for test generation [264]. As the OR gate for modeling *sa1* should remain

ineffective as we are targeting *sa0*, one can simply ignore it by having its input y_3 tied to 0. Without loss of generality, we assume the SAT solver first makes a decision to the upper input node (corresponding to the output of the faulty circuit) of miter XOR gate G_m with logic 1. With unit clause propagation, the output of the faulty circuit is 1, and all nodes inside the faulty circuit can be uniquely determined. Conflict arises at the fault-free circuit's Boolean assignment as the input $\{x_0, x_1, x_2\} = \{111\}$ derived from the faulty circuit cannot satisfy the required logic 0 output for the fault-free circuit. Therefore, the SAT solver is required to perform backtracks and resolves the conflict with logic 0 decision for the faulty circuit response and 1 for fault-free. In addition, if a conflict arises, it can be determined after logic assignments for all the nodes. This motivates us to construct a miter that reduces the possible backtracks or resolves conflicts at an earlier stage to ensure manageable test generation time.

7.1.1 SAT attack on Logic Locking

Over the years, the optimization and advancement of SAT algorithms have led to a significant decrease in average runtime for SAT solvers [267]. This leads to a growing number of SAT-based applications. One of the most prominent attacks to counter logic locking, proposed by Subramanyan et al. [100], invokes SAT solver [268] to trim keyspace efficiently and derive the correct key. Although various logic locking approaches [23, 84, 89, 98] have been proposed to obfuscate the original circuitry, SAT attack breaks all of them effectively. Furthermore, the SAT attack is also the backbone of the subsequent logic locking attacks [6, 152, 153, 159]. Unlike structural attack [269] that exploits logic redundancy to recover the secret key partially, the SAT attack relies on finding the DIP, which produces differential output for circuits with incorrect keys, analogous to test patterns that differentiate the faulty and fault-free circuits. This oracle-guided attack receives two circuits as its input, the original circuit, $C_O(X, Y)$, and its locked version, $C(X, K, Y)$. The correct key K_c unlocks the circuit so that it behaves identically to the oracle, $C(X, K_c, Y) = C_O(X, Y)$, but the circuit with an incorrect key would lead to one or more output bits mismatch under certain input vectors. This discrepancy in output response, compared with the oracle, is exploited by the SAT attack. The SAT attack works in two steps, the initialization and the iterative process of pruning the key space.

Algorithm 4: SAT attack on logic locking [100].

Input : Unlocked circuit, oracle ($C_O(X, Y)$) and locked circuit ($C(X, K, Y)$)
Output: Correct Key (K_c)

1 $i \leftarrow 1$;
2 $F \leftarrow C(X, K_{A_1}, Y_{A_1}) \wedge C(X, K_{B_1}, Y_{B_1})$;
3 $[X_i, K_i, f] = \texttt{sat}[F \wedge (Y_{A_i} \neq Y_{B_i})]$;
4 **while** *(*$f == \texttt{true}$*)* **do**
5 $Y_i = \texttt{sim\_eval}(X_i)$;
6 $F \leftarrow F \wedge C(X_i, K_{A_i}, Y_i) \wedge C(X_i, K_{B_i}, Y_i)$;
7 $[X_{i+1}, K_{i+1}, f] = \texttt{sat}[F \wedge (Y_{A_{i+1}} \neq Y_{B_{i+1}})]$;
8 $i \leftarrow i + 1$;
9 **end**
10 $K_c \leftarrow K_i$;
11 return K_c ;

Initialization

It first constructs the miter circuit, where the locked circuit is replicated twice, $C(X, K_A, Y_A)$ and $C(X, K_B, Y_B)$, Algorithm 4, Line 2. The two circuits share input X but not the keys K_A, K_B. Any output mismatch between the two circuits can be easily identified. In the miter circuit, the corresponding output bits from Y_A and Y_B are XORed and then ORed together so that a logic one at the final output indicates the output disagreement between Y_A and Y_B while a logic zero does not.

Pruning of key space

The attack iteratively removes the equivalence classes of incorrect keys. Since the main focus of SAT attack is the use of SAT solver to generate the appropriate input vectors, we denote the i^{th} query of the SAT solver (abstracted as a function $\texttt{sat}[\cdot]$) as the i^{th} iteration of the SAT attack. At i^{th} round, it finds a distinguishing input pattern X_i along with assigning $f == \texttt{true}$, where at least one output bit diverges between $C(X_i, K_{A_i}, Y_{A_i})$ and $C(X_i, K_{B_i}, Y_{B_i})$, $Y_{A_i} \neq Y_{B_i}$, Line 3, Line 7. The actual output Y_i for this distinguishing input X_i is obtained from oracle simulation, $C_O(X_i, Y_i)$, Line 5. Both X_i and Y_i are stored in the solver assumptions, Line 6, and carried to the subsequent iterations. By appending this input-output pair $\{X_i, Y_i\}$ to the conjunctive normal form (CNF) in F, it facilitates the removal of any incorrect key combination that produces output other than the correct one Y_i. The input-output pairs are

accumulated so that, at the subsequent iteration, the distinguishing input pattern that the SAT solver finds not only differential output for the miter circuit but also satisfies all the constraint pairs $\bigwedge_{i=1,2...}\{C(X_i, K_{A_i}, Y_i) \wedge C(X_i, K_{B_i}, Y_i)\}$ of the previous findings. Note that the SAT attack initializes key $K_{A_{i+1}}$ with logic values consistent with these learned IO pairs from the previous iterations. The decisions the SAT attack makes depend on the solver seeds. The SAT attack continues to eliminate the incorrect key classes and shrinks keyspace until no more distinguishing input patterns can be found, then assigns $f == \texttt{false}$. This implies that no more incorrect keys remain. It may occur to some circuits, though rare, that more than one key is left in the key space when distinguishing input patterns no longer exists to differentiate these keys. These keys are in the equivalence class of the correct key since none would produce an output that diverges from the oracle's output. The SAT solver returns the key assignment of the last iteration as the correct key K_c. The detailed attack is shown in Algorithm 4. It is worth noting that, for every locked circuit, the very last iteration of the SAT attack always produces a UNSAT result ($f == \texttt{false}$) where the SAT solver has exhausted all distinguishing input patterns.

7.2 Proposed SAT-based Test Generation Approach

The test pattern returned by ATPG for detecting a *sa1* (or *sa0*) fault for a given node will yield one or more output differences for the faulty circuit against the fault-free one. In particular, the ATPG tool controls the faulty line with the opposite fault value and generates a test pattern where the faulty response is visible at the output. However, the ATPG tool may fail to find the appropriate input pattern during test pattern generation due to the complexity of making fault observable, like the D-Algorithm's fault activation, fault propagation, and line justification. We can broadly categorize faults as redundant and non-redundant. If the fault is redundant, no test pattern can detect it since the faulty logic does not affect the circuit's functionality. If the fault is not redundant, an input pattern must exist to propagate the fault to the output. Although ATPG may not successfully deduce a test pattern, it does not necessarily say that the fault is redundant. The fault could still belong to either group, redundant or non-redundant. The focus

of this section is to generate test patterns for non-redundant faults and, at the same time, separate the redundant faults. *In particular, this section presents how to precisely label a fault as redundant or not when ATPG fails to give the test pattern for an undetected fault or determines the appropriate test pattern in concurrence with the identification of a hard-to-detect fault.* We introduce a novel approach to construct an equivalence mapping between test pattern generation of stuck-at faults and the SAT attack on logic locking. Our fault modeling inserts key gates at the faulty lines so that both fault observability and controllability are fulfilled when the SAT attack tries to find distinguishing patterns to decrypt the key bits. For redundant faults, our model returns UNSAT at the first iteration of the SAT attack without any distinguishing input pattern, indicating that no pattern could make the fault observable. For any hard-to-detect fault, the SAT attack obtains a satisfiable input assignment at the first round, which is the desired test vector. Our approach offers a solution for the test pattern generation problem of hard-to-detect faults, in a novel perspective from logic locking, where any patterns derived from the SAT attack are the ones we need in the test pattern generation domain.

7.2.1 Novel miter construction for stuck-at fault with key-dependent circuit in logic locking

The SAT attack on logic locking has shown tremendous success in deriving the correct key of various locking in a few seconds [100]. As described in Section 7.1.1, this means that the SAT attack found the input patterns necessary for removing all incorrect key combinations within the recorded time frame. For example, the locked benchmark of *c880* with the 192-bit key from random logic locking (randomly inserts XOR/XNOR key gates) is broken by the SAT attack using only 30 distinguishing input patterns in less than 1 second. The efficiency of SAT attack, in terms of both attack time and the number of input patterns, motivates us to exploit it to identify any non-redundant faults and generate the associated test patterns for these hard-to-detect faults that were not previously detected by a commercial ATPG tool. Moreover, it is also desired if we can simultaneously determine any undetected faults that are redundant. To equivalently transform a circuit with stuck-at faults to a locked one, we need to make sure that the properties of these faults are controllable and observable when the SAT attack derives

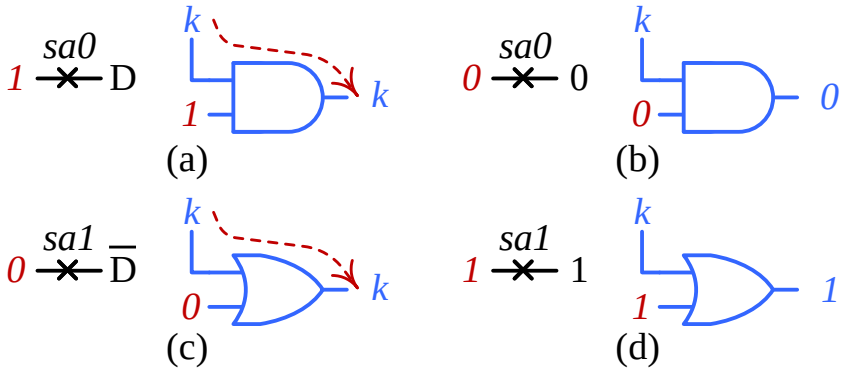

Figure 7.3: Logic locking-based modeling of a *saf* with AND or OR key gate. Converting a *sa0* to AND key gate, (a) successful propagation of key k with logic 1, (b) failed propagation of k with logic 0; a *sa1* to OR key gate, (c) successful propagation of k with logic 0, and (d) failed propagation of k with logic 1.

distinguishing input patterns. This section presents the application of novel miter construction in the SAT attack for test generation of stuck-at faults.

As logic locking uses XOR key gates, the question that first comes to mind is whether it is possible to model a stuck-at fault using an XOR gate. Unfortunately, we cannot model a stuck-at fault due to the symmetric nature of the XOR gate. If we model a stuck-at fault with an XOR gate as a key, either a logic 0 or 1 at the input can propagate the key to its output. However, during ATPG of a stuck-at 0 *sa0*, or stuck-at 1 (*sa1*), a logic 1 or 0, should be placed on the fault site to activate it. This made it impossible for XOR-based locking to model *saf* as both patterns are valid DIPs for the key bit.

As XOR/XNOR key gates can not be applied inside the miter construction in generating a test for a stuck-at fault, *we need to find a different key gate so that its input can only have the opposite value of the stuck-at fault for key propagation.* Logic 1 complements *sa0* fault for making it observable, while logic 0 does not change the functionality. If we pick AND gate, instead of XOR, as the key gate, logic 1 at the input of the key gate will help k to the key gate's output, as shown in Figure 7.3(a), but a logic 0 at the input blocks the key propagation with a constant 0 at the output, as shown in Figure 7.3(b). This means that, if a DIP exists, the SAT attack will assign logic 1 to the input of the key gate for the miter circuit since logic 0 could not fulfill the differential output condition between the incorrect and correct keys. The input

vector used for key derivation in the SAT attack satisfies the controllability and observability requirement of *sa0* in test pattern generation.

For *sa1* fault, we need to assign logic 0 at the fault site for observing the *sa1* since having the same logic as the fault, logic 1, impedes it from being revealed. Analogous to selecting AND key gate for *sa0* test pattern generation, we choose OR gate to represent *sa1*. A logic 0 at the input of the OR key gate allows k to appear at the key gate's output, as shown in Figure 7.3(c). However, placing a logic 1 at the input blocks the key visibility with a constant 1 at the output, as shown in Figure 7.3(d). Similar to the analysis on AND key gate and *sa0*, for generating DIPs, the SAT attack must assign logic 0 to the input of the key gate because logic 1 fails to differentiate the circuit's output between the incorrect and the correct key bit.

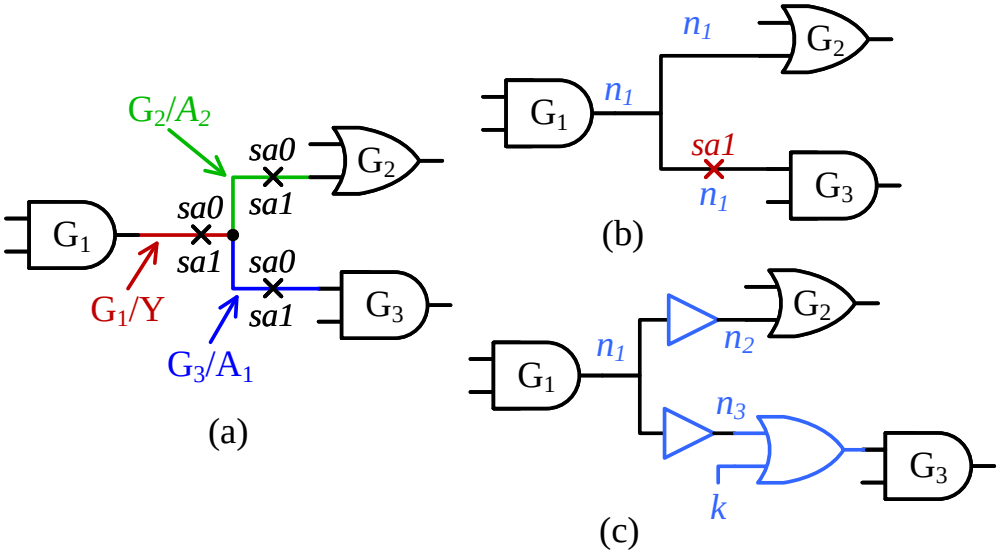

Figure 7.4: Stuck-at-faults in the presence of multiple fanout branches. (a) Fanout branch naming with TetraMAX. (b) Naming convention used in *.bench* file. (c) Fanout branch renaming using buffers.

To address the stuck-at fault detection at fanouts, we need to rename the fanout segment in the *.bench* file used by the SAT attack. Note that any synthesized netlist from a commercial tool considers each fanout segment with a unique name, which is tied to either the input or the output of a gate. Figure 7.4(a) shows the fanout where the output of gate G_1 is connected to both the inputs of G_2 and G_3. TetraMAX inserts faults, both *sa1* and *sa0*, at all the fanout segments, named as $G_1/Y, G_2/A_2$, and G_3/A_1, respectively. However, as these three segments

share the same logic value, in the *.bench* file, all these segments will be treated as a single node, say n_1. As a result, we cannot add faults to the green or blue segments only. However, the SAT attack requires the bench file as input, all the fanout branches have the same name, as shown in Figure 7.4(b). So, we can only add two faults instead of six. To address this problem, we added one buffer to rename the fanout branches. Figure 7.4(c) shows the equivalent locked circuit of a *sa1* fault at the first input of gate G_3.

Based on the above analysis, any *sa0* or *sa1* can be converted to its AND key gate or OR key gate equivalent in logic locking while preserving the observability of stuck-at fault. Note that, aside from generating DIPs, the SAT attack also derives the key value. From the logic locking perspective, the correct key decrypts the locked circuit so that it is functionally the same as the oracle, $C(X, K_c, Y) = C_O(X, Y)$. For the AND key gate, the logical value on the wire (before locking) can pass through the key gate unmodified with key $k = 1$, but assigning key $k = 0$ forces the AND output to constant 0, which alters the original circuit functionality. Likewise, with OR as the key gate, the correct key is $k = 0$, while the output of the key gate will be kept at constant 1 for the incorrect key value $k = 1$. Hence, in addition to DIPs, our equivalent representation of stuck-at faults can be further confirmed by checking the correct key $k = 1$ for all *sa0* and $k = 0$ for *sa1* faults.

7.2.2 Identification of Redundant Faults

Our proposed miter construction with the SAT attack for test generation can also identify any redundant faults. If a stuck-at fault (either *sa0* or *sa1*) is redundant, no pattern can ever propagate this fault since it is not influencing the circuit's functionality. The output behaves the same for the faulty and fault-free circuits. In the same way, when we turn the redundant fault to its equivalent locked circuit, the key cannot be observed from the output as well, as it is located at the redundant line without affecting the primary output.

Let us assume that the i^{th} stuck-at fault in the circuit is redundant. Once we lock it with the appropriate key gate (and buffer if needed) with 1-bit key k_i, we invoke the SAT attack in an attempt to find a DIP for this fault. Since the SAT attack constructs the miter circuit to search for the DIP, as illustrated in Figure 7.5, both locked circuits share the common *m*-bit

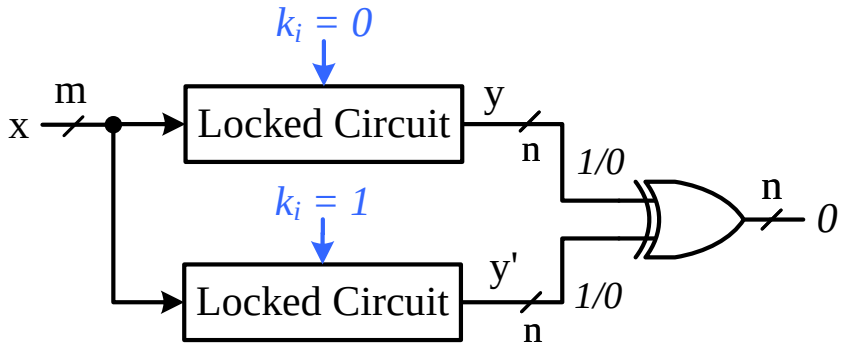

Figure 7.5: SAT attack's miter circuit with 1-bit key k_i for i^{th} *saf* (redundant fault).

input x. The n-bit outputs, y and y', are XORed in a bit-by-bit manner. As the i^{th} fault is redundant, the key bit k_i has no impact on the n-bit output y, and the two locked circuits have an identical response for $k_i = 0$ and $k_i = 1$. For test pattern generation, the miter circuit would produce the exact same output $y = y'$ under any input combinations. As a result, the miter output is always zero, and no output difference between y and y' can be observed. This means that the SAT attack could not find any DIP to differentiate $k_i = 0$ and $k_i = 1$, and it would reach the UNSAT conclusion at the first query of SAT solver on Line 3, Algorithm 4. Note that, *as redundant faults do not change the circuit's functionality, we can ignore them during fault coverage computation if identified correctly.*

7.2.3 ATPG using the SAT Attack on Logic Locking

Just as a few test patterns from the ATPG tool could expose multiple stuck-at faults, the SAT attack can also rule out the exponential number of incorrect keys with a few distinguishing patterns. When it comes to test pattern generation for hard-to-detect faults, we have the option to select how many of these faults we can analyze together. The conservative approach is to generate a test pattern for every fault. This approach can also identify whether a fault is redundant or not by checking if the SAT attack returns a DIP. On the other hand, we can combine the equivalent conversion of multiple faults in one locked circuit with the same number of key gates as the faults. We then ask the SAT attack to break this locked circuit and collect all the DIPs. As the SAT attack generally trims multiple incorrect keys from the search space with only a few DIPs, analyzing a group of faults has the potential of reduced pattern set than inspecting one fault at a time. Both strategies work for any stuck-at fault, regardless of

being redundant or not. The following sections present a comprehensive discussion of both approaches by focusing on undetected faults. The first approach asks the SAT attack for a DIP on every undetected fault, while the second one targets a group of faults so that the SAT attack solves key bits simultaneously.

Approach 1 – Generate One Test Pattern per Fault

Approach 1 focuses on finding a single test pattern for an undetected stuck-at fault using the SAT attack. The equivalent locked circuit contains a 1-bit key. To solve the 1-bit key, the SAT attack only needs to query the SAT solver twice. At the first query, Line 3 of Algorithm 4, the SAT solver returns the input pattern where the primary output differs for the correct and incorrect key assignments, that is, between logic 0 and logic 1. This input pattern, along with the corresponding output, simulated from the oracle, is saved in the IO constraints F. Note that the wrong key bit is implicitly removed from the search space as it does not satisfy the IO pair stored in F. Only the correct key bit matches the IO behavior in F, and it is the only candidate that remains in the search space. With constraint, F appended in the satisfiability of the miter circuit (Algorithm 4, Line 6), the SAT solver must return UNSAT at the second query, and it could not produce any differential output when no more incorrect keys exist in the key space. The second scenario is that the SAT attack could not find any distinguishing input pattern to differentiate the keys in the search space at the first query of SAT solver, and it terminates the while loop (Algorithm 4, Lines 4-9). It also returns the hypothesis key K_i, but it may not align with the correct key value discussed in the novel miter construction for stuck-at faults, Section 7.2.1. This is caused by the fault at the redundant line where faulty value can never reach the output ports, and no input pattern can be found. By including one fault at a time, we can quickly identify which fault is redundant by determining whether DIP is obtained from the SAT attack.

Compared to [264], our proposed approach can determine the test patterns without conflicts or resolve conflicts at an earlier stage due to the initialization of keys in one locked circuit inside the miter. Figure 7.6(a) shows the example circuit with a *sa0* fault. Note that two possible scenarios exist for our proposed approach where the SAT attack assigns k_1 to logic 1 or

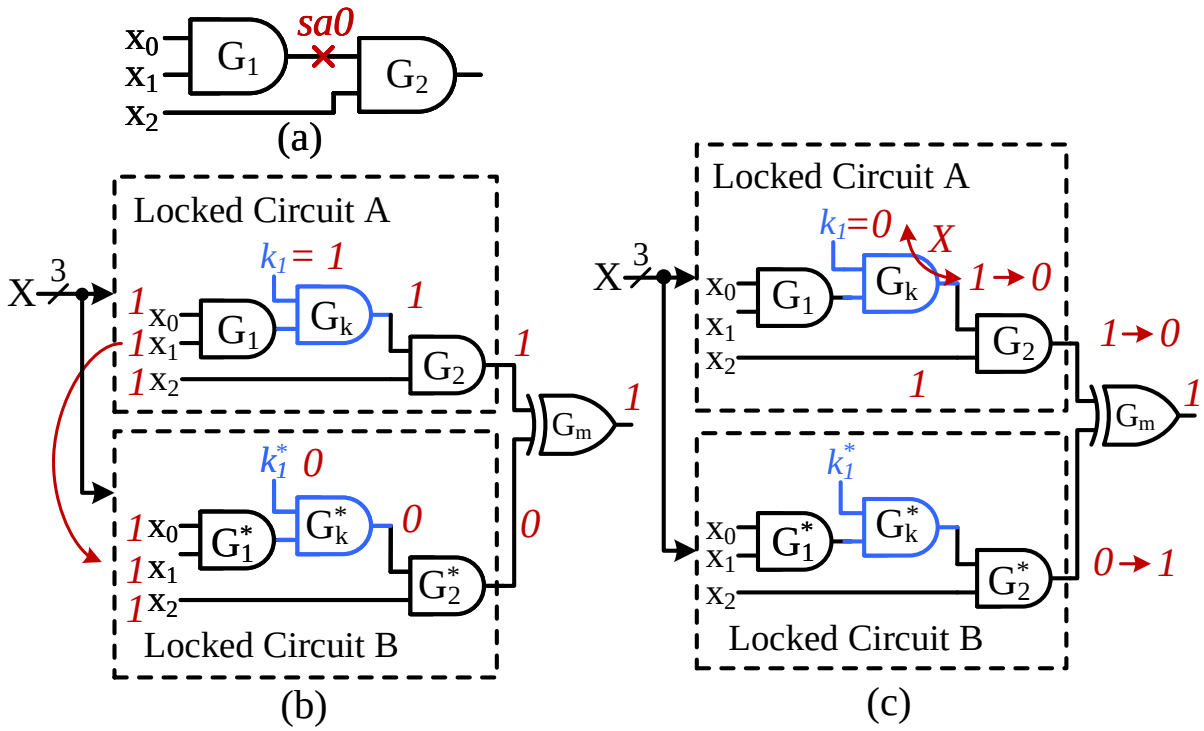

Figure 7.6: The proposed test pattern generation with the SAT attack miter. (a) A simple circuit with a *sa0* fault. (b) No backtrack in deriving the satisfiable assignment with $k_1 = 1$. (c) Backtrack at an earlier stage with $k_1 = 0$.

0 at the start. Let us first consider when the SAT attack assigns $k_1 = 1$, denotes as *Case 1*) and is shown in Figure 7.6(b). Applying the same assumption mentioned in Section 7.1, the SAT solver assigns the first input of G_m to logic 1. All the literals in the miter can be iteratively implied without raising conflict or backtracking for CNF clauses, and a test is found. If the SAT attack starts with $k_1 = 0$, denoted as *Case 2* and shown in Figure 7.6(c), a conflict arises at the output of key gate G_k and is resolved locally by conflict-driven clause learning (CDCL) [270] without having to trace back the entire miter circuit like [264]. As a result, the output of circuit A is reassigned to 0 at an earlier stage [264]. This ensures the SAT solver backtrack at a much earlier stage to determine the satisfiability, and a hard-to-detect fault can be found efficiently.

Approach 2 – Generate Test Patterns for a Group of Faults

While the first approach details the test pattern generation considering a single fault, this Approach 2 targets multiple faults simultaneously. Instead of adding one key gate per locked circuit as in Approach 1, the second approach locks a circuit with multiple key gates where the

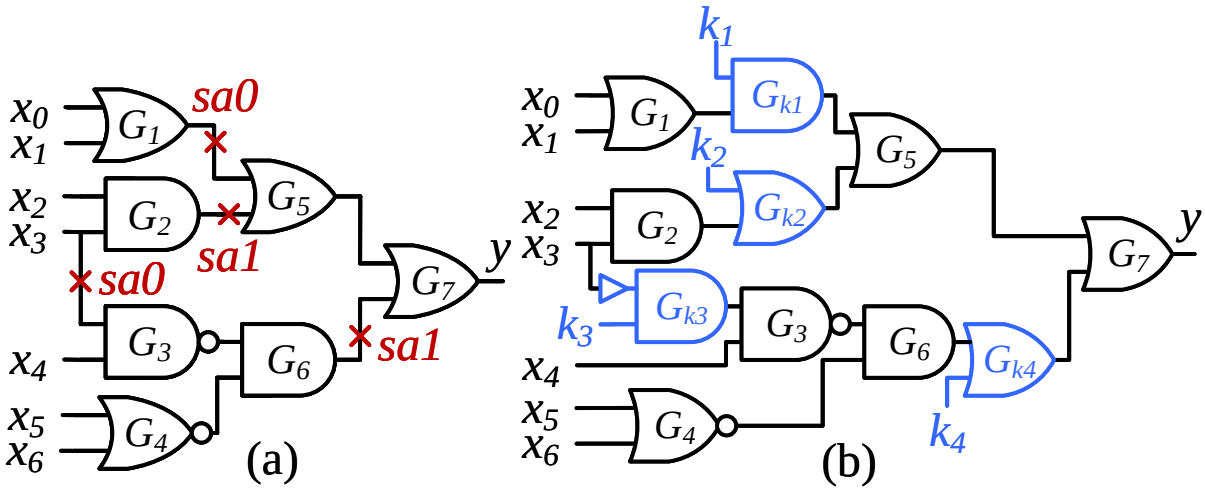

Figure 7.7: Test generation with a group of faults. (a) circuit with multiple faults (b) the *saf*'s equivalence with logic locking.

number of keys is the same as the to-be-analyzed faults. This approach resembles the prevalent strategy within the logic locking community, where a circuit is locked with multiple key bits. From the SAT attack perspective, each distinguishing pattern can, in general, remove multiple incorrect keys in the search space. This is because any incorrect key assignment in the traditional logic locking techniques is more likely to produce the wrong and corrupted output on a given input vector. This statement also applies to our proposed logic locking conversion of stuck-at faults with AND/OR key gates at the fault sites (as illustrated in the example below). In addition, our key-dependent fault-equivalence conversion supports the test pattern generation of detecting both *sa0* and *sa1* of the same fault site, where one can simply insert two key gates, an AND and OR gate each, in serial at the target locations. The goal of this approach is to reduce the number of test patterns. Following the novel miter construction for faults in Section 7.2.1, we transform n hard-to-detect faults into its equivalent locked circuit with n key bits. We collect all the DIPs the SAT solver identifies and the key value from the SAT attack. When faults are detectable, one or more input patterns always exist to differentiate the correct key from the wrong one. It may be true that some faults could be situated at redundant lines, where no test pattern can be generated since the circuit output does not depend on these faults. We run the simulation with the target group of faults and the extracted patterns to know precisely how many faults are detected through the SAT attack's distinguishing pattern.

Table 7.1: The SAT attack uses only 2 patterns to eliminate the incorrect keys from the search space. If the output matches the correct output, we put ✓, else ✗.

4-bit key $\{k_1, ..., k_4\}$	Pattern 1 $\{1001100\}$	Pattern 2 $\{0001100\}$
0000	✓	✗
0001	✓	✗
0010	✗	✓
0011	✓	✗
0100	✓	✗
0101	✓	✗
0110	✓	✗
0111	✓	✗
1000	✓	✗
1001	✓	✗
1010	✓	✓
1011	✓	✗
1100	✓	✗
1101	✓	✗
1110	✓	✗
1111	✓	✗

We take the combinational circuit with 4 stuck-at faults, shown in Figure 7.7(a), as an example. Both *sa0*s are turned to AND gates G_{k1}, G_{k3} with keys k_1, k_3, and the *sa1*s are locked with OR gates G_{k2}, G_{k4} with keys k_2, k_4. Note that the input x_3 branches to two lines, and only the wire connected to the input of gate G_3 has *sa0*, but not for the one at the input of gate G_2. So, we include one buffer for the conversion of this *sa0*, as described in Section 7.2.1. With Approach 2, the SAT attack only uses 2 DIPs, $P_1 = \{x_0, x_1, ...x_6\} = \{1001100\}$, $P_2 = \{x_0, x_1, ...x_6\} = \{0001100\}$, instead of 4 DIPs with Approach 1, to prune all 15 wrong key combinations and correctly derive the key vector $\{k_1, k_2, k_3, k_4\} = \{1010\}$. For both patterns, we show in Table 7.1 detailing whether the 16 possible keys produce the correct output y or not, where ✓ indicates a match with the oracle output and ✗ for a mismatch, and the correct key is highlighted in red. On the first iteration of the SAT attack, it finds the first distinguishing pattern $P_1 = \{1001100\}$, which removes key $\{k_1, ..., k_4\} = \{0010\}$ from the key space. On the second iteration, it returns another distinguishing pattern $P_2 = \{0001100\}$ that crosses out another 14 keys, leaving only one key in the key space. On the third iteration, no more DIP can be found to create differential output with the only remaining key $\{k_1, ..., k_4\} = \{1010\}$ as the

SAT attack already removed all the incorrect keys at the first two iterations. As it does not find any satisfiable pattern in the third iteration, the SAT attack terminates and returns the correct key $\{k_1, ..., k_4\} = \{1010\}$.

The proposed Approach 2 can significantly reduce the test pattern generation time compared to Approach 1. As the miter construction in the SAT attack initialize the key inside the locked circuit (e.g., instance A, see Figure 7.6) and is assigned with Boolean logic consistent with the learned clauses of the previous rounds, it offers n locations potentially for reducing the conflicts or backtracks for the SAT solver when n faults are grouped together. This can offer much more efficiency to deduce the satisfiable assignment as the miter circuit of the single fault has 1 initial starting point while the n-bit key provides $n - 1$ more pre-assigned locations. This leads to further test time reduction compared with a single stuck-at fault per run. This observation is verified in Table 7.5 in Section IV by comparing the ratio of Approach 2 to Approach 1 on the test time of hard-to-detect fault list DF_P. The total time can be much smaller than the time required to generate a single pattern in Approach 1 for b_19 benchmark circuit. An increased test time gain can be obtained when more faults are transformed together in one locked circuit.

Approach 2 can also identify redundant faults. If all the n-faults are redundant, the SAT-attack tool will not return any DIPs and will identify them uniquely. However, if there exist one or more detectable faults, we cannot identify them uniquely from the UNSAT conclusion. One more step is necessary to identify the redundant faults using fault simulation. The patterns obtained from the proposed Approach 2 need to be applied to the fault simulator and identify the undetected faults. As no test pattern exists for any redundant faults, all these undetected faults reported by the fault simulator must be redundant. In summary, Approach 2 can target a group of redundant faults where the SAT attack can finish solving all keys in the first few iterations, with the UNSAT conclusion at the very last round.

7.2.4 Test Pattern Generation and Redundant Fault Identification Algorithm

Algorithm 5 shows the identification of redundant faults and the test pattern generation process for hard-to-detect faults. The algorithm has three steps, and it first performs traditional ATPG

Algorithm 5: Proposed SAT-based test pattern generation for hard-to-detect faults and identification of redundant faults.

Input : Combinational circuit in *.bench* format (C_N) and standard cell library ($stdlib$)

Output: Redundant fault set (RF), hard-to-detect fault set (DF), test pattern set (P_{A1} and P_{A2})

//——— Step-1: Generate undetected fault list L ———

$C_{IN} \leftarrow$ `fromBench`(C_N) ;
$C_{DN} \leftarrow$ `toTechDependentNetlist`$(C_{IN}, stdlib)$;
$tp \leftarrow$ `write_drc_file`$(C_{DN}, stdlib)$;
`loadATPG`$(C_{DN}, stdlib, tp)$;
`addFaults`$(sa0, sa1,$ *'all'*$)$;
`setAbortLimit`(max) ;
`runATPG` ;
$L \leftarrow$ `reportFault`(*undetected*) ;
$[RF_{A1}, DF_{A1}, P_{A1}] \leftarrow$ `Approach-1`(C_{DN}, C_N, L) ;
$[RF_{A2}, DF_{A2}, P_{A2}] \leftarrow$ `Approach-2`(C_{DN}, C_N, L) ;
return $RF_{A1,A2}$, $DF_{A1,A2}$, $P_{A1,A2}$;

//——— Step-2: Approach 1 ———

function `Approach-1` *(*C_{DN}, C_N, L*)* **is**
    $[RF_{A1}, DF_{A1}, P_{A1}] \leftarrow \varnothing$;
    for $i \leftarrow 1$ **to** $|L|$ **do**
        $f \leftarrow L[i]$;
        $C_{L_{A1}} \leftarrow$ `locked-ckt`$(C_{DN}, f,$ *'Approach 1'*$)$;
        $C_{B_{A1}} \leftarrow$ `toBench`$(C_{L_{A1}})$;
        $[p, k] \leftarrow$ `SAT-attack`$(C_{B_{A1}}, C_N)$;
        if $p = \varnothing$ **then**
            $RF \leftarrow$ `append`(f) ;
        else if $p! = \varnothing$ & $k = k_{ref}$ **then**
            $DF \leftarrow$ `append`(f) ;
            $P_{A1} \leftarrow$ `append`(p)
        end
    end
    return RF_{A1}, DF_{A1}, and P_{A1} ;
end

//——— Step-3: Approach 2 ———

function `Approach-2` *(*C_{DN}, C_N, L*)* **is**
    $[RF_{A2}, DF_{A2}, P_{A2}] \leftarrow \varnothing$;
    $C_{L_{A2}} \leftarrow$ `locked-ckt`$(C_{DN}, L,$ *'Approach 2'*$)$;
    $C_{B_{A2}} \leftarrow$ `toBench`$(C_{L_{A2}})$;
    $[P, K] \leftarrow$ `SAT-attack`$(C_{B_{A2}}, C_N)$;
    $[DF_{A2}, RF_{A2}] \leftarrow$ `faultSim`(C_{DN}, L, P) ;
    $P_{A2} \leftarrow P$;
    return RF_{A2}, DF_{A2}, and P_{A2} ;
end

using a commercial tool to generate test patterns and report undetected faults. As our objective is to use the SAT attack to generate test patterns and identify redundant faults, we made a few adjustments to the traditional approach of test pattern generation, which starts from synthesizing a circuit using a commercial tool (e.g., Synopsys Design Compiler). If we are given an RTL code, we follow the traditional approach to obtain the technology-dependent gate-level netlist from design synthesis with standard cell library *stdlib*. If the synthesized netlist is a sequential design, scan-chain insertion is required to convert the sequential design to a combinational one so that ATPG can generate test patterns efficiently. On the other hand, if the benchmark is already in the combinational *bench* format, *e.g.,* the '*_C*' circuits in the ITC'99 benchmark suite (`https://github.com/squillero/itc99-poli`), we can directly convert the *bench* file C_N to a technology-independent gate-level netlist C_{IN}, Algorithm 5, Line 1. Then, this technology-independent netlist C_{IN} can be mapped to a technology-dependent netlist C_{DN} with standard cell library $stdlib$, Line 2. This can be done without synthesizing the design C_{IN}, which may introduce potential line mismatch during optimization, and the synthesized netlist may deviate from its original *bench* netlist C_N. Any standard cell library can map the technology-independent netlist to a technology-dependent one for commercial ATPG tools. As we target only the stuck-at faults, they are independent of the parameters in the library, unlike delay, bridging faults, or resistive opens that are dependent on the intrinsic properties of the technology node. The ATPG tool also requires a test protocol tp in *SPF* format, which can be either generated from netlist synthesis or directly written within the ATPG tool [244] by the command `write_drc_file`, Line 3. After loading netlist C_{DN}, library $stdlib$ and test protocol tp to ATPG tool, Line 4, stuck-at 0 (*sa0*) and stuck-at 1 (*sa1*) faults are assigned to all lines in the circuit, including the primary input and output, Line 5. Since fault coverage can be improved by increasing the allotted number of backtracks and remade decisions of the ATPG tool, we set the abort limit to its maximum value, Line 6. ATPG is then invoked to run test pattern generation and fault coverage analysis, Line 7, and report any undetected faults by the tool to a list L, Line 8.

The algorithm identifies the redundant faults from the undetected fault list L (Lines 9, 12-27) in Step-2 using Approach 1. Three empty sets are initialized, hard-to-detect fault set

DF_{A1}, redundant fault set RF_{A1}, and test pattern set P_{A1}, Line 13. For each fault f in the undetected list L (Line 15), the locked circuit $C_{L_{A1}}$ is modeled with a single key bit, Line 16. After converting the locked netlist $C_{L_{A1}}$ to *bench* format $C_{B_{A1}}$, Line 17, the SAT attack is executed with $C_{B_{A1}}$ and the oracle C_N to obtain the DIP p and the key value k, Line 18. If the SAT attack does not return a DIP from the miter circuit, p is empty, and the fault is redundant, where it is added to the redundant list RF_{A1}, Lines 19-20. However, if the SAT attack finds a DIP and the correct key value compared to the reference key ($k_{ref} = 1$ for *sa0* and $k_{ref} = 0$ for *sa1*) for the targeted fault, this fault f is detected (Line 21). It is appended to the hard-to-detect list DF_{A1} and its DIP p is added to test pattern set P_{A1}, Lines 22-23. Note that fault f belongs to either category, RF_{A1} or DF_{A1}, and no fault skips the *if-else-if* statement, as analyzed in Section 7.2.1, 7.2.2.

In Step-3, the algorithm optimizes the test pattern set for all undetected faults in list L, Lines 10, 28-36, as the previous step reports either one test pattern or none per fault. The hard-to-detect fault set DF_{A2}, redundant fault set RF_{A2}, test pattern set P_{A2} is initialized as an empty set, Line 29. All faults in the undetected list L are converted to a locked circuit $C_{L_{A2}}$ with $|L|$ number of key gates, Line 30, as described in Section 7.2.3. The locked circuit $C_{L_{A2}}$ is then mapped it to its equivalent *bench* file $C_{B_{A2}}$, Line 31. Both $C_{B_{A2}}$ and C_N are applied to the SAT attack, and the returned $|L|$-bit key value K and the DIPs P are saved, Line 32. The key K is validated by checking individual bits with the corresponding equivalent fault representation. The fault simulation is performed to identify the detected and redundant faults, Line 33. As we correctly determine the key (K), the test pattern must detect all the faults except the redundant ones that do not impact the functionality. As a result, undetected faults from the fault simulation must be redundant. These patterns in P are recorded in the set P_{A2}, Line 34. Upon execution of the algorithm, four sets of redundant faults RF_{A2}, hard-to-detect fault DF_{A2}, test patterns P_{A1} and P_{A2} are reported, Line 35.

7.3 Result and Analysis

This section presents the experimental results of our proposed SAT-based test pattern generation and redundant fault identification. The proposed miter construction with the SAT attack and test pattern generation are analyzed using ITC'99 benchmark circuits (`https://github.com/squillero/itc99-poli.`). We use Synopsys 32nm SAED32 library to map the benchmark circuits to technology-dependent netlists, which are read in with TetraMAX II ATPG [244]. Any advanced technology nodes can also be applied to map the technology-independent *bench* file with the standard cells in the library, as described in Section 7.2.4. We first apply a commercial ATPG tool, Synopsys TetraMAX II, to generate test patterns for detecting all the *sa0* and *sa1* faults in a circuit. The tool reports test patterns, fault coverage, and undetected faults. We have not modified the existing test pattern generation process. To determine the hard-to-detect faults that are previously undetected and find the corresponding test vectors, we (i) replaced all undetected faults with their key-based equivalent gates, and (ii) apply the proposed Approaches 1 and 2 to obtain additional test patterns. Note that our proposed technique provides additional coverage to the test results from TetraMax II.

Table 7.2 summarizes our findings with 25 combinational benchmarks from ITC'99. We excluded simple benchmark circuits, where the TetraMAX II detects all stuck-at faults. Any faults that have no pattern generated are labeled as undetected ones. All the undetected faults reported by TetraMAX II have been evaluated with the proposed logic locking-based fault representation and the SAT attack. We apply the proposed approaches for detecting these undetected faults. Column 2 shows the total gate count for each benchmark. The total number of stuck-at faults, TF, for each benchmark is recorded in Column 3. For TetraMAX II [244], it includes faults under the following four categories, PT (Possibly Detected), UD (Undetectable), AU (ATPG Untestable), and ND (Not Detected) and shown in Columns 4, 5, 6, and 7 respectively. The total undetected fault count ($UF = PT + UD + AU + ND$) and fault coverage (FC) obtained from TetraMAX II are listed in Columns 8 and 9. Our proposed approach identifies these undetected faults as either redundant faults (RF_P) or hard-to-detect faults (RF_P), which are listed in Columns 10 and 11. Column 12 represents the total fault coverage (FC_T)

Table 7.2: Stuck-at Faults Summary for ITC'99 Benchmarks.

Benchmark	Gate Count	Total Faults (TF)	DF_{TMAX}	TetraMAX II						PA		Total
				PT	UD	AU	ND	UF	FC (%)	RF_P	DF_P	FC_T(%)
b04_opt_C	543	3554	3549	0	5	0	0	5	99.86	5	0	100
b04_C	657	4144	4094	0	50	0	0	50	98.79	50	0	100
b05_opt_C	505	3272	3265	4	3	0	0	7	99.85	3	4	100
b05_C	943	5850	4747	0	1099	4	0	1103	81.15	1103	0	100
b07_opt_C	371	2456	2455	0	1	0	0	1	99.96	1	0	100
b07_C	385	2470	2464	0	6	0	0	6	99.76	6	0	100
b11_opt_C	511	3318	3316	0	2	0	0	2	99.94	2	0	100
b11_C	734	4378	4212	0	161	0	5	166	96.21	161	5	100
b12_opt_C	886	6048	6047	0	1	0	0	1	99.98	1	0	100
b13_C	290	1928	1848	0	80	0	0	80	95.85	80	0	100
b14_opt_C	9811	58584	58265	0	318	0	1	319	99.46	319	0	100
b14_C	5477	35844	35806	0	33	2	3	38	99.89	38	0	100
b15_opt_C	7206	48220	46922	19	1054	38	187	1298	97.33	1287	11	100
b15_C	8462	53470	51952	0	1329	76	113	1518	97.16	1518	0	100
b17_opt_C	23523	157418	154248	15	1153	300	1702	3170	97.99	3144	26	100
b17_C	31091	192174	187897	0	3866	33	378	4277	97.77	4276	1	100
b20_opt_C	12170	79748	79644	0	99	0	5	104	99.87	104	0	100
b20_C	19792	118298	117614	35	631	18	0	684	99.44	631	53	100
b21_opt_C	12344	80504	80398	0	95	0	11	106	99.87	106	0	100
b21_C	20109	120436	119742	8	684	0	2	694	99.43	686	8	100
b22_opt_C	17614	114556	114387	1	165	0	3	169	99.85	169	0	100
b22_C	29316	175510	174653	33	795	29	0	857	99.52	796	61	100
b18_opt_C	71392	469602	469044	0	479	17	62	558	99.88	544	14	100
b18_C	112421	672242	668478	0	3658	96	10	3764	99.44	3760	4	100
b19_C	226936	1355584	1347025	6	8236	191	126	8559	99.37	8460	99	100

after applying our proposed SAT-based test pattern generation in addition to TetraMAX II. We computed the total fault coverage for Column 12 using the following Equation:

$$FC_T = \frac{DF_{TMAX} + DF_P + RF_P}{TF} \times 100,$$

where, DF_{TMAX} is the number of detected faults from the TetraMAX II tool. For example, the $b20\_C$ benchmark has 118298 faults, out of which 684 faults are not detected by TetraMAX II. Our proposed approach detects 53 hard-to-detect faults and identifies the rest 631 faults as redundant. Note that many of the small circuits do not have any hard-to-detect faults (e.g., $b04\_C$, $b05\_C$, etc.), and all the undetected faults are redundant. For bigger benchmark circuits (e.g., $b19\_C$), we observe an increased number of both the hard-to-detect and redundant faults. Note that our approach can generate test patterns for all the hard-to-detect faults and identify all the redundant faults resulting in a perfect fault coverage of 100%.

Table 7.3: Hard-to-Detect Faults summary in ITC'99 Benchmarks.

Benchmark	**Detected Fault by Category**				**Total**
	D/PT	D/UD	D/AU	D/ND	DF_P
b05_opt_C	4/4	0/3	0/0	0/0	4
b11_C	0/0	0/161	0/0	5/5	5
b15_opt_C	3/19	0/1054	0/38	8/187	11
b17_opt_C	10/15	0/1153	0/300	16/1702	26
b17_C	0/0	0/3866	0/33	1/378	1
b20_C	35/35	0/631	18/18	0/0	53
b21_C	8/8	0/684	0/0	0/2	8
b22_C	32/33	0/795	29/29	0/0	61
b18_opt_C	0/0	0/479	0/17	14/62	14
b18_C	0/0	0/3658	0/96	4/10	4
b19_C	4/6	0/8236	0/191	95/126	99

We identify these DF_Ps from the four undetected fault categories reported by TetraMAX. Table 7.3 shows the number of hard-to-detect faults from PT, UD, AU, and ND categories. The second column represents the additional detected faults (D) from PT and is presented as D/PT. Similarly, Columns 3, 4, and 5 show additional detected faults from UD, AU, and ND, respectively. We have detected a few faults from PT, AU, and ND categories, except

UD categories. For example, 32 faults from PT and 29 faults from AU are detected for *b22_C* benchmark. Similarly, 95 out of 126 faults are detected from the ND category for *b19_C* benchmark. However, we did not observe any detected faults from the UD category, and they are all redundant. In summary, we found that some faults from all the other categories, except UD, are hard-to-detect while others are redundant.

Table 7.4: Comparison on number of test patterns on SAT detected faults between Approach 1 and Approach 2.

Benchmark	**Approach 1**	**Approach 2**	**Reduction**
b05_opt_C	4	1	75.00%
b11_C	5	2	60.00%
b15_opt_C	11	7	36.36%
b17_opt_C	26	12	53.85%
b17_C	1	1	0%
b20_C	53	25	52.83%
b21_C	8	3	62.50%
b22_C	61	26	57.38%
b18_opt_C	14	7	50.00%
b18_C	4	1	75.00%
b19_C	99	8	91.92%

For each benchmark, we combine all faults in the hard-to-detect fault set DF_P to generate the optimized test set with the proposed Approach 2 and the SAT attack, presented in Section 7.2.3. Table 7.4 compares the number of test patterns required for Approach 1 and Approach 2. Columns 2 and 3 record the total test pattern count for Approach 1 and Approach 2 on DF_P, respectively. Column 3 represents the percentage decrease in the number of test patterns between Approach 2 and Approach 1. As shown in Table 7.4, we can see a significant reduction in the number of test patterns required to identify the faults, with an average of 52.29% fewer test vectors. For example, the 61 hard-to-detect faults in DF_P in *b22_C* benchmark need 61 test patterns with Approach 1 but only 26 test vectors using the proposed Approach 2. The fault simulation validated all the input vectors returned by the SAT attack. In addition, all the key bits in each locked circuit have been validated with the proposed miter construction for stuck-at faults, and they all match the expected key values.

Table 7.5: Test Generation Time Summary for ITC'99 Benchmarks.

Benchmark	# DF_P	# UF	**Approach 1 (s)**		**Approach 2 (s)**		**Improvement** ($\times$)	
			DF_P	$DF_P + RF_P$	DF_P	$DF_P + RF_P$	DF_P	$DF_P + RF_P$
b04_opt_C	0	5	–	0.43	–	0.07	–	5.8
b04_C	0	50	–	4.9	–	0.11	–	44.5
b05_opt_C	4	7	0.60	0.77	0.15	0.08	4	9.6
b05_C	0	1103	–	576.4	–	558.7	–	1.0
b07_opt_C	0	1	–	0.05	–	0.05	–	1.0
b07_C	0	6	–	0.41	–	0.08	–	5.5
b11_opt_C	0	2	–	0.20	–	0.07	–	2.8
b11_C	5	166	0.54	58.7	0.10	0.14	5.4	419.9
b12_opt_C	0	1	–	0.10	–	0.10	–	1.0
b13_C	0	80	–	2.3	–	0.05	–	42.7
b14_opt_C	0	319	–	414.9	–	11.6	–	35.6
b14_C	0	38	–	71.9	–	30.9	–	2.3
b15_opt_C	11	1298	62.4	4,122.9	9.9	170.4	6.3	24.2
b15_C	0	1518	–	4,071.6	–	10.9	–	374.2
b17_opt_C	26	3170	457.8	25,967.4	22.5	4780.1	20.3	5.4
b17_C	1	4277	16.8	31,559.3	16.8	133.7	1	236.0
b20_opt_C	0	104	–	334.8	–	22.3	–	15.0
b20_C	53	684	139.5	1,764.3	6.8	135.8	20.5	13.0
b21_opt_C	0	106	–	284.7	–	22.8	–	12.5
b21_C	8	694	15.7	2,085.4	1.2	247.7	13.1	8.4
b22_opt_C	0	169	–	731.7	–	31.0	–	23.6
b22_C	61	857	257.2	4,021.4	8.2	584.6	31.4	6.9
b18_opt_C	14	558	96,199.7	7,045,240.5	1,518.7	1471.1	63.3	4789.1
b18_C	4	3764	12,474.5	*timeout*	1,992.7	3740.7	6.3	$>$1000
b19_C	99	8559	2,083,961.1	*timeout*	8,728.8	4675.1	238.7	$\gg$1000

We run our proposed algorithms on a 20-core Intel Xeon CPU with 2.60 GHz and 64 GB RAM. The SAT program runs on a single thread in CentOS Linux 7 operating system. We only consider the SAT attack time as the preprocessing, such as technology-dependent netlists to technology-independent bench file conversion, processing of TetraMAX report, fault simulation, etc., can be performed in parallel. Table 7.5 shows the runtime for detecting both hard-to-detect faults DF_P and the redundant faults RF_P using our proposed Approaches 1 and 2. Note that we compare the time complexity for our proposed approaches only due to the –(i) unavailability of programs for prior SAT-based approaches in the public domain, and (ii) many optimizations performed over the years for commercial ATPG tools. As our server has 20 cores, a total of 40 threads that target 40 faults can be run in parallel. We set 5 days as the timeout for all undetected faults. Columns 2 and 3 are the fault count for DF_P and total undetected faults (UF) from TetraMAX II, respectively. Note that $UF = DF_P + RF_P$ as we either detect all these faults or identify them as redundant. Columns 4 and 5 show the total test time for DF_P and $DF_P + RF_P$ using our proposed Approach 1. The same is shown in Columns 6 and 7 for Approach 2. Columns 8 and 9 represent the test time improvement of Approach 2 over Approach 1. Please note that $\#DF_P$ fault count (Column 2) for b18_C and b19_C are computed from fault simulation with the test patterns obtained from the SAT attack with $DF_P + RF_P$ and Approach 2 (Column 7), where a timeout of 5 days for a single fault is observed, marked as *timeout* in Column 5. For example, Approach 1 takes 0.54s, whereas Approach 2 requires only 0.10s to detect 5 hard-to-detect faults for the b11_C benchmark circuit. However, the gain for Approach 2 becomes significant for larger benchmark circuits. For b19_C, Approach 1 takes 2,083,961.1s to detect 99 faults, which is much larger than Approach 2, 8728.8s. The improvement in test time is $2,083,961.1/8,728.8 = 238.7$ times. *This signifies the fact that detecting 99 DF_Ps together is easier than detecting a single fault. Interestingly, generating test patterns even for detecting and identifying 8559 faults as DF_P and RF_P takes less time than detecting 99 DF_Ps.* We observe that the overall gain becomes significant when there are an increased number of faults to be grouped during test pattern generation, and the number of conflicts typically increases with the benchmark size. The reason for this increase in the ratio is

due to the reduction of conflicts from the initialization of the multiple key bits in the SAT-attack tool.

7.4 Summary

In this chapter, we presented how the widely explored SAT attack on logic locking can be used to identify redundant faults and generate test patterns for hard-to-detect stuck-at faults. We first present the miter construction of stuck-at faults to a key-dependent locked circuit so that the powerful SAT tool can be used. This ensures that the input patterns used to break our logic locking technique can be applied to detect the stuck-at faults. Since the SAT-based attack effectively breaks multiple logic locking schemes, we exploit it to generate test vectors for stuck-at faults with the corresponding locked circuits. If faults are observable at the primary output, the distinguishing input patterns returned from the SAT attack can expose them. On the other hand, if any faults are redundant, no distinguishing input can be found from the SAT attack, and the program finishes directly with the UNSAT conclusion from the SAT solver. By applying our proposed approach, we were able to identify any redundant faults from the undetected stuck-at faults reported by the ATPG tool or obtain the necessary test patterns for those non-redundant hard-to-detect faults. Our test pattern generation approach can also be optimized for a reduced pattern set by grouping multiple faults into a single locked circuit.

Chapter 8

On-Demand Device Authentication using Zero-Knowledge Proofs for Smart Systems

Over the past decades, the world has experienced a booming Internet along with its wide range of web applications. The success of the interconnected computer network fuels the emergence of lightweight Internet of Things (IoTs), or edge devices. In turn, the advancement and adoption of IoTs and cyber-physical systems (CPS) offer foundations for smart cities and offered solutions to critical infrastructures such as transportation, public safety, health care, smart grid, etc. With 14.4 billion IoT devices currently connected to the Internet [271], this number is estimated to climb to 55.7 billion by 2025 [272]. These low-cost devices are deployed in the field for better integrating computing, cyber, and physical spaces, often in which proprietary information is included. However, this abundance of devices presents a severe challenge to ensuring the overall security of the connected infrastructure, as a genuine device can be replaced with a cloned one with a backdoor. Authenticating each device is a must to ensure the integrity and security of the IoT network as a whole. The insider threat [273, 274] is a looming issue, as it is challenging to monitor all edge devices while they are operational. An adversary can replace an authentic device with its cloned version and can gain access to the entire infrastructure, bypassing all security measures [275–277]. It is essential to authenticate an edge device regularly so the server can periodically evaluate and monitor its fidelity. The response received from an edge device serves as an indicator of its true identity.

Device authentication mechanisms using physically unclonable functions (PUFs) have been proposed over the years [278–283]. Unfortunately, PUFs can be modeled if an adversary observes a set of challenge-response pairs (CRPs), and the responses can be predicted

without accessing the physical device [284–288]. It is necessary to develop a novel communication protocol where an adversary cannot access PUF responses and thus cannot model the PUF. In addition, the secrecy of the PUF response should be kept secure even after repeated authentication. When deployed, the device must be checked regularly to detect the malicious replacement with a cloned counterpart. The same level of trust for a device must be maintained after post-deployment [289].

This chapter proposes an efficient, secure, and on-demand device authentication protocol using zero-knowledge proofs (ZKPs) [31]. The signature from a physically unclonable function (PUF) is used to create the proof, which can be verified by the central server acting as a verifier without storing the actual PUF response. The edge device periodically generates proofs using a self-generated random seed. The proof can then be made public for verification for anyone with the common reference string (CRS). This allows repeated authentication by verifying the identity of an edge device without access to the PUF secret.

This chapter is organized as follows. We introduce PUF and ZKP in Section 8.1. The proposed IoT device authentication framework is described in Section 8.2. The experimental result and performance analysis for the proposed approach are described in Section 8.3. Finally, we conclude the paper in Section 8.4.

8.1 Background

This section presents an overview of physically unclonable functions (PUF), the PUF-based secure communication protocols, and their challenges. We describe the prior work on ZKPs, Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK), and elliptic curve pairing-based zk-SNARK.

8.1.1 Physically Unclonable Functions

Throughout the years, the research community has proposed various architectures for physically unclonable functions (PUFs) and their implementations [278, 290–292]. PUFs derive their behavior from the inherently uncontrollable and unpredictable variations in the semiconductor manufacturing process. Therefore, its application involves security and privacy-related

research for its ability to generate unclonable bits as a unique key for identification and authentication. Over the years, different types of PUFs have been proposed, *e.g.,* arbiter PUF [290], ring oscillator PUF (ROPUF) [278, 293], SRAM PUF [291, 292], among others. These designs leverage the unclonable characteristic in path delay and/or threshold voltage biases to demonstrate the uniqueness of PUF.

Multiple PUF-based device authentication protocols have been proposed in recent years [278–283]. Arbiter PUFs and ROPUFs were first applied in device authentication [278], where the server keeps a database of CRPs for each PUF instance. SRAM PUF-based authentication protocols [279, 283] also explicitly store PUF responses on the server side for verification or key derivation. Chatterjee et al. does not keep PUF responses in the verifier's database, but the offlined security credential generator has those copies from the enrollment phase [281]. Although security is guaranteed for the above-mentioned algorithms with unclonable PUFs, machine learning attacks with a subset of given challenge-response pairs (CRPs) have been demonstrated effective in accurately predicting responses [284–288]. Rührmair et al. first exploits logic regression for learning the features in CRPs to predict unseen Arbiter and XOR Arbiter PUF responses with high probability [284]. Subsequently, different machine learning-based attacks have surfaced to target the extrapolation of PUF responses from various PUF architectures [285–288].

8.1.2 Zero-Knowledge Proofs

The overarching goal of ZKPs is to convince a verifier that a prover possesses an existing secret without ever revealing the secret. These proofs were conceptualized initially in 1985 by Goldwasser et al. [29]. This laid the foundation for the subsequent development of the Feige-Fiat-Sharir protocol, one of the first identification schemes using interactive ZKPs [294]. In interactive ZKPs, real-time communication between the prover and the verifier is necessary to complete the protocol. Non-interactive ZKPs, on the other hand, relieve both parties from repeated communications. This concept is first explored through Fiat-Shamir heuristic [295] and then developed by Blum et al. [296]. The non-interactive nature and the adopted random oracle

model allow any entity to verify the validity of a proof, where the proof generation no longer relies on the verifier-dependent random values. Succinct non-interactive zero-knowledge proofs (zk-SNARKs), however, requires the verification time on the order of logarithmic time complexity with respect to the circuit size. In other words, zk-SNARKs are non-interactive zero-knowledge proofs efficient to be validated by the verifier. The popular cryptocurrency ZCash uses the Zerocash framework [297] with zk-SNARK proof system to ensure the anonymity of user identity [298]. Over the years, different zk-SNARKs protocols have been proposed and quickly gained popularity in privacy-based blockchain applications, such as Pinocchio [299], Groth'16 [31], Bulletproofs [32], Sonic [300], Marlin [301], etc.

8.1.3 Pairing-based zk-SNARKs

Despite various ZKPs having been proposed recently, the zk-SNARK proposed by Groth [31] remains to be very efficient both for the shorter proof size and less computation complexity for the verifier [31, 300]. Groth'16 operates on pairing-based cryptography, whose underlying cryptographic hardness lies in the bilinear Diffie-Hellman problem on elliptic curves [302]. It constructs the rank-1 constraint system (R1CS) of a quadratic arithmetic program (QAP) with ℓ public statement, where each gate represents a multiplication/exponentiation operation on elliptic curve. Computations are performed under bilinear groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ that forms a non-degenerate bilinear mapping $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. The constant proof size contains 3 group elements, 2 in $\mathbb{G}_1$ and 1 in $\mathbb{G}_2$. Once a proof is received, the verifier will compute ℓ exponentiation in $\mathbb{G}_1$, 3 pairing values, and check the quantitative equivalence of pairings in $\mathbb{G}_T$.

8.2 Proposed Approach

This section details our proposed zk-SNARK-based approach for IoT device authentication. The zero-knowledge scheme provides security to PUF-based IoT verification methods without revealing the secret asset in each edge device, *e.g.,* PUF response. Figure 8.1 shows the overview of the proposed method, which consists of two phases, the registration phase and the operational phase. The first phase consists of PUF calibration and response extraction, and initialization of parameters necessary for the trusted setup of zk-SNARK. This phase is performed

in a trusted environment right before the deployment of IoT devices. Once the device is in the field, it can be periodically checked and verified by the trusted server by validating zk-SNARK proofs it generated.

8.2.1 Threat Model

This chapter assumes the adversarial capability of replacing the authentic device with a counterfeit, tampered, or cloned device. In addition, the threat model assumes that the device does not provide direct access of the PUF challenge-response pairs. This is practical as the semiconductor industry typically disables the scan chain access after the manufacturing test [77]. Similar features of blocking scan access can be implemented here to prevent attackers from obtaining the PUF responses. The adversary simply cannot bypass the security mechanism while the device is in the field.

8.2.2 Registration Phase

Registration of the edge device begins with creating a stable PUF response at a given challenge. We follow the pairing-based zk-SNARK construction of Groth'16 [31] to authenticate and verify IoT devices using each device's inherent secret asset without revealing it to them. In the registration phase, both the central server and the edge device need to build a mutually agreed relation $\mathcal{R}$ and trusted setup σ, as shown in steps ① and ② in Figure 8.1. Additionally, a valid hash for the PUF response and its error correction syndrome will be computed and made public by the edge device, as shown in steps ③ and ④ in Figure 8.1. This initialization phase can be completed in a secure environment as the central server is considered as trusted.

• **Construct Relation $\mathcal{R}$:** Before constructing and validating proofs, both parties need to agree on the public parameters, similar to Alice and Bob coordinating on the same prime and primitive root for computations in the Diffie-Hellman protocol [303]. In the ZKP protocol, the relation $\mathcal{R}$ is mutually communicated between both the prover and the verifier. Let $\mathcal{R}$ be a relation of a given security parameter λ with prime field $\mathbb{F}_p$, generators $g_1, g_2 \in \mathbb{F}_p$ of elliptic curve groups $\mathbb{G}_1, \mathbb{G}_2$ of prime order q, bilinear mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ of prime order q

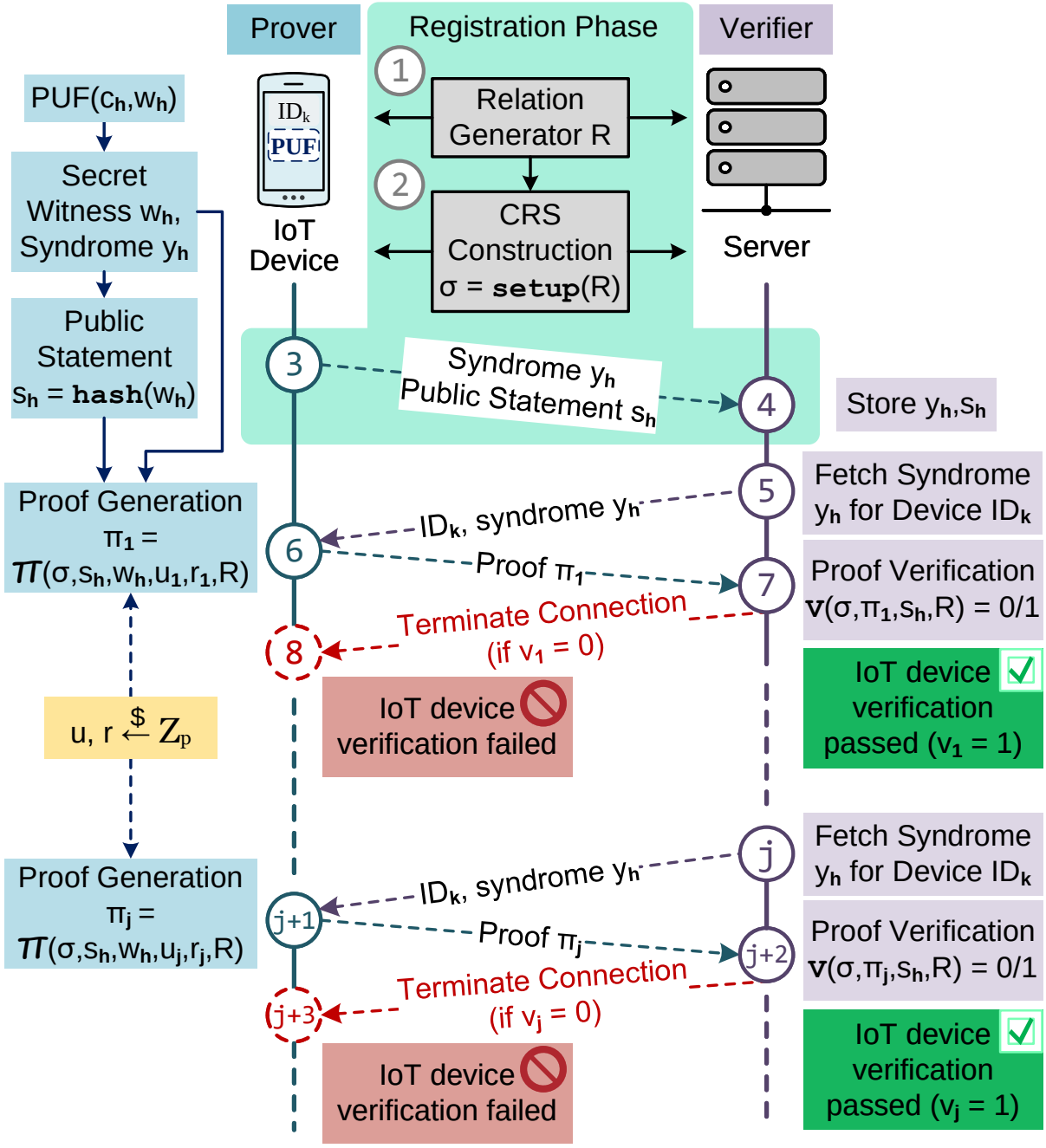

Figure 8.1: The proposed device authentication scheme using ZKPs.

with generator $g_t = e(g_1, g_2)$, scalars m, ℓ, $3m$ polynomials $\{f_{A_i}(X), f_{B_i}(X), f_{C_i}(X)\}_{i=1}^{m}$ of degree $n-1$, and polynomial $f_Z(X)$ of degree n,

$$\mathcal{R} = \left(\begin{array}{c} p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \ell, \\ \{f_{A_i}(X), f_{B_i}(X), f_{C_i}(X)\}_{i=1}^{m}, f_Z(X) \end{array} \right).$$

Due to the process variations in chip manufacturing, the SRAM inside each IoT device contains its own unique signature, *i.e.,* the power-up state of the SRAM cells [291, 304]. It is feasible to extract an SRAM PUF from the stable bits by calibrating and eliminating the unstable bits from the power-up state. This is applicable to other types of PUFs also. A set of challenges $\{c_1, \ldots, c_h, \ldots\}$ can be constructed for a PUF. For a given challenge c_h, one can obtain a unique PUF response w_h for each device and the corresponding error correction syndrome y_h to ensure consistency in w_h. Due to the unclonable property of the PUF response, the IoT device has secret witness w_h, and the corresponding hash value as the public statement $s_h = \texttt{hash}(w_h)$. The concatenation of witness w_h and statement s_h is the solution to the quadratic arithmetic circuit in R1CS defined over $\{f_{A_i}(X), f_{B_i}(X), f_{C_i}(X)\}_{i=1}^{m}$ and $f_Z(X)$,

$$\sum_{i=1}^{m} d_i f_{A_i}(X) \cdot \sum_{i=1}^{m} d_i f_{B_i}(X) - \sum_{i=1}^{m} d_i f_{C_i}(X) = f_Q(X) f_Z(X),$$

where $f_Q(X)$ is the quotient polynomial of degree $n-2$, $s_h = (d_1, \ldots, d_\ell) \in \mathbb{F}_p^\ell$, $w_h = (d_{l+1}, \ldots, d_m) \in \mathbb{F}_p^{m-\ell}$, and $d = (d_1, \ldots, d_m) = (s_h, w_h) \in \mathbb{F}_p^m$. As it involves the secret witness w_h, the quotient polynomial $f_Q(X)$ is computed by the edge device and is not shared with the central server.

• **Initialize Trusted Setup σ:** During the trusted setup, the IoT device and the central server compute the common reference string (CRS), which is essentially the basis for both proof generation and validation. It applies the polynomials derived from the relation $\mathcal{R}$ with R1CS arithmetic circuit to construct the public elliptic curve points in $\mathbb{G}_1, \mathbb{G}_2$. It is called a trusted setup due to the fact that the creation of CRS σ needs the uniform sampling of 5 random values $\alpha, \beta, \gamma, \delta, x$ from the prime field $\mathbb{F}_p$, denoted as $\alpha, \beta, \gamma, \delta, x \xleftarrow{\$} \mathbb{F}_p$. Note that, in the registration phase, we consider both the edge device and the servers as trusted. One can select these variables at random and from the uniform distribution. Once these values are determined, elements in both cyclic groups $\mathbb{G}_1, \mathbb{G}_2$ are generated to construct the CRS σ,

$$\sigma = \left(\begin{array}{c} g_1^{\alpha}, g_1^{\beta}, g_1^{\delta}, \left\{ g_1^{x^i} \right\}_{i=0}^{n-1}, \left\{ g_1^{\delta^{-1} x^i f_Z(x)} \right\}_{i=1}^{n-2} \\ \left\{ g_1^{\gamma^{-1} \cdot \left(\beta f_{A_i}(x) + \alpha f_{B_i}(x) + f_{C_i}(x) \right)} \right\}_{i=1}^{\ell} \\ \left\{ g_1^{\delta^{-1} \cdot \left(\beta f_{A_i}(x) + \alpha f_{B_i}(x) + f_{C_i}(x) \right)} \right\}_{i=\ell+1}^{m} \\ g_2^{\beta}, g_2^{\gamma}, g_2^{\delta}, \left\{ g_2^{x^i} \right\}_{i=0}^{n-1}, g_T^{\alpha\beta} \end{array} \right).$$

The pairing $g_T^{\alpha\beta} = e(g_1^{\alpha}, g_2^{\beta})$ is also evaluated to improve further the computational succinctness for the verifier (server). Note that this device-specific CRS is stored on both parties and can be made public. As for the server, it is required to distinguish the CRS from one edge device to another, where a unique set of parameters in σ is constructed from each IoT device with its secret witness.

• **Generate and Publicize Syndrome y_h and Statement s_h:** Although error correction syndrome is required for computing the stable PUF response, each edge device can further optimize its resource utilization without saving them on the device itself. The syndromes for all challenges in set $\{c_1, \ldots, c_h, \ldots\}$ can be sent to the server instead. Syndrome y_h is computed over the PUF response w_h given the challenge c_h, where the server will transmit it back to the device before proof generation. Proving the knowledge of a secret witness, *i.e.,* PUF response, involves statements in non-deterministic polynomial time complexity (NP) [29]. It can be performed by proving to the verifier the knowledge of the preimage of a hash generated from collision-resistant and preimage-resistant hash algorithm [305]. This requires the server to be able to check the statement of a valid confirmed response against a newly generated one. Thus, during the trusted phase, the statement $s_h = \texttt{hash}(w_h)$ is computed at the IoT node and is then made public by the corresponding node. The device can simply broadcast its s_h value in the IoT network. Steps ③ and ④ of Figure 8.1 show the transmission and acceptance of s_h from the device to the server. Note that the adversary cannot obtain the PUF response from its hash s_h due to the preimage resistance of the secure hash algorithm. In case of updating the PUF response with another challenge $c_{h'}$, new statement and syndrome can be generated and made public based on the updated response $w_{h'}$.

8.2.3 Operational Phase

Once the IoT device is deployed in the field, the central server needs to ensure that adversaries cannot masquerade a malicious device as the genuine one, nor can they replace an authentic device with a clone/counterfeit one. Therefore, it is important to guarantee the authenticity of the device, where the server needs to query the device for a response and evaluate or certify the legitimacy of the device. To counter the attacker from the malicious substitution of edge devices, we propose the periodic authentication of devices, where proofs are generated from IoT nodes and are delivered to the server for verification. For each device, we denote its j^{th} proof π_j and the corresponding verification v_j with subscript j, where $j = 1, 2, 3, ...$ The proposed on-demand authentication allows the prover to initiate the proof generation, independent from the verifier, the server. In case of the required error correction syndrome needs to be retrieved online, the server can then initiates the proof-verification process by sending the corresponding syndrome y_h to the edge device for error correction in PUF extraction, as in step ⑤ in Figure 8.1. Proof generation begins in an IoT device after it obtains the witness w_h with challenge c_h and syndrome y_h.

• **Proof Generation π_j:** To prevent an adversary's malicious replacement of the edge device while deployed in the field, we secure the device authentication with proof generation and validation from the zk-SNARK protocol. The authentic device is trying to prove to the server that it knows the preimage of the hash value s_h without disclosing it to the server. The proofs in the zk-SNARK reveal nothing about the secret witness itself, *i.e.,* perfect zero-knowledge, due to the fact that it incorporates additional random values. In order to generate multiple proofs for the same device at different time intervals, it can be done for the j^{th} proof π_j by randomly selecting a different set of (u, r), sampled from $\mathbb{F}_p$ with uniform distribution, *e.g.,* a random oracle, denoted as $u_j, r_j \xleftarrow{\$} \mathbb{F}_p$. Both the secret witness $w_h = (d_{l+1}, \ldots, d_m)$ and the public statement $s_h = (d_1, \ldots, d_\ell)$ are used during the proof construction. The proof $\pi_j = \mathit{\Pi}(\sigma, s_h, w_h, u_j, r_j, \mathcal{R}) = (\pi_A, \pi_B, \pi_C)$ consists of three elliptic curve points (π_A, π_B, π_C)

with $\pi_A, \pi_C \in \mathbb{G}_1$ and $\pi_B \in \mathbb{G}_2$,

$$\pi_j = \left(\pi_A = g_1^{z_1},\ \pi_B = g_2^{z_2},\ \pi_C = g_1^{r_j z_1 + u_j z_2 + z_3 - u_j r_j \delta} \right),$$

where z_1, z_2, z_3 are computed as follows.

$$z_1 = \alpha + \sum_{i=1}^{m} d_i f_{A_i}(x) + u_j \delta; \quad z_2 = \beta + \sum_{i=1}^{m} d_i f_{B_i}(x) + r_j \delta;$$
$$z_3 = \delta^{-1} \sum_{i=\ell+1}^{m} d_i \big(\beta f_{A_i}(x) + \alpha f_{B_i}(x) + f_{C_i}(x)\big) + f_Q(x) f_Z(x)$$

• **Proof Verification** v_j**:** Once proof π_j has been received, the verifier can check the validity of the device by evaluating π_j with the public statement s_h. The server performs computations in $\mathbb{G}_T$ due to the bilinear property of $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$ in $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. The verification $v_j = \mathcal{V}(\sigma, \pi_j, s_h, \mathcal{R})$ check whether the following equality holds,

$$e(\pi_A, \pi_B) = g_t^{\alpha\beta} \cdot e(g_1^{\phi}, g_2^{\gamma}) \cdot e(\pi_C, g_2^{\delta})$$

with ϕ defined as:

$$\phi = \gamma^{-1} \Big(\sum_{i=1}^{\ell} d_i \big(\beta f_{A_i}(x) + \alpha f_{B_i}(x) + f_{C_i}(x)\big) \Big).$$

The pairing equality is satisfied only when the prover, the IoT node, has full knowledge of the input w_h that matches $s_h = \texttt{hash}(w_h)$. This is due to the negligible probability of the adversary succeeding in convincing the verifier of a false statement or finding the preimage of hash s_h, *e.g.,* the soundness property of zero-knowledge argument. If the device verification passes ($v_j = 1$), the device is considered trusted at present, and communications and interactions can proceed as normal. If the server cannot validate the proof ($v_j = 0$), the device authentication fails, and that IoT node is marked as untrusted. The server can then terminate its connection to

the specific edge device, followed by additional countermeasures if possible.

$$v_j = \begin{cases} 1,\ e(\pi_A, \pi_B) = g_t^{\alpha\beta} \cdot e(g_1^{\phi}, g_2^{\gamma}) \cdot e(\pi_C, g_2^{\delta}) \\ 0,\ e(\pi_A, \pi_B) \neq g_t^{\alpha\beta} \cdot e(g_1^{\phi}, g_2^{\gamma}) \cdot e(\pi_C, g_2^{\delta}) \end{cases}$$

With a predefined interval set between the server and each edge device, the IoT node can periodically send proofs to server and get it verified. For example, step ⑥ of Figure 8.1 shows device generates its first proof π_1 with random values $(u_1, r_1) \xleftarrow{\$} \mathbb{F}_p$. The server verifies proof π_1 in step ⑦, and it would issue termination of connection if proof verification failed ($v_1 = 0$). Similarly, the device generates the j^{th} proof π_j at the appointed time interval with syndrome y_h (step ⓙ) after all prior $(j-1)$ proofs are validated, shown in step ⓙ+1 of Figure 8.1. The server would perform the same verification as before to determine the authenticity of the device.

8.2.4 Security Analysis

The zk-SNARK protocol offers the proposed device authentication framework with additional security strength over the conventional counterparts. We analyze and provide proof sketch for the security strength of the proposed framework under three fundamental properties of ZKP, completeness, soundness, and zero-knowledge [29].

• **Machine Learning Attack Resistance Analysis:** As the device fingerprint is proved to the verifier without transmitting it or revealing it through the communication channel, the adversary will not find the secret fingerprint when monitoring the data packets between the device and server. The zero-knowledge property guarantees the proof π_j reveals nothing about the secret witness itself or the PUF response w_h. Moreover, due to the preimage resistance of the hash algorithm, the probability of the attacker's success in finding the preimage w_h from its hash value s_h is low and generally less likely than the probability of hash collision. For the widely adopted MiMC hash construction [306], it has a 256-bits preimage security and a 128-bits collision security. This means the adversary cannot determine the PUF response of a device from its hash. As no response is ever leaked to the attacker during the IoT device authentication, it is not practical to construct a set of known CRPs for training the machine learning algorithm

nor the subsequent attack to predict the unseen PUF response. Thus, it is infeasible for a malicious actor to launch machine learning-based attacks to model the PUF, as no training dataset can be extracted.

• **Proof Verification with Trusted Delegates:** Conventional device authentication schemes have a designated machine (*i.e.,* server) for validating the IoT device. This is due to the fact that the challenge-response pairs of edge devices, along with the public error correction syndromes, are stored in the secure server. Recall that the CRS σ and statement s_h are treated as public parameters, the server can essentially outsource the verification to some trusted parties for performing the necessary computation instead. With the soundness property, a non-polynomial time adversary would be impossible to use randomly generated proofs to convince the trusted delegates (or the server) of the validity of fake proof [31]. For honest prover, the completeness property ensures that its proof can be validated by verification protocol. Therefore, having trusted delegates, it is possible to ease the workload of the central server in response to huge verification queries.

• **Replay Attack Prevention:** The proof validation relies on matching pairing-based computation of hash value and proof. However, considering these proofs are made public after generation, an adversary that passively observes the communication channel could save these valid proofs and reuse them if the edge device was compromised. To prevent this, an extra layer of security must be added to save and check if a proof has been used before, as newly generated proofs will be unique. Therefore, the server may need to save each proof for later comparison to check for proof uniqueness.

8.3 Experimental Results

We have implemented our proposed IoT device authentication protocol with the Raspberry Pi 4 Model B. This device is commonly used in IoT applications today and is one of the most flexible microcontrollers on the market. Proof verification is done on a Ubuntu laptop, acting as the central server. Our proposed authentication protocol is built on top of the ZPiE library [307, 308] with Groth'16 zk-SNARK [31]. Figure 8.2 shows the setup of our implementation, where

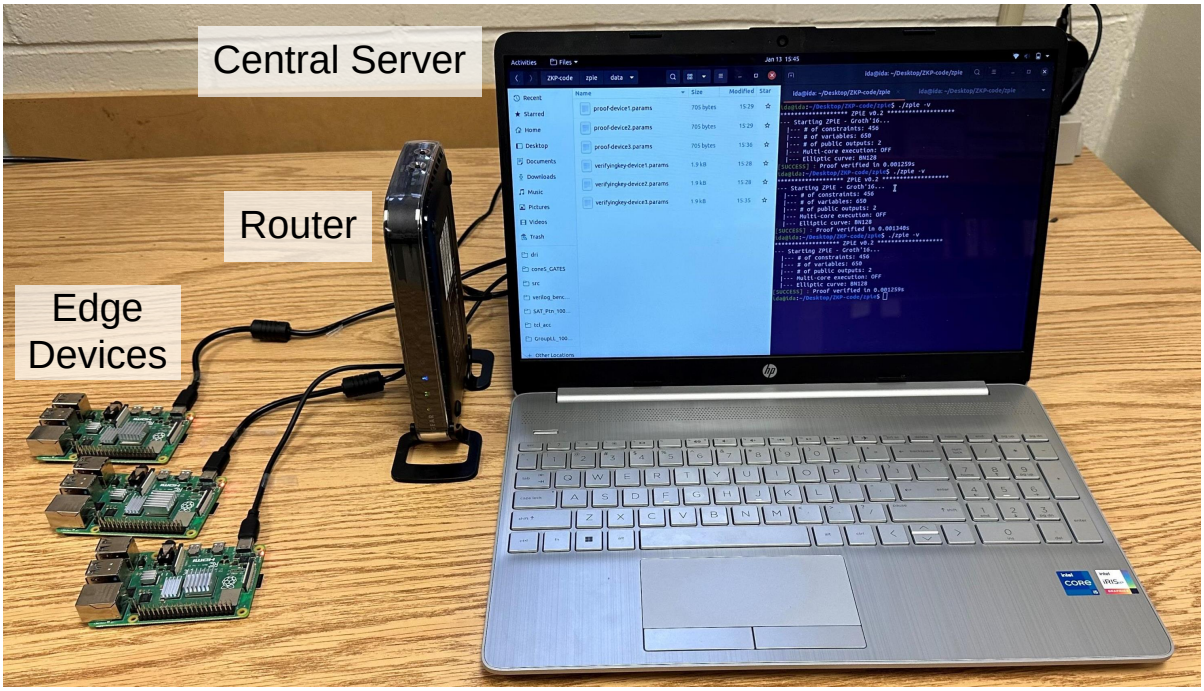

Figure 8.2: Implementation setup with Raspberry Pi 4 Model B as edge devices and laptop as the central server.

edge devices and the central server communicate through a wireless network. The edge device creates a proof once it periodically receives the server's request for its authentication.

8.3.1 Performance Analysis

Although zk-SNARK proof generation has a relatively high computational workload than proof verification, it only needs to be performed sparingly to be used effectively. Note that the zk-SNARK is designed to allow the prover to convince the verifier of the knowledge of its secret in a single proof, due to the negligible success of a dishonest prover cheating with a false statement under a sound zk-SNARK protocol [31]. For edge devices already deployed in the field, monitoring with regular authentication is needed to ensure the integrity of the device. In our proposed authentication scheme, the proof generation is performed on device startup to simulate a periodic yet moderately frequent device verification schedule. As the device would likely be powered off during any form of replacement, this schedule would check at each possible window while not using extremely high amounts of computational power. Figure 8.3 shows the distributions of both proof generation and verification with box plots. Over 60 tests on the Raspberry Pi 4 Model B using ZPiE's single-threaded setting, the average proof generation time

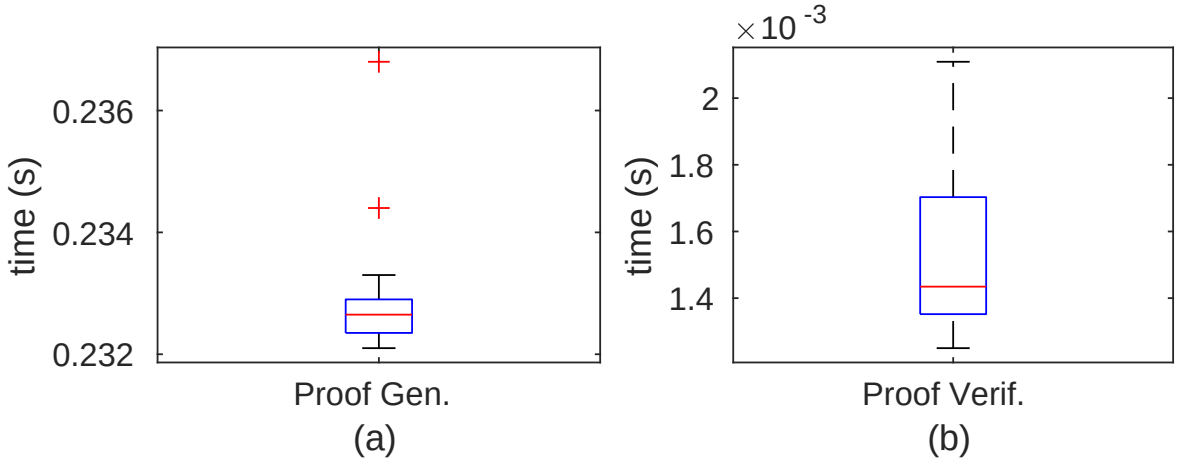

Figure 8.3: Box plots of (a) proof generation time by Raspberry Pi as IoT devices, (b) proof verification time by a laptop as a server.

is 232.7 ms. Similarly, the average verification time of Groth'16 proofs on the server is 1.54 ms. This advantage of fast proof verification allows the server to compute them frequently without noticeable delays in simultaneously fulfilling verification requests from multiple devices. The computationally lightweight proof verification by the server permits the rapid expansion of enrolled devices in the IoT network as the verification time is scalable to a larger IoT device count. Overall, the verification time is extremely short, and the proving time is also short when considering the scale and frequency with which this protocol would be performed.

8.4 Summary

This chapter presents a novel authentication protocol for IoT devices using the state-of-the-art zk-SNARK algorithm with SRAM PUF as the device fingerprint. Our proposed ZKP-based approach can guarantee the secrecy of the PUF response even if the central server gets compromised. It resists machine learning-based attacks and the attacker's attempts to extrapolate the PUF responses, as the adversary can never observe any challenge-response pair from analyzing the entire communication between the server and a particular edge device. Our proposed scheme can further prevent the potential replay attack or impersonation if the adversary monitors the communication channel. In addition, delegating the verification work to trusted parties is possible when the server is busy with different verification requests. We demonstrated our proposed framework by implementing it on Raspberry Pi 4 Model B boards as edge devices

with ZPiE open-source library. Proofs generated by the IoT devices are sent to the server via a wireless network, and the server can verify them very efficiently within a few milliseconds.

Chapter 9

Conclusions and Future Work

This dissertation focuses on hardware-based attacks and/or solutions in cryptography, hardware security and VLSI testing.

In Chapter 3, we focused on attacking the GIFT-COFB, a NIST Lightweight Cryptography finalist. Due to the similar SPN structure it has with AES, we proposed an attack to extract the keys from GIFT-COFB in $\boldsymbol{O}(2^4)$ key-search complexity. Our attack thwarts the interdependencies of 4-bit key cells in the partial-unrolled structure of GIFT-COFB.

In addition, we proposed novel fault injection attacks on Advanced Encryption Standard (AES). For fault injection attack of AES in Chapter 4, a single fault injected in the completion-indicator register allows a $\boldsymbol{O}(2^8)$ key-search complexity to determine the correct key. The interdependencies of key bytes are fully decoupled in this attack. This attack offers insights into fault injection-based attack combined with algebraic cryptanalysis.

In hardware assurance, we examined secure logic locking techniques, one of the primary methods to prevent IP piracy and IC overproduction. The obfuscation of the inner details of a circuit with secret key and key gates are the primary elements in logic locking. However, SAT attack proposed by Subramanyan et al. [100] effectively breaks the keys of these locked circuits with minutes or even a tiny fraction of a second. The theoretical and analytical justifications behind the efficiency of the SAT attack were explored in this dissertation for the first time (Chapter 5). By examining the conjunctive normal forms (CNFs) stored in the SAT solver, the iterative implicit pruning of incorrect keys is revealed. We demonstrated the average linear iteration complexity in the SAT attack against secure locking schemes, ranging from pre-SAT techniques to post-SAT solutions. In secure logic locking, ATPG-guided fault injection attack

was proposed in Chapter 6, where the average number of faults required to break full key length is linear with respect to the total key length, with worst-case complexity being $\mathcal{K}^2$. For the SAT-resistance SFLL schemes, our proposed attack can break with $\frac{\mathcal{K}\cdot(\mathcal{K}-1)}{2}$ faults in total.

Although the SAT attack is devastating to logic locking solutions in the pre-SAT era, the constructive use of the SAT attack can further boost the fault coverage in VLSI test pattern generation. We demonstrated in Chapter 7 that the SAT attack-based miter construction and fault modeling can identify redundant faults and generate test patterns for hard-to-detect stuck-at faults. Further optimized test pattern sets can be obtained when converting multiple faults inside a single locked circuit.

Finally, we explored the application side of privacy-enhancing technologies (PETs) and how it can benefit the IoT device authentication. In this dissertation, we took a novel perspective to apply PETs, specifically Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK), for protecting on-device PUF secret even in the worst-case scenario of a devastating hack at the central server. The adversary could not learn PUF's unique challenge-response pairs using machine learning-based attacks due to the completeness, soundness, and zero-knowledge properties of zk-SNARK.

9.1 Future Work and Possible Directions

Multiple directions are possible as future research. With the blooming field of 3D integration, solutions providing assurance in the 3D chips' supply chain will certainly be welcomed by the research community and the broader industrial partners. This can range from modular blockchain-based proof-of-concepts and solutions [11] to secure obfuscation designs of chiplets with passive and active defense against malicious adversaries.

Boolean Satisfiability (SAT) is one of the NP problems with interesting applications ranging from mixed integer linear programming, hardware security, model counting, and more. The recent advancement of SAT solvers offers more capabilities to solver complex problems, once proper CNF conversion is done. It is possible to explore test pattern generation with better test compaction by SAT solver alone, without invoking any commercial ATPG tool. As test time per chip plays an important role in manufacturing tests after chip fabrication, keeping a

low number of test patterns, or reduce test time, becomes one of the prime objectives for VLSI testing in addition to achieving the desired fault coverage. We plan to apply random patterns to the fault simulator to detect a majority of the easy-to-detect faults and combining multiple faults in a single locked circuit with SAT-based approach.

Last but not the least, future studies can also be focused on achieving security assurance with the latest development in privacy-enhancing technologies, where the adversaries can corrupt some parties and maliciously obtain partial information, however, the overall system remains secure and they still cannot gain control over the entire digital system. This will be particularly useful even if the attackers possess more sophisticated capability than previous published assumptions, i.e., where some of the adversaries will try to collude together to try to break the hardware assurance. Novel solutions tailored for various stronger adversarial models would likely be desired in the near future.

Bibliography

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[2] National Security Agency, "TEMPEST: A Signal Problem," 1972.

[3] NIST Lightweight Cryptography Standardization Project. `https://csrc.nist.gov/Projects/lightweight-cryptography`.

[4] Y. Zhong and U. Guin, "Chosen-Plaintext Attack on Energy-Efficient Hardware Implementation of GIFT-COFB," in *2022 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 73–76, 2022.

[5] Y. Zhong and U. Guin, "Fault-Injection Based Chosen-Plaintext Attacks on Multicycle AES Implementations," in *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, pp. 443–448, 2022.

[6] Y. Zhong and U. Guin, "Complexity Analysis of the SAT Attack on Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits (TCAD)*, pp. 1–14, 2023.

[7] Y. Zhong, A. Jain, M. T. Rahman, N. Asadizanjani, J. Xie, and U. Guin, "AFIA: ATPG-Guided Fault Injection Attack on Secure Logic Locking," *Journal of Electronic Testing*, pp. 1–20, 2022.

[8] Y. Zhong and U. Guin, "A Comprehensive Test Pattern Generation Approach Exploiting SAT Attack for Logic Locking," *IEEE Transactions on Computers*, pp. 1–13, 2023.

[9] Y. Zhong, J. Hovanes, and U. Guin, "On-Demand Device Authentication using Zero-Knowledge Proofs for Smart Systems," in *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, pp. 569–574, 2023.

[10] J. Hovanes, Y. Zhong, and U. Guin, "A Novel IoT Device Authentication Scheme Using Zero-Knowledge Proofs," in *GOMACTech*, pp. 1–4, 2023.

[11] Y. Zhong, A. Ebrahim, and U. Guin, "A Modular Blockchain Framework for Enabling Supply Chain Provenance," in *IEEE Physical Assurance and Inspection of Electronics (PAINE)*, pp. 1–7, 2023.

[12] S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, "GIFT: A Small Present," in *International Conference on Cryptographic Hardware and Embedded Systems*, pp. 321–345, Springer, 2017.

[13] NIST Internal Report NIST IR 8454: Status Report on the Final Round of the NIST Lightweight Cryptography Standardization Process, NIST, June 2023. `https://doi.org/10.6028/NIST.IR.8454`.

[14] W. Shih, 2022. Intel's $88 billion European expansion is part of a new phase in the globalization of the semiconductor industry, Forbes.

[15] M. Tehranipoor, U. Guin, and D. Forte, *Counterfeit Integrated Circuits: Detection and Avoidance*. Springer International Publishing, 2015.

[16] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor, and Y. Makris, "Counterfeit Integrated Circuits: A Rising Threat in the Global Semiconductor Supply Chain," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1207–1228, 2014.

[17] U. Guin, D. DiMase, and M. Tehranipoor, "Counterfeit Integrated Circuits: Detection, Avoidance, and the Challenges Ahead," *Journal of Electronic Testing*, vol. 30, no. 1, pp. 9–23, 2014.

[18] P. Cui, J. Dixon, U. Guin, and D. DiMase, "A blockchain-based framework for supply chain provenance," *IEEE Access*, vol. 7, pp. 157113–157125, 2019.

[19] U. Guin, W. Wang, C. Harper, and A. D. Singh, "Detecting Recycled SOCs by Exploiting Aging Induced Biases in Memory Cells," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 72–80, 2019.

[20] M. M. Tehranipoor, U. Guin, and S. Bhunia, "Invasion of the hardware snatchers," *IEEE Spectrum*, vol. 54, no. 5, pp. 36–41, 2017.

[21] E. Castillo, U. Meyer-Baese, A. García, L. Parrilla, and A. Lloris, "IPPHDL: Efficient Intellectual Property Protection Scheme for IP Cores," *IEEE Trans. on VLSI (TVLSI)*, pp. 578–591, 2007.

[22] M. Tehranipoor and C. Wang, *Introduction to Hardware Security and Trust*. Springer Science & Business Media, 2011.

[23] U. Guin, Q. Shi, D. Forte, and M. M. Tehranipoor, "FORTIS: A Comprehensive Solution for Establishing Forward Trust for Protecting IPs and ICs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 21, no. 4, pp. 1–20, 2016.

[24] S. Adee, "The Hunt for the Kill Switch," *IEEE Spectrum*, vol. 45, no. 5, pp. 34–39, 2008.

[25] M. Tehranipoor and F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection," *IEEE Design & Test of Computers*, vol. 27, no. 1, 2010.

[26] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware Trojans: Lessons learned after one decade of research," *Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 1, p. 6, 2016.

[27] J. Robertson and M. Riley, "The Big Hack: How China Used a Tiny Chip to Infiltrate U.S. Companies," 2018.

[28] J. Robertson and M. Riley, "The Long Hack: How China Exploited a U.S. Tech Supplier," 2021.

[29] S. Goldwasser, S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof-Systems," in *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pp. 291–304, 1985.

[30] O. Goldreich, S. Micali, and A. Wigderson, "Proofs That Yield Nothing but Their Validity or All Languages in NP Have Zero-knowledge Proof Systems," Journal of ACM, pp. 690–728, July 1991.

[31] J. Groth, "On the size of pairing-based non-interactive arguments," in *International Conference on Theory and Applications of Cryptographic Techniques*, pp. 305–326, Springer, 2016.

[32] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *IEEE Symposium on Security and Privacy (SP)*, pp. 315–334, 2018.

[33] L. Bassham, Ç. Çalık, K. McKay, and M. S. Turan, "Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process," *US National Institute of Standards and Technology*, 2018.

[34] NIST CSRC, Finalists - Lightweight Cryptography, `https://csrc.nist.gov/Projects/lightweight-cryptography/finalists`.

[35] S. Banik, A. Chakraborti, T. Iwata, K. Minematsu, M. Nandi, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, "GIFT-COFB," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 738, 2020.

[36] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "Present: An ultra-lightweight block cipher," in *International workshop on cryptographic hardware and embedded systems*, pp. 450–466, Springer, 2007.

[37] J. Nakahara, P. Sepehrdad, B. Zhang, and M. Wang, "Linear (Hull) and Algebraic Cryptanalysis of the Block Cipher PRESENT," in *International Conference on Cryptology and Network Security*, pp. 58–75, Springer, 2009.

[38] C. Reinbrecht, A. Aljuffri, S. Hamdioui, M. Taouil, and J. Sepúlveda, "GRINCH: A Cache Attack against GIFT Lightweight Cipher," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 549–554, IEEE, 2021.

[39] S. Patranabis, N. Datta, D. Jap, J. Breier, S. Bhasin, and D. Mukhopadhyay, "SCADFA: Combined SCA+DFA Attacks on Block Ciphers with Practical Validations," *IEEE Transactions on Computers*, vol. 68, no. 10, pp. 1498–1510, 2019.

[40] H. Luo, W. Chen, X. Ming, and Y. Wu, "General differential fault attack on present and gift cipher with nibble," *IEEE Access*, vol. 9, pp. 37697–37706, 2021.

[41] A. Inoue and K. Minematsu, "GIFT-COFB is Tightly Birthday Secure with Encryption Queries," *Cryptology ePrint Archive*, 2021.

[42] M. Khairallah, "Observations on the tightness of the security bounds of gift-cofb and hyena.," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 1463, 2020.

[43] M. Khairallah, "Security of cofb against chosen ciphertext attacks," tech. rep., Cryptology ePrint Archive, Report 2021/648, 2021.

[44] A. Aghaie, A. Moradi, S. Rasoolzadeh, A. R. Shahmirzadi, F. Schellenberg, and T. Schneider, "Impeccable circuits," *IEEE Transactions on Computers*, vol. 69, no. 3, pp. 361–376, 2019.

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

[46] R. Torrance and D. James, "The State-of-the-Art in IC Reverse Engineering," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 363–381, 2009.

[47] Energy Analysis of Lightweight AEAD Circuit, https://github.com/qantik/Energy-Analysis-of-Lightweight-AEAD-Circuit.

[48] A. Caforio, F. Balli, and S. Banik, "Energy Analysis of Lightweight AEAD Circuits," in *International Conference on Cryptology and Network Security*, pp. 23–42, Springer, 2020.

[49] Chosen Plaintext Attack on Energy-Efficient Hardware Implementation of GIFT-COFB, https://github.com/yadi14/cpa-energy-efficient-gift-cofb.

[50] V. Rijmen and J. Daemen, "Advanced encryption standard," *Proceedings of Federal Information Processing Standards Publications, National Institute of Standards and Technology*, pp. 19–22, 2001.

[51] The Official NVIDIA Blog, "Nvidia's new CPU to 'grace' world's most powerful AI-capable supercomputer, https://blogs.nvidia.com/blog/2021/04/12/cpu-grace-cscs-alps/," Apr 2021.

[52] Apple Inc., "Apple unleashes M1, https://www.apple.com/newsroom/2020/11/apple-unleashes-m1/," Oct 2021.

[53] Intel Corp., "Intel Advanced Encryption Standard New Instructions (AES-NI), https://www.intel.com/content/www/us/en/developer/articles/technical/advanced-encryption-standard-instructions-aes-ni.html," 2012.

[54] Apple Inc., "Apple T2 Security Chip: Security Overview, https://www.apple.com/ca/mac/docs/Apple_T2_Security_Chip_Overview.pdf," Oct 2018.

[55] Unit 42 IoT Threat Report, https://unit42.paloaltonetworks.com/iot-threat-report-2020/, 2020.

[56] OpenCores, hrefhttps://www.opencores.orghttps://www.opencores.org.

[57] N. T. Courtois and J. Pieprzyk, "Cryptanalysis of block ciphers with overdefined systems of equations," in *International conference on the theory and application of cryptology and information security*, pp. 267–287, Springer, 2002.

[58] A. Bar-On, O. Dunkelman, N. Keller, E. Ronen, and A. Shamir, "Improved key recovery attacks on reduced-round aes with practical data and memory complexities," *Journal of Cryptology*, vol. 33, no. 3, pp. 1003–1043, 2020.

[59] C. Bouillaguet, P. Derbez, O. Dunkelman, P.-A. Fouque, N. Keller, and V. Rijmen, "Low data complexity attacks on aes," *IEEE transactions on information theory*, vol. 58, no. 11, pp. 7002–7017, 2012.

[60] O. Dunkelman and N. Keller, "The effects of the omission of last round's mixcolumns on aes," *Information Processing Letters*, vol. 110, no. 8-9, pp. 304–308, 2010.

[61] J. Blömer and J.-P. Seifert, "Fault based cryptanalysis of the advanced encryption standard (AES)," in *International Conference on Financial Cryptography*, pp. 162–181, Springer, 2003.

[62] A. Moradi, M. T. M. Shalmani, and M. Salmasizadeh, "A generalized method of differential fault attack against AES cryptosystem," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 91–100, Springer, 2006.

[63] C. H. Kim, "Improved differential fault analysis on AES key schedule," *IEEE transactions on information forensics and security*, vol. 7, no. 1, pp. 41–50, 2011.

[64] S. S. Ali and D. Mukhopadhyay, "A differential fault analysis on aes key schedule using single fault," in *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 35–42, IEEE, 2011.

[65] C. Giraud, "DFA on AES," in *International Conference on Advanced Encryption Standard*, pp. 27–41, Springer, 2004.

[66] A. Singh, M. Kar, S. K. Mathew, A. Rajan, V. De, and S. Mukhopadhyay, "Improved power/em side-channel attack resistance of 128-bit aes engines with random fast voltage dithering," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 2, pp. 569–583, 2018.

[67] S. Mangard, "A simple power-analysis (spa) attack on implementations of the aes key expansion," in *International Conference on Information Security and Cryptology*, pp. 343–358, Springer, 2002.

[68] Charles Bouillaguet, "`https://www-almasty.lip6.fr/~bouillaguet/static/attacks/aes_1r_5_guesses.tar.gz`," 2011.

[69] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures," *Proceedings of the IEEE*, pp. 3056–3076, 2012.

[70] T. E. Pogue and N. Nicolici, "Incremental fault analysis: Relaxing the fault model of differential fault attacks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 3, pp. 750–763, 2019.

[71] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "A lightweight concurrent fault detection scheme for the aes s-boxes using normal basis," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 113–129, Springer, 2008.

[72] W. Jiang, L. Wen, J. Zhan, and K. Jiang, "Design optimization of confidentiality-critical cyber physical systems with fault detection," *Journal of Systems Architecture*, vol. 107, p. 101739, 2020.

[73] S. Azzi, B. Barras, M. Christofi, and D. Vigilant, "Using linear codes as a fault countermeasure for nonlinear operations: application to aes and formal verification," *Journal of Cryptographic Engineering*, vol. 7, no. 1, pp. 75–85, 2017.

[74] S. Sheikhpour, S.-B. Ko, and A. Mahani, "A low cost fault-attack resilient aes for iot applications," *Microelectronics Reliability*, vol. 123, p. 114202, 2021.

[75] A. Lasheras, R. Canal, E. Rodríguez, and L. Cassano, "Lightweight protection of cryptographic hardware accelerators against differential fault analysis," in *2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pp. 1–6, IEEE, 2020.

[76] H. Mestiri, N. Benhadjyoussef, and M. Machhout, "Fault attacks resistant aes hardware implementation," in *2019 IEEE International Conference on Design & Test of Integrated Micro & Nano-Systems (DTS)*, pp. 1–6, IEEE, 2019.

[77] M. Bushnell and V. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*, vol. 17. Springer Science & Business Media, 2004.

[78] M. T. Rahman, S. Tajik, M. S. Rahman, M. Tehranipoor, and N. Asadizanjani, "The Key is Left under the Mat: On the Inappropriate Security Assumption of Logic Locking Schemes," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 262–272, 2020.

[79] OpenCores, "AES-VHDL, `https://opencores.org/projects/aes`," 2019.

[80] OpenCores, "AES Encryption All Keylength, `https://opencores.org/projects/aes_all_keylength`," 2013.

[81] OpenCores, "High Throughput and Low Area AES Core, `https://opencores.org/projects/aes_highthroughput_lowarea`," 2010.

[82] OpenCores, "AES128, `https://opencores.org/projects/aes_crypto_core`," 2002.

[83] Chosen Plaintext Attack on Multicycle AES, `https://github.com/yadi14/cpa-multicycle-AES`, 2021.

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

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

[86] R. S. Chakraborty and S. Bhunia, "Hardware Protection and Authentication Through Netlist Level Obfuscation," in *Proc. of IEEE/ACM International Conference on Computer-Aided Design*, pp. 674–677, 2008.

[87] Y. Alkabani, F. Koushanfar, and M. Potkonjak, "Remote Activation of ICs for Piracy Prevention and Digital Right Management," in *International Conference on Computer-Aided design*, pp. 674–677, 2007.

[88] J. Huang and J. Lach, "IC Activation and User Authentication for Security-Sensitive Systems," in *IEEE International Workshop on Hardware-Oriented Security and Trust*, pp. 76–80, 2008.

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

[90] U. Guin, Q. Shi, D. Forte, and M. M. Tehranipoor, "FORTIS: a comprehensive solution for establishing forward trust for protecting IPs and ICs," *Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 21, no. 4, p. 63, 2016.

[91] M. Tehranipoor, U. Guin, and D. Forte, *Counterfeit Integrated Circuits: Detection and Avoidance*. Springer International Publishing, 2015.

[92] S. Bhunia and M. Tehranipoor, *Hardware Security: A Hands-on Learning Approach*. Morgan Kaufmann, 2018.

[93] R. Torrance and D. James, "Reverse Engineering in the Semiconductor Industry," in *IEEE Custom ICs Conference*, pp. 429–436, 2007.

[94] R. Torrance and D. James, "The state-of-the-art in semiconductor reverse engineering," in *Proc. of the Design Automation Conference*, pp. 333–338, 2011.

[95] E. Charbon, "Hierarchical watermarking in IC design," in *Proc. of the IEEE Custom Integrated Circuits Conference*, pp. 295–298, 1998.

[96] A. B. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Constraint-based watermarking techniques for design IP protection," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 20, no. 10, pp. 1236–1252, 2001.

[97] G. Qu and M. Potkonjak, *Intellectual Property Protection in VLSI Designs: Theory and Practice*. Springer Sc. & Business Media, 2007.

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

[99] J. Rajendran, H. Zhang, C. Zhang, G. S. Rose, Y. Pino, O. Sinanoglu, and R. Karri, "Fault Analysis-Based Logic Encryption," *IEEE Transactions on computers*, vol. 64, no. 2, pp. 410–424, 2015.

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

[101] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," in *International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 236–241, 2016.

[102] Y. Xie and A. Srivastava, "Anti-SAT: Mitigating SAT Attack on Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 2, pp. 199–207, 2019.

[103] K. Shamsi, T. Meade, M. Li, D. Z. Pan, and Y. Jin, "On the approximation resiliency of logic locking and ic camouflaging schemes," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 2, pp. 347–359, 2018.

[104] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu, "TTLock: Tenacious and traceless logic locking," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 166–166, 2017.

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

[106] A. Sengupta, M. Nabeel, N. Limaye, M. Ashraf, and O. Sinanoglu, "Truly stripping functionality for logic locking: A fault-based perspective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4439–4452, 2020.

[107] B. Shakya, X. Xu, M. Tehranipoor, and D. Forte, "Cas-lock: A security-corruptibility trade-off resilient logic locking scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 175–202, 2020.

[108] M. Yasin, A. Sengupta, B. C. Schafer, Y. Makris, O. Sinanoglu, and J. J. Rajendran, "What to Lock?: Functional and Parametric Locking," in *Proc. of Great Lakes Symposium on VLSI*, pp. 351–356, 2017.

[109] J. Zhou and X. Zhang, "Generalized sat-attack-resistant logic locking," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2581–2592, 2021.

[110] Y. Liu, M. Zuzak, Y. Xie, A. Chakraborty, and A. Srivastava, "Strong anti-sat: Secure and effective logic locking," in *21st International Symposium on Quality Electronic Design (ISQED)*, pp. 199–205, 2020.

[111] H. Zhou, "A humble theory and application for logic encryption," *Cryptology ePrint Archive*, 2017.

[112] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "Cyclic obfuscation for creating sat-unresolvable circuits," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pp. 173–178, 2017.

[113] S. Roshanisefat, H. Mardani Kamali, and A. Sasan, "Srclock: Sat-resistant cyclic logic locking for protecting the hardware," in *Proceedings of 2018 Great Lakes Symposium on VLSI*, pp. 153–158, 2018.

[114] A. Rezaei, Y. Li, Y. Shen, S. Kong, and H. Zhou, "Cycsat-unresolvable cyclic logic encryption using unreachable states," in *Proc. of the 24th Asia and South Pacific Design Automation Conference*, pp. 358–363, 2019.

[115] S. Roshanisefat, H. M. Kamali, H. Homayoun, and A. Sasan, "Sat-hard cyclic logic obfuscation for protecting the ip in the manufacturing supply chain," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 4, pp. 954–967, 2020.

[116] A. Rezaei, Y. Shen, S. Kong, J. Gu, and H. Zhou, "Cyclic locking and memristor-based obfuscation against cycsat and inside foundry attacks," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 85–90, 2018.

[117] X.-M. Yang, P.-P. Chen, H.-Y. Chiang, C.-C. Lin, Y.-C. Chen, and C.-Y. Wang, "Looplock 2.0: An enhanced cyclic logic locking approach," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 1, pp. 29–34, 2021.

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

[119] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Full-lock: Hard distributions of sat instances for obfuscating circuits using fully configurable logic and routing blocks," in *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

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

[121] S. D. Chowdhury, G. Zhang, Y. Hu, and P. Nuzzo, "Enhancing SAT-attack resiliency and cost-effectiveness of reconfigurable-logic-based circuit obfuscation," in *International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2021.

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

[123] S. Patnaik, M. Ashraf, O. Sinanoglu, and J. Knechtel, "Obfuscating the interconnects: Low-cost and resilient full-chip layout camouflaging," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4466–4481, 2020.

[124] J. Sweeney, M. J. Heule, and L. Pileggi, "Modeling techniques for logic locking," in *International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, 2020.

[125] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Interlock: An intercorrelated logic and routing locking," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, 2020.

[126] D. Zhang, M. He, X. Wang, and M. Tehranipoor, "Dynamically obfuscated scan for protecting IPs against scan-based attacks throughout supply chain," in *VLSI Test Symposium (VTS)*, pp. 1–6, 2017.

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

[128] R. Karmakar, H. Kumar, and S. Chattopadhyay, "Efficient key-gate placement and dynamic scan obfuscation towards robust logic encryption," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 2109–2124, 2019.

[129] S. Potluri, A. Aysu, and A. Kumar, "SeqL: Secure Scan-Locking for IP Protection," *arXiv preprint arXiv:2005.13032*, 2020.

[130] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Scramble: The state, connectivity and routing augmentation model for building logic encryption," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 153–159, 2020.

[131] M. S. Rahman, A. Nahiyan, F. Rahman, S. Fazzari, K. Plaks, F. Farahmandi, D. Forte, and M. Tehranipoor, "Security assessment of dynamically obfuscated scan chain against oracle-guided attacks," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 26, no. 4, pp. 1–27, 2021.

[132] F. Koushanfar, "Active hardware metering by finite state machine obfuscation," in *Hardware Protection through Obfuscation*, pp. 161–187, Springer, 2017.

[133] J. Dofe and Q. Yu, "Novel dynamic state-deflection method for gate-level design obfuscation," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 273–285, 2017.

[134] T. Meade, Z. Zhao, S. Zhang, D. Pan, and Y. Jin, "Revisit sequential logic obfuscation: Attacks and defenses," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, 2017.

[135] S. Roshanisefat, H. M. Kamali, K. Z. Azar, S. M. P. Dinakarrao, N. Karimi, H. Homayoun, and A. Sasan, "Dfssd: Deep faults and shallow state duality, a provably strong obfuscation solution for circuits with restricted access to scan chain," in *IEEE 38th VLSI Test Symposium (VTS)*, pp. 1–6, 2020.

[136] L. Li, S. Ni, and A. Orailoglu, "Janus: Boosting logic obfuscation scope through reconfigurable fsm synthesis," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 292–303, 2021.

[137] L. Li and A. Orailoglu, "Janus-hd: exploiting fsm sequentiality and synthesis flexibility in logic obfuscation to thwart sat attack while offering strong corruption," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1323–1328, 2022.

[138] Y. Xie and A. Srivastava, "Delay Locking: Security Enhancement of Logic Locking against IC Counterfeiting and Overproduction," in *Proceedings of the 54th Design Automation Conference*, pp. 1–6, 2017.

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

[140] M. Alam, S. Ghosh, and S. S. Hosur, "TOIC: Timing Obfuscated Integrated Circuits," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, pp. 105–110, 2019.

[141] J. Sweeney, M. Zackriya V, S. Pagliarini, and L. Pileggi, "Latch-Based Logic Locking," *arXiv preprint arXiv:2005.10649*, 2020.

[142] K. Z. Azar, H. M. Kamali, S. Roshanisefat, H. Homayoun, C. P. Sotiriou, and A. Sasan, "Data flow obfuscation: A new paradigm for obfuscating circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 4, pp. 643–656, 2021.

[143] M. S. Rahman, R. Guo, H. M. Kamali, F. Rahman, F. Farahmandi, and M. Abdel-Moneum, "O'clock: Lock the clock via clock-gating for soc ip protection," in *Design Automation Conference (DAC)*, pp. 1–6, 2022.

[144] C. Pilato, F. Regazzoni, R. Karri, and S. Garg, "Tao: Techniques for algorithm-level obfuscation during high-level synthesis," in *Proceedings of the 55th Annual Design Automation Conference*, pp. 1–6, 2018.

[145] C. Pilato, A. B. Chowdhury, D. Sciuto, S. Garg, and R. Karri, "Assure: Rtl locking against an untrusted foundry," *IEEE Transactions on VLSI Systems*, vol. 29, no. 7, pp. 1306–1318, 2021.

[146] M. Zuzak, Y. Liu, and A. Srivastava, "A resource binding approach to logic obfuscation," in *Design Automation Conference*, pp. 235–240, 2021.

[147] M. R. Muttaki, R. Mohammadivojdan, M. Tehranipoor, and F. Farahmandi, "Hlock: Locking ips at the high-level language," in *58th ACM/IEEE Design Automation Conference (DAC)*, pp. 79–84, 2021.

[148] N. Limaye, A. B. Chowdhury, C. Pilato, M. T. Nabeel, O. Sinanoglu, S. Garg, and R. Karri, "Fortifying rtl locking against oracle-less (untrusted foundry) and oracle-guided attacks," in *58th ACM/IEEE Design Automation Conference (DAC)*, pp. 91–96, 2021.

[149] C. Karfa, T. A. Khader, Y. Nigam, R. Chouksey, and R. Karri, "HOST: HLS Obfuscations against SMT ATtack," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 32–37, 2021.

[150] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal attacks on logic locking and camouflaging techniques," *Transactions on Emerging Topics in Computing*, 2017.

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

[152] X. Xu, B. Shakya, M. M. Tehranipoor, and D. Forte, "Novel bypass attack and BDD-based tradeoff analysis against all known logic locking attacks," in *International Conference on Cryptographic Hardware and Embedded Syst.*, 2017.

[153] Y. Shen and H. Zhou, "Double DIP: Re-Evaluating Security of Logic Encryption Algorithms," in *Proc. of the GLSVLSI*, pp. 179–184, 2017.

[154] Y. Shen, A. Rezaei, and H. Zhou, "SAT-based bit-flipping attack on logic encryptions," in *Design, Automation & Test in Europe Conference& Exhibition (DATE)*, pp. 629–632, 2018.

[155] D. Sirone and P. Subramanyan, "Functional Analysis Attacks on Logic Locking," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 936–939, 2019.

[156] Y. Zhang, P. Cui, Z. Zhou, and U. Guin, "TGA: An Oracle-less and Topology-Guided Attack on Logic Locking," in *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop*, pp. 75–83, 2019.

[157] A. Jain, M. T. Rahman, and U. Guin, "ATPG-Guided Fault Injection Attacks on Logic Locking," in *IEEE Physical Assurance and Inspection of Electronics (PAINE)*, pp. 1–6, 2020.

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

[159] N. Limaye, S. Patnaik, and O. Sinanoglu, "Fa-SAT: Fault-aided SAT-based Attack on Compound Logic Locking Techniques," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1166–1171, 2021.

[160] A. Sengupta, N. Limaye, and O. Sinanoglu, "Breaking CAS-Lock and Its Variants by Exploiting Structural Traces," *Cryptology ePrint Archive*, 2021.

[161] L. Alrahis, S. Patnaik, F. Khalid, M. A. Hanif, H. Saleh, M. Shafique, and O. Sinanoglu, "GNNUnlock: Graph Neural Networks-based Oracle-less Unlocking Scheme for Provably Secure Logic Locking," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 780–785, 2021.

[162] Z. Han, M. Yasin, and J. J. Rajendran, "Does logic locking work with {EDA} tools?," in *USENIX Security Symposium*, pp. 1055–1072, 2021.

[163] N. Limaye, S. Patnaik, and O. Sinanoglu, "Valkyrie: Vulnerability assessment tool and attack for provably-secure logic locking techniques," *IEEE Trans. on Inf. Forensics and Secur.*, vol. 17, pp. 744–759, 2022.

[164] D. Duvalsaint, X. Jin, B. Niewenhuis, and R. Blanton, "Characterization of Locked Combinational Circuits via ATPG," in *International Test Conference (ITC)*, pp. 1–10, 2019.

[165] K. Shamsi, D. Z. Pan, and Y. Jin, "On the Impossibility of Approximation-Resilient Circuit Locking," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 161–170, 2019.

[166] K. Shamsi and Y. Jin, "In Praise of Exact-Functional-Secrecy in Circuit Locking," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 5225–5238, 2021.

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

[168] Y. Shen, Y. Li, A. Rezaei, S. Kong, D. Dlott, and H. Zhou, "BeSAT: Behavioral SAT-based attack on cyclic logic encryption," in *Proc. of the 24th ASP Design Automation Conference*, pp. 657–662, 2019.

[169] K. Shamsi, D. Z. Pan, and Y. Jin, "Icysat: Improved sat-based attacks on cyclic locked circuits," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–7, 2019.

[170] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "NNgSAT: Neural Network guided SAT Attack on Logic Locked Complex Structures," in *International Conference Comput.-Aided Des. (ICCAD)*, pp. 1–9, 2020.

[171] L. Alrahis, S. Patnaik, M. A. Hanif, M. Shafique, and O. Sinanoglu, "Untangle: unlocking routing and logic obfuscation using graph neural networks-based link prediction," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, 2021.

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

[173] N. Limaye, A. Sengupta, M. Nabeel, and O. Sinanoglu, "Is Robust Design-for-Security Robust Enough? Attack on Locked Circuits with Restricted Scan Chain Access," *arXiv preprint:1906.07806*, 2019.

[174] N. Limaye and O. Sinanoglu, "Dynunlock: Unlocking scan chains obfuscated using dynamic keys," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 270–273, 2020.

[175] M. El Massad, S. Garg, and M. Tripunitara, "Reverse engineering camouflaged sequential circuits without scan access," in *International Conference on Computer-Aided Design (ICCAD)*, pp. 33–40, 2017.

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

[177] S. Roshanisefat, H. Mardani Kamali, H. Homayoun, and A. Sasan, "Rane: An open-source formal de-obfuscation attack for reverse engineering of logic encrypted circuits," in *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, pp. 221–228, 2021.

[178] K. Azar, H. Kamali, F. Farahmandi, and M. Tehranipoor, "Warm Up before Circuit Deobfuscation? An Exploration through Bounded-Model-Checkers," in *International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 1–4, 2022.

[179] Y. Hu, Y. Zhang, K. Yang, D. Chen, P. A. Beerel, and P. Nuzzo, "Fun-SAT: Functional corruptibility-guided SAT-based attack on sequential logic encryption," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 281–291, 2021.

[180] A. Saha, H. Banerjee, R. S. Chakraborty, and D. Mukhopadhyay, "Oracall: an oracle-based attack on cellular automata guided logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 12, pp. 2445–2454, 2021.

[181] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "SMT Attack: Next Generation Attack on Obfuscated Circuits with Capabilities and Performance Beyond the SAT Attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 97–122, 2019.

[182] A. Chakraborty, Y. Liu, and A. Srivastava, "Timingsat: Timing profile embedded sat attack," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–6, ACM, 2018.

[183] P. Chakraborty, J. Cruz, and S. Bhunia, "SURF: Joint Structural Functional Attack on Logic Locking," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 181–190, 2019.

[184] P. Chakraborty, J. Cruz, A. Alaql, and S. Bhunia, "SAIL: Analyzing Structural Artifacts of Logic Locking Using Machine Learning," *IEEE Transaction on Information Forensics and Security*, vol. 16, pp. 3828–3842, 2021.

[185] D. Sisejkovic, F. Merchant, L. M. Reimann, H. Srivastava, A. Hallawa, and R. Leupers, "Challenging the Security of Logic Locking Schemes in the Era of Deep Learning: A Neuroevolutionary Approach," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 17, no. 3, pp. 1–26, 2021.

[186] L. Alrahis, S. Patnaik, M. Shafique, and O. Sinanoglu, "OMLA: An Oracle-Less Machine Learning-Based Attack on Logic Locking," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 3, pp. 1602–1606, 2021.

[187] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification," *IEEE Transactions on*

Computer-Aided Design of Integrated Circuits and Systems, vol. 21, no. 12, pp. 1377–1394, 2002.

[188] Y. Xie and A. Srivastava, "Anti-sat: Mitigating sat attack on logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 2, pp. 199–207, 2018.

[189] Synopsys Design Compiler, Synopsys, Inc., 2017.

[190] MATLAB flipedge function, https://www.mathworks.com/help/matlab/ref/digraph.flipedge.html.

[191] MATLAB bfsearch function, https://www.mathworks.com/help/matlab/ref/graph.bfsearch.html.

[192] K. Juretus and I. Savidis, "Increasing the SAT Attack Resiliency of In-Cone Logic Locking," in *International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2019.

[193] H. L. Royden and P. Fitzpatrick, *Real Analysis*, vol. 32. Macmillan New York, 1988.

[194] V. Ganesh and M. Y. Vardi, "On the Unreasonable Effectiveness of SAT Solvers," *Rice University*, 2020.

[195] Age Yeh, "Trends in the global IC design service market ." DIGITIMES Research, 2012. [online] Available at: `https://www.digitimes.com/news/a20120313RS400.html&chid=2`.

[196] R. W. Jarvis and M. G. McIntyre, "Split manufacturing method for advanced semiconductor circuits," 2007. US Patent 7,195,931.

[197] J. Lee, M. Tebranipoor, and J. Plusquellic, "A low-cost solution for protecting IPs against scan-based side-channel attacks," in *24th IEEE VLSI Test Symposium*, pp. 6–pp, 2006.

[198] Y. Xie and A. Srivastava, "Anti-SAT: Mitigating SAT Attack on Logic Locking," in *International Conference on Cryptographic Hardware and Embedded Systems*, pp. 127–146, 2016.

[199] X. Wang, D. Zhang, M. He, D. Su, and M. Tehranipoor, "Secure Scan and Test Using Obfuscation Throughout Supply Chain," *Trans. on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1867–1880, 2017.

[200] U. Guin, Z. Zhou, and A. Singh, "Robust design-for-security architecture for enabling trust in IC manufacturing and test," *Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 5, pp. 818–830, 2018.

[201] A. Sengupta, M. Ashraf, M. Nabeel, and O. Sinanoglu, "Customized Locking of IP Blocks on a Multi-Million-Gate SoC," in *International Conference on Computer-Aided Design (ICCAD)*, pp. 1–7, 2018.

[202] K. Shamsi, M. Li, K. Plaks, S. Fazzari, D. Z. Pan, and Y. Jin, "IP Protection and Supply Chain Security through Logic Obfuscation: A Systematic Overview," *Trans. on Design Automation of Electronic Systems (TODAES)*, p. 65, 2019.

[203] D. Duvalsaint, Z. Liu, A. Ravikumar, and R. D. Blanton, "Characterization of locked sequential circuits via ATPG," in *2019 IEEE International Test Conference in Asia (ITC-Asia)*, pp. 97–102, IEEE, 2019.

[204] H. M. Kamali, K. Z. Azar, F. Farahmandi, and M. Tehranipoor, "Advances in Logic Locking: Past, Present, and Prospects," *Cryptology ePrint Archive*, 2022.

[205] M. T. Rahman, M. S. Rahman, H. Wang, S. Tajik, W. Khalil, F. Farahmandi, D. Forte, N. Asadizanjani, and M. Tehranipoor, "Defense-in-depth: A recipe for logic locking to prevail," *Integration*, 2020.

[206] Y. Zhang, A. Jain, P. Cui, Z. Zhou, and U. Guin, "A novel topology-guided attack and its countermeasure towards secure logic locking," *Journal of Cryptographic Engineering*, pp. 1–14, 2020.

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

[208] J. Zhang, F. Yuan, L. Wei, Y. Liu, and Q. Xu, "VeriTrust: Verification for Hardware Trust," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1148–1161, 2015.

[209] H. Shen, N. Asadizanjani, M. Tehranipoor, and D. Forte, "Nanopyramid: An Optical Scrambler Against Backside Probing Attacks," in *Proc. International Symposium for Testing and Failure Analysis(ISTFA)*, p. 280, 2018.

[210] N. Vashistha, H. Lu, Q. Shi, M. T. Rahman, H. Shen, D. L. Woodard, N. Asadizanjani, and M. Tehranipoor, "Trojan Scanner: Detecting Hardware Trojans with Rapid SEM Imaging combined with Image Processing and Machine Learning," in *Proceedings of International Symposium for Testing and Failure Analysis (ISTFA)*, p. 256, 2018.

[211] D. Sisejkovic, F. Merchant, L. M. Reimann, and R. Leupers, "Deceptive logic locking for hardware integrity protection against machine learning attacks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[212] L. Alrahis, S. Patnaik, J. Knechtel, H. Saleh, B. Mohammad, M. Al-Qutayri, and O. Sinanoglu, "UNSAIL: Thwarting oracle-less machine learning attacks on logic locking," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2508–2523, 2021.

[213] M. J. Dworkin, E. B. Barker, J. R. Nechvatal, J. Foti, L. E. Bassham, E. Roback, J. F. Dray Jr, *et al.*, "Advanced Encryption Standard (AES)," 2001.

[214] C. Paar and J. Pelzl, *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media, 2009.

[215] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending Piracy of Integrated Circuits," *Computer*, pp. 30–38, 2010.

[216] U. Guin, Z. Zhou, and A. Singh, "A Novel Design-for-Security (DFS) Architecture to Prevent Unauthorized IC Overproduction," in *VLSI Test Symposium (VTS)*, pp. 1–6, 2017.

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

[218] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "From Cryptography to Logic Locking: A Survey on the Architecture Evolution of Secure Scan Chains," *IEEE Access*, vol. 9, pp. 73133–73151, 2021.

[219] SFLL_rem. `https://github.com/micky960/SFLL_fault`.

[220] P. Beerel, M. Georgiou, B. Hamlin, A. J. Malozemoff, and P. Nuzzo, "Towards a formal treatment of logic locking," *Cryptology ePrint Archive*, 2022.

[221] C. E. Shannon, "Communication theory of secrecy systems," *The Bell system technical journal*, vol. 28, no. 4, pp. 656–715, 1949.

[222] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee, "Embedded Deterministic Test," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 5, pp. 776–792, 2004.

[223] TestMAX DFT, Design-for-Test Implementation, Synopsys, `https://www.synopsys.com/content/dam/synopsys/implementation&signoff/datasheets/testmax-dft-ds.pdf`.

[224] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," in *International Conference on the theory and applications of cryptographic techniques*, pp. 37–51, 1997.

[225] G. Piret and J.-J. Quisquater, "A differential fault attack technique against SPN structures, with application to the AES and KHAZAD," in *International workshop on cryptographic hardware and embedded systems*, pp. 77–88, 2003.

[226] P. Dusart, G. Letourneux, and O. Vivolo, "Differential Fault Analysis on AES," in *International Conference on Applied Cryptography and Network Security*, pp. 293–306, 2003.

[227] C.-Y. Lee and J. Xie, "High capability and low-complexity: Novel fault detection scheme for finite field multipliers over gf (2 m) based on mspb," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 21–30, 2019.

[228] T. Fukunaga and J. Takahashi, "Practical Fault Attack on a Cryptographic LSI with ISO/IEC 18033-3 Block Ciphers," in *Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 84–92, 2009.

[229] N. Selmane, S. Guilley, and J.-L. Danger, "Practical Setup Time Violation Attacks on AES," in *Seventh European Dependable Computing Conference*, pp. 91–96, 2008.

[230] S. Guilley, L. Sauvage, J.-L. Danger, N. Selmane, and R. Pacalet, "Silicon-level solutions to counteract passive and active attacks," in *Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 3–17, 2008.

[231] A. Barenghi, G. M. Bertoni, L. Breveglieri, and G. Pelosi, "A fault induction technique based on voltage underfeeding with application to attacks against AES and RSA," *Journal of Systems and Software*, pp. 1864–1878, 2013.

[232] A. Barenghi, G. M. Bertoni, L. Breveglieri, M. Pellicioli, and G. Pelosi, "Low voltage fault attacks to AES," in *International Symposium on Hardware-Oriented Security and Trust (HOST)*, pp. 7–12, 2010.

[233] J.-M. Schmidt and M. Hutter, *Optical and EM fault-attacks on CRT-based RSA: Concrete results*. 2007.

[234] A. Dehbaoui, J.-M. Dutertre, B. Robisson, and A. Tria, "Electromagnetic transient faults injection on a hardware and a software implementations of AES," in *Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 7–15, 2012.

[235] S. P. Skorobogatov and R. J. Anderson, "Optical fault induction attacks," in *International workshop on cryptographic hardware and embedded systems*, pp. 2–12, 2002.

[236] S. Skorobogatov, "Optical fault masking attacks," in *Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 23–29, 2010.

[237] G. Canivet, P. Maistri, R. Leveugle, J. Clédière, F. Valette, and M. Renaudin, "Glitch and laser fault attacks onto a secure AES implementation on a SRAM-based FPGA," *Journal of cryptology*, pp. 247–268, 2011.

[238] V. Pouget, A. Douin, D. Lewis, P. Fouillat, G. Foucard, P. Peronnard, V. Maingot, J. Ferron, L. Anghel, R. Leveugle, *et al.*, "Tools and methodology development for pulsed laser fault injection in SRAM-based FPGAs," in *Latin-American Test Workshop (LATW).*, 2007.

[239] B. Selmke, J. Heyszl, and G. Sigl, "Attack on a DFA protected AES by simultaneous laser fault injections," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pp. 36–46, 2016.

[240] H. Wu, D. Ferranti, and L. Stern, "Precise nanofabrication with multiple ion beams for advanced circuit edit," *Microelectronics Reliability*, pp. 1779–1784, 2014.

[241] D. Skarin, R. Barbosa, and J. Karlsson, "GOOFI-2: A tool for experimental dependability assessment," in *IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, pp. 557–562, 2010.

[242] T. Tsai and R. Iyer, "FTAPE-A fault injection tool to measure fault tolerance," in *Computing in Aerospace Conference*, p. 1041, 1995.

[243] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault Injection Techniques and Tools," *Computer*, pp. 75–82, 1997.

[244] Synopsys Inc., Mountain View, CA, USA, "TetraMAX II ATPG: Automatic Test Pattern Generation," 2017.

[245] H. Salmani and M. Tehranipoor, "Trust-Hub," 2018. [Online]. Available: https://trust-hub.org/home.

[246] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*. Computer science, MIT Press, 2009.

[247] Xilinx, "Xilinx Kintex-7 FPGA KC705 Evaluation Kit, https://www.xilinx.com/products/boards-and-kits/ek-k7-kc705-g.html," 2021.

[248] PHEMOS-1000 Emission microscope, HAMAMATSU, https://www.hamamatsu.com/eu/en/product/semiconductor-manufacturing-support-systems/failure-analysis-system/C11222-16.html.

[249] M. T. Rahman and N. Asadizanjani, "Backside Security Assessment of Modern SoCs," in *International Workshop on Microprocessor/SoC Test, Security and Verification (MTV)*, pp. 18–24, 2019.

[250] A. Sengupta, M. Nabeel, M. Yasin, and O. Sinanoglu, "ATPG-based cost-effective, secure logic locking," in *VLSI Test Symposium (VTS)*, pp. 1–6, 2018.

[251] ISO 26262-1:2018, https://www.iso.org/standard/68383.html, 2018.

[252] J. P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method," *IBM journal of Research and Development*, pp. 278–291, 1966.

[253] J. P. Roth, W. G. Bouricius, and P. R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits," *IEEE Trans. on Electronic Comp.*, pp. 567–580, 1967.

[254] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Trans. on Comput.*, vol. 30, no. 03, pp. 215–222, 1981.

[255] H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms," *IEEE Trans. on Comput.*, vol. 32, no. 12, pp. 1137–1144, 1983.

[256] M. H. Schulz, E. Trischler, and T. M. Sarfert, "SOCRATES: A highly efficient automatic test pattern generation system," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 7, no. 1, pp. 126–137, 1988.

[257] S. T. Chakradhar, V. D. Agrawal, and S. G. Rothweiler, "A Transitive Closure Algorithm for Test Generation," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 12, no. 7, pp. 1015–1028, 1993.

[258] T. Larrabee, "Test Pattern Generation Using Boolean Satisfiability," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 11, no. 1, pp. 4–15, 1992.

[259] P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Combinational Test Generation Using Satisfiability," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 15, no. 9, pp. 1167–1176, 1996.

[260] S. Eggersgluss and R. Drechsler, "Improving Test Pattern Compactness in SAT-based ATPG," in *Asian Test Symposium*, pp. 445–452, 2007.

[261] S. Eggersglüß, R. Krenz-Bååth, A. Glowatz, F. Hapke, and R. Drechsler, "A new SAT-based ATPG for generating highly compacted test sets," in *International Symposium on Design and Diagnostics of Electronic Circuits & Systems*, pp. 230–235, IEEE, 2012.

[262] S. Eggersglüß, R. Wille, and R. Drechsler, "Improved SAT-based ATPG: More constraints, better compaction," in *International Conference on Computer-Aided Design (ICCAD)*, pp. 85–90, 2013.

[263] R. Drechsler, S. Eggergluss, G. Fey, A. Glowatz, F. Hapke, J. Schloeffel, and D. Tille, "On Acceleration of SAT-Based ATPG for Industrial Designs," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 27, no. 7, pp. 1329–1333, 2008.

[264] M. Fujita and A. Mishchenko, "Efficient SAT-based ATPG techniques for all multiple stuck-at faults," in *International Test Conference*, pp. 1–10, 2014.

[265] J. Balcarek, P. Fiser, and J. Schmidt, "Techniques for SAT-based constrained test pattern generation," *Microprocessors and Microsystems*, vol. 37, no. 2, pp. 185–195, 2013.

[266] D. Zhang, X. Wang, M. T. Rahman, and M. Tehranipoor, "An On-Chip Dynamically Obfuscated Wrapper for Protecting Supply Chain Against IP and IC Piracies," *IEEE Trans. on VLSI Systems*, vol. 26, no. 11, pp. 2456–2469, 2018.

[267] J. K. Fichte, M. Hecher, and S. Szeider, "A Time Leap Challenge for SAT Solving," in *International Conference on Principles and Practice of Constraint Programming*, pp. 267–285, Springer, 2020.

[268] A. Biere *et al.*, "Lingeling, Plingeling and Treengeling entering the SAT competition 2013," *Pro. of SAT competition*, vol. 2013, p. 1, 2013.

[269] L. Li and A. Orailoglu, "Piercing logic locking keys through redundancy identification," in *Design, Automation & Test in Europe Conference& Exhibition (DATE)*, pp. 540–545, 2019.

[270] A. Biere, M. Heule, and H. van Maaren, *Handbook of satisfiability*, vol. 336. IOS Press, 2021.

[271] IoT Analytics, "State of IoT 2022: Number of Connected IoT Devices Growing 18% to 14.4 Billion Globally, `https://iot-analytics.com/number-connected-iot-devices/`," 2022.

[272] International Data Corporation (IDC), "Future of Industry Ecosystems: Shared Data and Insights, `https://blogs.idc.com/2021/01/06/future-of-industry-ecosystems-shared-data-and-insights/`," 2021.

[273] Cybersecurity and Infrastructure Security Agency (CISA), Insider Threat Mitigation, `https://www.cisa.gov/insider-threat-mitigation`, 2022.

[274] Forbes, "Was The Cybersecurity Crystal Ball Cloudy Or Clear Two Years Ago? https://www.forbes.com/sites/forbestechcouncil/2023/02/09/was-the-cybersecurity-crystal-ball-cloudy-or-clear-two-years-ago/," 2023.

[275] NPR, "U.S. HVAC Firm Reportedly Linked To Target's Data Security Breach, https://www.npr.org/sections/thetwo-way/2014/02/05/272101928/u-s-hvac-firm-reportedly-linked-to-target-s-data-security-breach," 2014.

[276] Bloomberg, "The Big Hack: How China Used a Tiny Chip to Infiltrate U.S. Companies, https://www.bloomberg.com/news/features/2018-10-04/the-big-hack-how-china-used-a-tiny-chip-to-infiltrate-america-s-top-companies," 2018.

[277] N. Karimian, M. Tehranipoor, D. Woodard, and D. Forte, "Unlock your heart: Next generation biometric in resource-constrained healthcare systems and IoT," *IEEE Access*, vol. 7, pp. 49135–49149, 2019.

[278] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *44th ACM/IEEE Design Automation Conference*, pp. 9–14, 2007.

[279] A. Aysu, E. Gulcan, D. Moriyama, P. Schaumont, and M. Yung, "End-to-end design of a puf-based privacy preserving authentication protocol," in *Cryptographic Hardware and Embedded Systems (CHES)*, pp. 556–576, 2015.

[280] W. Che, F. Saqib, and J. Plusquellic, "Puf-based authentication," in *International Conference on Computer-Aided Design (ICCAD)*, pp. 337–344, 2015.

[281] U. Chatterjee, V. Govindan, R. Sadhukhan, D. Mukhopadhyay, R. S. Chakraborty, D. Mahata, and M. M. Prabhu, "Building PUF based authentication and key exchange protocol for IoT without explicit CRPs in verifier database," *IEEE Trans. on Dependable and Secure Computing*, vol. 16, no. 3, pp. 424–437, 2018.

[282] M. El-Hajj, A. Fadlallah, M. Chamoun, and A. Serhrouchni, "A Survey of Internet of Things (IoT) Authentication Schemes," *Sensors*, vol. 19, no. 5, p. 1141, 2019.

[283] J. Mahmod and U. Guin, "A Robust, Low-Cost and Secure Authentication Scheme for IoT Applications," *Cryptography*, vol. 4, no. 1, p. 8, 2020.

[284] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *Proceedings of ACM Conference on Computer and Communications Security*, pp. 237–249, 2010.

[285] U. Rührmair, J. Sölter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, and S. Devadas, "PUF modeling attacks on simulated and silicon data," *IEEE transactions on information forensics and security*, vol. 8, no. 11, pp. 1876–1891, 2013.

[286] G. T. Becker, "The gap between promise and reality: On the insecurity of XOR arbiter PUFs," in *Cryptographic Hardware and Embedded Systems (CHES)*, pp. 535–555, 2015.

[287] F. Ganji, S. Tajik, F. Fäßler, and J.-P. Seifert, "Strong machine learning attack against pufs with no mathematical model," in *Cryptographic Hardware and Embedded Systems (CHES)*, pp. 391–411, 2016.

[288] J. Delvaux, "Machine-Learning Attacks on PolyPUFs, OB-PUFs, RPUFs, LHS-PUFs, and PUF–FSMs," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 8, pp. 2043–2058, 2019.

[289] J. Hovanes, Y. Zhong, and U. Guin, "Beware of Discarding Used SRAMs: Information is Stored Permanently," in *IEEE Physical Assurance and Inspection of Electronics (PAINE)*, pp. 1–7, 2022 (Best Paper Award).

[290] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Silicon Physical Random Functions," in *ACM Conference on Computer and Communications Security*, pp. 148–160, 2002.

[291] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "Fpga intrinsic pufs and their use for ip protection," in *International workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2007.

[292] D. E. Holcomb, W. P. Burleson, and K. Fu, "Power-up sram state as an identifying fingerprint and source of true random numbers," *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1198–1210, 2008.

[293] X. Xin, J.-P. Kaps, and K. Gaj, "A configurable ring-oscillator-based PUF for Xilinx FPGAs," in *IEEE Euromicro Conference on Digital System Design*, pp. 651–657, 2011.

[294] U. Feige, A. Fiat, and A. Shamir, "Zero-knowledge proofs of identity," *Journal of cryptology*, vol. 1, no. 2, pp. 77–94, 1988.

[295] A. Fiat and A. Shamir, "How to Prove Yourself: Practical Solutions to Identification and Signature Problems.," in *Crypto*, vol. 86, pp. 186–194, Springer, 1986.

[296] M. Blum, P. Feldman, and S. Micali, "Non-Interactive Zero-Knowledge and Its Applications," in *Symposium on Theory of Computing*, pp. 103–112, 1988.

[297] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *Symposium on Security and Privacy*, pp. 459–474, 2014.

[298] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in *IEEE Symposium on Security and Privacy*, pp. 397–411, 2013.

[299] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly Practical Verifiable Computation," *Communications of the ACM*, vol. 59, no. 2, pp. 103–112, 2016.

[300] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn, "Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2111–2128, 2019.

[301] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward, "Marlin: preprocessing zkSNARKs with universal and updatable SRS," in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 738–768, Springer, 2020.

[302] D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing," in *International Cryptology Conference*, pp. 213–229, Springer, 2001.

[303] W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, 1976.

[304] K. Xiao, M. T. Rahman, D. Forte, Y. Huang, M. Su, and M. Tehranipoor, "Bit selection algorithm suitable for high-volume production of SRAM-PUF," in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pp. 101–106, 2014.

[305] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pp. 326–349, 2012.

[306] M. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen, "MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity," in *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 191–219, Springer, 2016.

[307] X. Salleras, and V. Daza, "ZPiE: Zero-Knowledge Proofs in Embedded Systems," Mathematics, pp.1-17, 2021.

[308] GitHub, "ZPiE: Zero-knowledge Proofs in Embedded systems, `https://github.com/xevisalle/zpie`," 2021.

Appendix

List of Publications (and Source Code) of this Dissertation

Journal Publications

- **Y. Zhong**, and U. Guin, "A Comprehensive Test Pattern Generation Approach Exploiting SAT Attack for Logic Locking," in IEEE Transactions on Computers, pp. 1-13, 2023.
- **Y. Zhong**, and U. Guin, "Complexity Analysis of the SAT Attack on Logic Locking," in IEEE Transactions on Computer-Aided Design of Integrated Circuits, pp. 1-14, 2023.
- **Y. Zhong**, A. Jain, M.T. Rahman, N. Adadi, J. Xie, and U. Guin, "AFIA: ATPG-Guided Fault Injection Attack on Secure Logic Locking," Journal of Electronic Testing: Theory and Applications, pp. 1-20, 2022.

Conference Publications

- **Y. Zhong**, A. Ebrahim, U. Guin, and V. Menon, "A Modular Blockchain Framework for Enabling Supply Chain Provenance," in IEEE Physical Assurance and Inspection of Electronics (PAINE), pp. 1-7, 2023.
- **Y. Zhong**, J. Hovanes, and U. Guin, "On-Demand Device Authentication using Zero-Knowledge Proofs for Smart Systems," in Great Lakes Symposium on VLSI (GLSVLSI), pp. 569-574, 2023.
- J. Hovanes, **Y. Zhong**, and U. Guin, "A Novel IoT Device Authentication Scheme Using Zero-Knowledge Proofs," in GOMACTech, pp. 1-4, 2023.
- J. Hovanes, **Y. Zhong**, and U. Guin, "Beware of Discarding Used SRAMs: Information is Stored Permanently," in IEEE Physical Assurance and Inspection of Electronics (PAINE), pp. 1-7, 2022. **(Best Paper Award)**
- **Y. Zhong**, and U. Guin, "Fault-Injection Based Chosen-Plaintext Attacks on Multicycle AES Implementations," in ACM Great Lakes Symposium on VLSI (GLSVLSI), pp. 1-6, 2022.

  [Source code: https://github.com/yadi14/cpa-multicycle-AES]

Y. Zhong, and U. Guin, "Chosen-Plaintext Attack on Energy-Efficient Hardware Implementation of GIFT–COFB," in IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 1-4, 2022.

[Source code: https://github.com/yadi14/cpa-energy-efficient-gift-cofb]