University of Massachusetts Amherst
ScholarWorks@UMass Amherst

Doctoral Dissertations | Dissertations and Theses

March 2020

Design of Hardware with Quantifiable Security against Reverse Engineering

Shahrzad Keshavarz

Follow this and additional works at: https://scholarworks.umass.edu/dissertations_2

Part of the Digital Circuits Commons, Hardware Systems Commons, and the VLSI and Circuits, Embedded and Hardware Systems Commons

Recommended Citation

Keshavarz, Shahrzad, "Design of Hardware with Quantifiable Security against Reverse Engineering" (2020). *Doctoral Dissertations*. 1837.
https://doi.org/10.7275/3nwv-1f33 https://scholarworks.umass.edu/dissertations_2/1837

This Open Access Dissertation is brought to you for free and open access by the Dissertations and Theses at ScholarWorks@UMass Amherst. It has been accepted for inclusion in Doctoral Dissertations by an authorized administrator of ScholarWorks@UMass Amherst. For more information, please contact scholarworks@library.umass.edu.

DESIGN OF HARDWARE WITH QUANTIFIABLE SECURITY AGAINST REVERSE ENGINEERING

A Dissertation Presented

by

SHAHRZAD KESHAVARZ

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 2020

Electrical and Computer Engineering

© Copyright by Shahrzad Keshavarz 2020

All Rights Reserved

DESIGN OF HARDWARE WITH QUANTIFIABLE SECURITY AGAINST REVERSE ENGINEERING

A Dissertation Presented

by

SHAHRZAD KESHAVARZ

Approved as to style and content by:

____

Daniel Holcomb, Chair

____

Wayne Burleson, Member

____

Russel Tessier, Member

____

Amir Houmansadr, Member

____

Christopher Hollot, Department Head
Electrical and Computer Engineering

DEDICATION

To my beloved family - My husband for his endless support, my parents for all the love and encouragement throughout my life, and my sister who has always been the closest friend to me.

ACKNOWLEDGMENTS

Firstly, I want to thank Prof. Holcomb for his continuous support and mentorship during my years as a Ph.D. student. This dissertation would not have been possible without his advice, knowledge, and encouragement. I would also like to thank my thesis committee: Prof. Burleson, Prof. Tessier, and Prof. Houmansadr, for their insightful comments and feedback regarding this dissertation.

I thank my fellow labmates, Peter Stanwicks, who worked on the preliminary results for parts of the research done towards this dissertation, and Siva Nishok Dhanuskodi for all his help.

And most importantly, I want to express my deepest gratitude to my family. Firstly, my parents who have dedicated their lives to their children's education and well-being and supported me throughout my entire life. My sister that has been there for me even though distance took us apart and my husband, who I have shared the ups and downs of my life with, and who helped me throughout all the challenges I had throughout these years.

ABSTRACT

DESIGN OF HARDWARE WITH QUANTIFIABLE SECURITY AGAINST REVERSE ENGINEERING

FEBRUARY 2020

SHAHRZAD KESHAVARZ

B.Sc., SHAHID BEHESHTI UNIVERSITY

M.Sc., UNIVERSITY OF TEHRAN

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Daniel Holcomb

Semiconductors are a 412 billion dollar industry and integrated circuits take on important roles in human life, from everyday use in smart-devices to critical applications like healthcare and aviation. Saving today's hardware systems from attackers can be a huge concern considering the budget spent on designing these chips and the sensitive information they may contain. In particular, after fabrication, the chip can be subject to a malicious reverse engineer that tries to invasively figure out the function of the chip or other sensitive data. Subsequent to an attack, a system can be subject to cloning, counterfeiting, or IP theft. This dissertation addresses some issues concerning the security of hardware systems in such scenarios.

First, the issue of privacy risks from approximate computing is investigated in Chapter 2. Simulation experiments show that the erroneous outputs produced on each chip instance can reveal the identity of the chip that performed the computation, which jeopardizes user privacy.

The next two chapters deal with camouflaging, which is a technique to prevent reverse engineering from extracting circuit information from the layout. Chapter 3 provides a design automation method to protect camouflaged circuits against an adversary with prior knowledge about the circuit's viable functions. Chapter 4 provides a method to reverse engineer camouflaged circuits. The proposed reverse engineering formulation uses Boolean Satisfiability (SAT) solving in a way that incorporates laser fault injection and laser voltage probing capabilities to figure out the function of an aggressively camouflaged circuit with unknown gate functions and connections.

Chapter 5 addresses the challenge of secure key storage in hardware by proposing a new key storage method that applies threshold-defined behavior of memory cells to store secret information in a way that achieves a high degree of protection against invasive reverse engineering. This approach requires foundry support to encode the secrets as threshold voltage offsets in transistors. In Chapter 6, a secret key storage approach is introduced that does not rely on a trusted foundry. This approach only relies on the foundry to fabricate the hardware infrastructure for key generation but not to encode the secret key. The key is programmed by the IP integrator or the user after fabrication via directed accelerated aging of transistors. Additionally, this chapter presents the design of a working hardware prototype on PCB that demonstrates this scheme.

Finally, chapter 7 concludes the dissertation and summarizes possible future research.

TABLE OF CONTENTS

Page

ACKNOWLEDGMENTS . . . . **v**

ABSTRACT . . . . **vi**

LIST OF TABLES . . . . **xii**

LIST OF FIGURES . . . . **xiv**

CHAPTER

1. INTRODUCTION **1**

1.1 Position of the Dissertation 2
1.2 Overview of Reverse Engineering 3

1.2.1 Attacker models 4

1.3 Process variation 5
1.4 Design obfuscation 7
1.5 Oracle-guided deobfuscation 8
1.6 Secure key storage 9

2. PRIVACY CHALLENGES OF APPROXIMATION **11**

2.1 Introduction 11
2.2 Related Work 13
2.3 Identification from Overscaling 15
2.4 Methodology 18
2.5 Evaluation 20

2.5.1 Measuring the Entropy of Vectors 20
2.5.2 Identification Results 21
2.5.3 Impact of Noise and Number of Vectors 25
2.5.4 Impact of Error Rate 25

2.6 Conclusion 27

3. CIRCUIT OBFUSCATION WITH MULTIPLE VIABLE FUNCTIONS . . . 28

3.1 Related Work . . . 32
3.2 Setting . . . 34

3.2.1 Illustrative Example: S-Box circuits . . . 34
3.2.2 Cell Library . . . 35

3.3 Problem Formulation . . . 35

3.3.1 Phase I: Multi-Function Synthesis . . . 36
3.3.2 Phase II: Maximizing Logic Sharing . . . 37
3.3.3 Phase III: Technology Mapping to Deploy Cells . . . 41

3.3.3.1 Example: . . . 45

3.4 Evaluation . . . 46
3.5 Summary of Results . . . 48

4. SAT-BASED REVERSE ENGINEERING OF OBFUSCATED CIRCUITS . . . 49

4.1 Introduction . . . 49

4.1.1 Related Work . . . 50
4.1.2 SAT Attacks . . . 50
4.1.3 Attacker Model . . . 52

4.2 SAT Formulation for Unknown Gates and Connections . . . 54

4.2.1 Configuration Variables for Unknown Connections . . . 55
4.2.2 Configuration Variables for Unknown Functions . . . 56

4.3 Learning from Voltage Probing . . . 56
4.4 Learning from Fault Injection . . . 60
4.5 Extended SAT Formulation . . . 62

4.5.1 Restriction to Acyclic Topologies . . . 62

4.5.1.1 Encoding Constraints . . . 64
4.5.1.2 Ordering of Levels . . . 64
4.5.1.3 Uniqueness of Levelization . . . 65

4.5.2 Adding Voltage Probing to SAT problem . . . 66
4.5.3 Adding fault injection results to SAT problem . . . 66

4.6 Results 67
4.6.1 Distribution of SAT variables 69
4.6.2 Effectiveness of fault injection and probing 69
4.6.3 Adding Additional Constraints 70
4.7 Conclusions 71

5. SECURE KEY STORAGE IN HARDWARE RESILIENT TO REVERSE ENGINEERING AFTER FABRICATION 73

5.1 Introduction 73
5.1.1 Related Work 74
5.1.1.1 Threshold Voltages to Prevent Reverse Engineering 74
5.1.1.2 Physical Unclonable Functions for secure keys 75
5.1.2 Proposed Approach 75
5.2 Sketch of Approach 77
5.3 Threshold-Based Key Storage Elements 77
5.3.1 Fabrication Questions 80
5.4 Reliability of Threshold-based Keys 80
5.4.1 Distribution of Error Probabilities across Cells 82
5.4.2 Distribution of Key Failures Across Chips 83
5.5 Resistance Against Invasive Readout 87
5.5.1 Attacker Model 88
5.5.2 Attacker's Success Rate for Key Readout 90
5.5.3 Cost of Readout by Attacking Multiple Chips 91
5.6 Design Tradeoffs 93
5.6.1 Loosening Reliability Constraints 93
5.6.2 Majority Voting 94
5.7 Conclusions 95

6. SECURE KEY STORAGE IN HARDWARE WITH PROGRAMMING BY DIRECTED AGING 96

6.1 Motivation and Background 96
6.1.1 Transistor Aging 97
6.2 Proposed Method 98
6.2.1 Error Correction 99
6.2.2 Inducing Bias in SRAM cells 102
6.3 Experiments on the SRAM Bias 102
6.3.1 Non-uniform SRAM Bias 102
6.3.2 SRAM Recovery 105
6.3.2.1 Recovery as a Function of SRAM Usage 107
6.4 Characterizing the Key Generation Scheme 108
6.4.1 Repetition Decoder 110
6.4.2 Key Reliability of Empirical Data 110
6.4.3 Key Reliability Model 111
6.4.4 Hardware Costs 114
6.4.5 Amount of Induced Bias in Cells 116
6.5 Hardware Prototype Implementation 117
6.5.1 Hardware Design 118
6.5.1.1 Read/Write Module 119
6.5.1.2 Repetition Decoder Hardware 121
6.5.2 Hardware Implementation 122
6.6 Resistance against Invasive Readout 125
6.7 Conclusions 127
7. CONCLUSIONS AND FUTURE WORK 129
7.1 Summary and Conclusions 129
7.2 Suggestions of Future Research 130
BIBLIOGRAPHY 133

LIST OF TABLES

Table | Page

2.1 Clock period used for each adder type to achieve desired error rate. 20

3.1 Area comparison for merged S-box circuits 48

4.1 Example showing that probed values can rule out certain connections. Among the three possible node pairings that could be the inputs of the gate producing node X, one of the three is non-deterministic and can be ruled out. 58

4.2 Each fault injection attempt comprises an applied input vector, a target node being faulted, and a specific induced value at the target node. When a fault is applied, a corresponding output observation is made that is either $o_{correct}$ (if it matches the non-faulted circuit output) or o_{faulty} (if it differs from the non-faulted output). The table summarizes the information that is revealed by each outcome. 62

4.3 The levelization constraints enforced in the SAT problem to avoid combinational loops 64

4.4 Results for c17 circuit 71

4.5 Results for the 4-bit PRESENT S-Box 71

5.1 Evaluation of equivalent-reliability designs. Each pairing of threshold offset and BCH code are chosen such that the BCH code is the lowest cost code that will satisfy the reliability criterion for that threshold offset. 87

6.1 Combinations of repetition and BCH error correction that meet the reliability criterion of 99% chips having less than 10^{-6} error probabilities. The highlighted row corresponds to the lowest area cost. The reported area is in the units of μm^2. 115

6.2 Threshold offsets of different technology nodes from SPICE simulation that can explain the bias observed in power-up state of the experimental data. Data in table is derived from the plot at right, which shows the average Hamming weight corresponding to different threshold voltages at each technology nodes. 117

LIST OF FIGURES

Figure **Page**

1.1 Image from Torrance and James [106] shows both an optical image (at top) and SEM image (at bottom) of a portion of an OMAP1510 chip in 130nm technology. It can be seen in the image that SEM is necessary for resolving features at this size of technology nodes. 4

2.1 An 8-bit ripple carry adder with full-adder blocks. 16

2.2 Timing diagram for FA7 of the ripple carry adder depicted in Figure 2.1 17

2.3 Entropy distribution of vectors from Ripple carry, carry lookahead and Han-Carlson adders styles. 22

2.4 Matching distance based on outputs produced for 40,000 random input vectors for each adder type. 23

2.5 ROC curve of three adder styles 24

2.6 Increasing the number of applied vectors increases the AUC. 26

2.7 ROC curve of carry lookahead adder at different error rates 27

3.1 Camouflaging of gate-level schematics 29

3.2 Layouts corresponding to standard 2-input (a) NAND and (b) NOR gates. Camouflaging creates look-alike (c) NAND and (d) NOR cells that are difficult to differentiate optically. Layout images are from the work of Rajendran et al. [92]. 30

3.3 The library cell for a 2-input NAND gate 36

3.4 High level schematic for the circuit merging n different 4-input 4-output functions, such as 4-bit S-boxes 37

3.5 An example showing the importance of input positioning for logic sharing 39

3.6 Example connectivity array used for genetic algorithm 40

3.7 Synthesized circuit area of 8 merged PRESENT S-boxes when pin assignment is random or chosen by genetic algorithm. 41

3.8 Example used for describing technology mapping 46

3.9 The layout for 8 S-box merged camouflaged circuit 47

4.1 Using SAT based approach to solve an obfuscated circuit with unknown key value K; (a) The model that includes a MITER for two circuit copies with different keys $K1$ and $K2$; (b) The Oracle is a working chip with correct key K being internally applied. 52

4.2 An example of the proposed gate model. Depending on the values of the configuration variables, this model allows each gate input to be driven from any node, and allows the gate to implement any possible logic function over its inputs. 57

4.3 Number of feasible input pairs for 50 different gates with respect to the number of primary inputs applied to an 8-bit AES S-box. 60

4.4 Cycles in a circuit 63

4.5 Sample levelization encoding in a circuit 65

4.6 The multiplexer that is used to model fault injection for each node n_x. 67

4.7 The circuit shown on top would be modeled as the one shown on the bottom. The multiplexers in thick lines are used to incorporate fault injection and the AND gate controlled with $Probe_C$ signal is used to add voltage probing into the SAT problem. 68

4.8 Proportion of CNF variables being used toward each component of model in ISCAS'85 c17 circuit. 69

5.1 The overall design approach, both from the designer's and attacker's perspective 76

5.2 Overall CAD flow of the work 78

5.3 A simple 6T SRAM cell. The cell is biased toward the 1-state by increasing the magnitude of transistor P2, and biased toward the 0-state by increasing the magnitude of transistor P1. 80

5.4 The 1-probability against different threshold voltage offsets for a PMOS and NMOS transistor 81

5.5 Cumulative distribution function of error probabilities from the simulation data and their relative fitted curves for different magnitudes of voltage offsets 83

5.6 Key failure rates of different design options 86

5.7 For different values of Δ_{vt}, plot shows the probability (P_{re}) that an attacker reads a value for a cell that differs from the value programmed by the designer, as a function of the attacker's measurement error (σ_{err}). 91

5.8 Effect of multiple chip measurements on reverse engineering success rate for different threshold offsets with σ_{err} = 200mV 92

5.9 Tradeoff between attacker's key read success rate (P_{RSkey}) and key failure rate (P_{Fkey}) that can be achieved by using different error correcting codes. 94

6.1 Proposed method of storing the secret key. The key storage infrastructure consists of SRAM cells and error correction block that are built by an untrusted foundry. The trusted user/IP integrator encodes the secret data through directed aging of the SRAM cell transistors. 99

6.2 Setup to induce aging and perform key evaluations after aging 99

6.3 Key generation scheme 101

6.4 Heatmap of SRAM memory before receiving directed aging, sorted by address. Each dot represents the Hamming weight of a single byte as a gradient of 0 to 8. Colors closer to red show a higher byte Hamming weight, meaning the bits are more likely to power-up to "1". The lower byte Hamming weights, where bits have a high tendency to power-up to "0" are shown with whiter dots. 104

6.5 Histogram showing the two distributions of byte groups before biasing. The average Hamming weight of a byte is 2.50 in the white distribution and 5.47 in the red distribution. 105

6.6 Process used to test recovery after directed aging. Parameter t_{off}, the time between power on trials, is varied in different experiments. 106

6.7 Recovery of three SRAM memory devices over 4000 hours with power-on cycles every 1000 seconds. Y-axis shows the percentage of difference between the pre-aging average Hamming weights and the after-aging average Hamming weight. 107

6.8 Plots show same data against different x axes. Recovery depends primarily on number of uses since the aging was applied, and not on the elapsed time. 109

6.9 Key reliability as a function of cell repetition size for `BCH[255,215,5]` 112

6.10 Cumulative distribution function of error probabilities from the experimental data and their relative fitted curves for the two SRAM cell groups with high and low Hamming weights. 113

6.11 Average failure rate of key generation based on model from experimental data, with different amount of repetition. Each line represents one particular BCH code. Increasing the repetition size and using a stronger BCH code can both reduce the key failure rate. 114

6.12 Sketch of the key storage hardware module 119

6.13 Read and write operations in SPI protocol [3] 120

6.14 Interfacing of the read/write module to the FPGA side and SRAM side 121

6.15 RTL schematic of the repetition decoder block from Fig 6.12 122

6.16 Device view from the implementation of the hardware design on the FPGA in Vivado 123

6.17 Key storage hardware prototype on breadboard 124

6.18 Key storage hardware prototype on printed circuit board 124

6.19 Bit error rate with only repetition and no BCH 126

6.20 Attacker's read success rate versus key reliability for a combination of repetition and BCH codes 127

CHAPTER 1

INTRODUCTION

Semiconductor production is a very large and growing industry, with a global sales totaling $412.2 billion in 2017 as reported by Semiconductor Industry Association (SIA) [1]. Creating an Integrated Circuit (IC) comprises multiple phases, including creating the design specification, making optimizations and finally putting the digital components into silicon. Computer-aided Design (CAD) tools try to assist the designer by automating each step from the design process, such as simulation and verification, synthesis and optimization, and placement and routing, such that the designer intervention becomes minimal. The target hardware can be an Application Specific Integrated Circuit (ASIC) that is made exclusively for certain functionality or a Field Programmable Gate Array (FPGA), which is comprised of configurable blocks that can be programmed to create the desired functionality; with each of them requiring their own set of CAD tools.

Today's hardware systems usually require a very complex fabrication process with multiple layers of lithographic masks and can consist of billions of transistors. For example in Intel's 14nm technology, a set of masks can cost from $10 million to $18 million, and wafer production cost can be between $6,225 to $9,960 [38]. It is desirable that all levels of fabrication are part of a trusted foundry; however, usually all or some parts (in case of split-manufacturing) of fabrication is distributed offshore to untrusted foundries because of budget concerns. In an untrusted foundry, a hardware design can be exposed to IP theft, counterfeiting or malicious hardware insertion [34, 42] at different levels of fabrications.

Even in the presence of a trusted foundry, the design can still be subject to reverse engineering after fabrication. An adversary may use invasive reverse engineering methods such as decapsulating and delayering using corrosive chemicals and mechanical etching, or non-invasive techniques such as laser fault injection from the chip's backside to figure out the chip functionality that can be used for cloning and counterfeiting the proprietary hardware, or figuring out the secret key in case of a hardware cryptography core.

1.1 Position of the Dissertation

This dissertation addresses a number of issues related to hardware security: What are the possible threats to conventional hardware security methods and how to enhance them for better resistance against reverse engineering. The position of this thesis is as follows:

I first give a brief overview of the terminologies and concepts used in this thesis in Chapter 1 as an introduction. In Chapter 2, I show that approximate computing, which is used in error-tolerant applications for the goal of power/performance savings can be revealing of device privacy. The obfuscation problem is then investigated in Chapter 3 and a new method is proposed to do a more meaningful gate camouflaging against a knowledgeable reverse engineer that knows about the set of functions that a circuit can implement. Then in Chapter 4, I demonstrate that even the most robust circuit obfuscation methods can be attacked with our proposed enhanced SAT-based reverse engineering approach. By incorporating additional information from laser fault injection and laser voltage probing, the proposed method facilitates reverse engineering by decreasing the search-space of the SAT problem. A novel key generation mechanism is proposed that provides a desired reliability and quantifiable security against reverse engineering in Chapter 5. In Chapter 6, a secure hardware key storage is proposed that addresses the issue of secure key storage with an untrusted foundry. A working

hardware prototype of the key generation system is designed and implemented, and the key correctness is verified over time.

Several important subjects that are discussed in this thesis are the concepts of reverse engineering, process variation, design obfuscation and deobfuscation, and secret key storage. The following subsections briefly introduce the significance of each one.

1.2 Overview of Reverse Engineering

A Reverse engineer tries to reveal what components the chip has, analyze the operations and functions of the chip, or extract the gate-level components or netlist of the circuit [106]. The IC reverse engineering can comprise of depackaging the IC using corrosive chemicals, delayering multi-layered IC by chemical etching and mechanical polishing to reveal the connectivity, traces, and vias in each of its internal layers, and imaging each layer.

Once the images of the layers are prepared, they are annotated and stitched together to be used by software algorithms that reconstruct the schematic [6]. These algorithms use template-based matching to determine the function of each component and trace the routing to understand the connectivity between the components. In some cases, non-destructive reverse engineering may be possible, including methods that use X-ray inspection or laser voltage probing [64].

With the increasing complexity of integrated circuits and small size of transistors, reverse engineering has become more of a challenging task than it was ever before. Today's systems include complex Systems on Chips (SoCs) with hundreds of millions of transistors and several layers of metal wiring. However, the increasing size and complexity of designs have not stopped the reverse engineers, as some of the same automated techniques that have helped the designers achieve a high level of complexity have also been helping the attackers in reverse engineering the designs. These advances in reverse engineering include advanced imaging techniques such as Scanning Electron

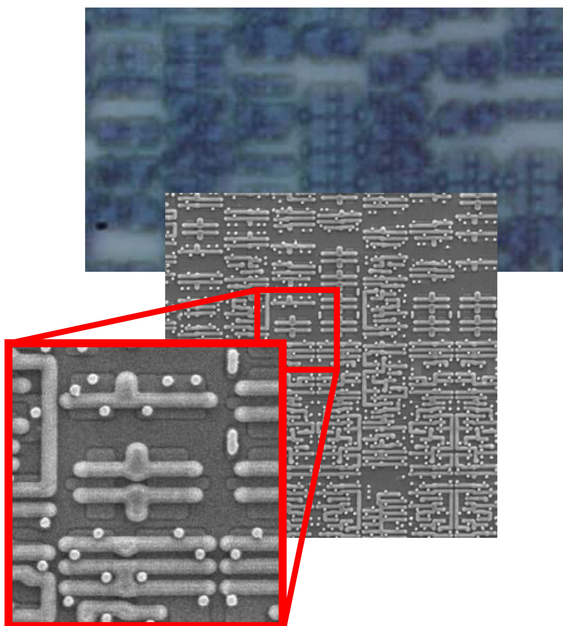


Figure 1.1. Image from Torrance and James [106] shows both an optical image (at top) and SEM image (at bottom) of a portion of an OMAP1510 chip in 130nm technology. It can be seen in the image that SEM is necessary for resolving features at this size of technology nodes.

Microscope (SEM - see Figure 1.1) [106], and automated tools such as Chipworks' ICworks [5] that help with analyzing and reconstructing netlist from images that are taken from a microscope.

1.2.1 Attacker models

Based on the attacker's goals, constraints, and available equipment, he might use different reverse engineering approaches that are translated into different attacker models. For example, Chapter 3 assumes an attacker that tries to decapsulate and delayer the chip to reverse engineer a circuit optically, which is the common case. Such an attacker that has knowledge about everything in the layout, including the gates and their connections, but not the function of camouflaged gates. The attacker in Chapter 5 also tries to reverse engineer the chip from the front-side and decapsulates and delayers the chip, but he has the ability to measure the concentration of dopant

atoms in transistors and estimate their threshold voltage. However in Chaper 4, we assume an adversary that tries to attack a chip from the backside and possesses the equipment to inject faults or probe internal voltage values with a laser. In this case, the attacker cannot know the metal or physical layers, which prevents him from knowing the function of gates and the connections between them. Each attacker model will be discussed in more detail in their relative chapters.

1.3 Process variation

During manufacturing, some amount of inaccuracies in design parameters such as impurity densities and transistor geometries are induced. The effect of these variations in the process has exacerbated over the years and was worsened in below 90nm technologies where feature size became smaller than the wavelength of light, which made the correct printing of the layout extremely difficult [43].

Process variation exhibits itself at different levels, such as changes in electrical parameters like threshold voltage and sheet resistance, which in turn change the delay or power consumption of the fabricated chip. Although process variation is usually an undesirable phenomenon that worsens fabrication yield and design reliability, it can be used as a means to create Physically Unclonable Functions (PUFs). PUFs are device-tied secrets that are repeatable (it can be re-generated via device evaluation under the same situation), yet unpredictable (one cannot tell what the secret would be, without evaluating the PUF). A PUF's value is dependent on the inherent process variation of the device and can be used to verify the identity of IC/FPGA to protect against counterfeiting.

The value of a PUF should solely rely on the inherent process variation of its components. Therefore, most PUF designs rely on differential circuits in which two paths are designed with identical logic and matched routing so that the difference only comes from process variation. This level of matching can be easily done in ASICs

because the designer has the freedom to control the layout of the PUF. However, FPGA designers are limited in this freedom and must work within the constraints of the unmovable look-up tables and routing tracks in the FPGA fabric.

The performance of PUFs is usually measured in the following terms:

- **Uniformity**: Indicates the balance of zeros and ones in the response of the PUF (ideally a PUF is expected to output zero and one in the same probability).
- **Reliability**: Indicates how stably a PUF outputs the same response in the same conditions.
- **Bit-aliasing**: If it happens, different chips may produce nearly identical PUF responses, which is undesirable.
- **Uniqueness**: How different the different PUFs respond in the same conditions [77].

As part of my research that is not explained in this thesis, I have proposed and evaluated a new FPGA-based PUF design. Unlike previous approaches that can only be implemented on SLICEMs, which is a less frequent resource type on the Xilinx FPGAs, our proposed design can be implemented using the standard SLICEL components that are about twice frequent compared to SLICEMs. Moreover, we offer a novel per-device PUF selection approach to increase the PUF reliability compared to other approaches. We then show that compared to other state-of-art approaches, our design has improved reliability while behaving about the same in other performance metrics. Additionally, the new design is more efficient in terms of the type of the required resources, making them more available to be allocated for the design goals. The work appears in IEEE Transactions on Very Large Scale Integration Systems (TVLSI) [108].

In another context, it can be shown that approximate computing can unintentionally reveal the identity of chips, or in other words, their unique process variations. In

recent years, the growing need for energy efficient designs and the emergence of error-tolerant application domains has prompted significant research interest in the area of approximate computing. Approximate computing relaxes some accuracy in the results in exchange for enhancing performance or power consumption. The inaccurate results, however, may uniquely be tied to each device and introduce new security concerns. With the increasing adoption of approximate computing systems in the coming years, designers of approximate computing systems should start considering the associated privacy risks and whether they warrant mitigation. In chapter 2 of the thesis, the privacy leakages from adders of different styles are considered, and the amount of identifying information leaked from each of them is explored. We investigate how some phenomena, such as aging or environmental noises, are causing imperfection in measurements and how different amounts of approximation would affect the accuracy of chip identification differently.

1.4 Design obfuscation

Hardware obfuscation tries to hide the functionality of the chip against reverse engineering. The hardware obfuscation may try to enhance the security of hardware systems at different stages, such as protecting the chip's functionality from an untrusted foundry, or from a reverse engineer after the chip is manufactured. To protect against an untrusted foundry, logic locking can be used that obfuscates the IC by locking the netlist with a secret key. The circuit would provide the correct functionality only when the correct key is applied to the chip by the IP vendor.

One of the state-of-art obfuscation approaches that improves the security of hardware systems after fabrication against reverse engineering is camouflaging, which seeks to visually hide or disguise the features of the chip so that imaging-based reverse engineering will not recover the true function. When using camouflaging, a designer is faced with a decision concerning which subset of gates or interconnects to camouflage.

The designer may consider the different design or security aspects when making this decision. He or she might simply choose random gates to camouflage, and limit the number of total camouflaged gates based on some amount of allowable area overhead; alternatively, the designer may use deploy camouflaging based on resolvability and corruptibility metrics [92].

In case of gate camouflaging, an attacker trying to reverse engineer the circuit must consider an exponential set of functions in order to find the true functionality of the circuit. However, in the case that the attacker has prior knowledge over the circuit functionality, he can rule out many of the possibilities and converge to an answer more easily and quickly. In Chapter 3 of this thesis, we try to consider such a scenario against a more knowledgeable attacker and propose an improved gate obfuscation technique. The proposed method can make each possible function of the camouflaged circuit seem viable from the attacker's perspective, and an automated synthesis flow is provided to ensure this. The resultant circuit is camouflaged in such a way that none of the plausible functions can be easily ruled out by an attacker with additional knowledge about the circuit functionality.

1.5 Oracle-guided deobfuscation

Although the hardware security approaches mentioned in this thesis improve the resistance of circuits against reverse engineering attacks, there are still methods that enable an adversary to reverse engineer an obfuscated circuit. One such method is the SAT-attack, which converts the reverse engineering problem to a Boolean satisfiability (SAT) formulation and uses the state-of-art SAT-solvers to find the circuit's function as a solution.

In case the circuit is invasively camouflaged such that all gate functions and connections are unknown, or reverse engineering is done from the chip backside with no information about the gate or connection, new sources of data are required

to help with the reverse engineering. With advances in technology, new methods and tools are also becoming more available to reverse engineers, such as powerful optical microscopes, laser voltage probing, Scanning Electron Microscopy (SEM). Incorporating the additional data from these techniques into the SAT formulation requires new models and methods.

In Chapter 4 of the thesis, a novel reverse engineering approach is proposed that incorporates laser fault injection and laser voltage probing to simplify the SAT formulation. Similar to the scan chain that can improve the strength of SAT attacks by increasing the circuit's observability, internal voltage probing can enhance the SAT attacks by adding extra points of observations in the circuit. There has not been any previous work that would address how to incorporate this additional information into a SAT formulation. The approach that we propose is based on a commonly used reverse engineering method, which turns the unknowns of a circuit into a SAT problem, and uses a working instance of the circuit as a guideline (Oracle). We model an excessively camouflaged circuit, where all gate functions and their connections are unknown. We then propose a new method to model this circuit, while being able to incorporate all the additional information from laser voltage probing and laser fault injection into the SAT problem. We then use a state-of-art SAT solver to find a solution that not only reveals the circuit functionality but recovers a netlist that is equivalent to the original camouflaged circuit on a gate-by-gate basis.

1.6 Secure key storage

One important idea in Cryptography is Kerckhoffs' principle: A cryptosystem should remain secure even if everything about the system, except the key, is public knowledge [2]. This is in contrast to "Security through obscurity" that attempts to use the secrecy of design to provide security, which is provided in most hardware security techniques such as gate camouflaging techniques. In this chapter of the thesis, we try

to adhere to Kerckhoffs' principle by proposing a secret key storage mechanism and allowing an attacker to know everything about the design except for the characteristics of certain transistors that determine the key. An example of such an approach is a merged circuit that consists of different functions, of whom only one is the true function of the circuit. To select the correct circuit's function among all of the merged functions, multiplexers can be used on the outputs (The process is similar to the one that will be explained in more detail in Chapter 3). The select value of the output multiplexers can be encoded as a secret key, which determines the true circuit function.

We accomplish secret key storage by encoding the secret data into the threshold voltages of the common 6-T SRAM cells and use statistical analysis to evaluate the reliability and security of the proposed design. One of the advantages of our method is that it provides quantifiable security against invasive readout, and enables the designers to achieve different trade-offs between security, cost, and reliability.

As the future work, I propose using direct accelerated aging as a mechanism to induce the desired value within SRAM cells. The advantage of this method is that it enables an IP designer or end-user to burn the secret data, without a need for a trusted fabrication or a fabrication process that supports multiple threshold voltages. Accelerated aging is usually induced by increasing the chip's temperature and supply voltage. Future work includes using the burnt-in secrets from actual SRAM chips that we age in our lab and use them to evaluate the key reliability and security.

CHAPTER 2

PRIVACY CHALLENGES OF APPROXIMATION

2.1 Introduction

In this chapter, we consider how approximate computing can compromise the privacy of a device or a device-bearer. The basic concept of approximate computing is simple: For many applications such as DSP, data mining and multimedia (audio, video, graphics), a perfect result is usually not necessary. In other words, these classes of applications can tolerate some amount of error. The relaxation of accuracy introduces an amount of design space freedom that can be exploited to reduce power consumption or increase performance. Many approximate computing proposals trade away more than just accuracy, but also uniformity of results across devices. When each device is allowed to produce a slightly different, and possibly identifying, result, privacy must now be considered. With predictions of increasing adoption of approximate computing systems in the coming years, designers of approximate computing systems should start considering the associated privacy risks and whether they warrant mitigation.

Although the privacy risk is more general than one particular scenario, we consider here an illustrative scenario of a microprocessor with approximate computing capability that allows results to be influenced by the unique process variations of the chip. In this setting, an adversary that can apply chosen operands to the processor and observe computed results can use this information to identify a device or correlate results to a single device. This leakage of identity to an unprivileged program is perhaps at odds with privacy trends that have resulted in, for example, Intel canceling plans for software accessible processor serial numbers and Apple removing developer access to

unique device identifiers. Approximate circuits exploit the potential error resilience of some classes of applications. This error resilience can have different reasons: *a*) the data is coming from the real world and therefore, is noisy by nature, *b*) the algorithm used is self-healing and can attenuate an amount of error, or *c*) the user of these applications is able to tolerate an amount of error in the result [111]. One method of approximate computing is to use deterministic functional approximation, in which a particular Boolean function is replaced by a simpler one that produces similar results at lower complexity [46, 47]. Because functional approximations compute identical results across all chips, they pose no risk to privacy.

The computational circuits that are of interest in this work are circuits that use non-deterministic approximations, or what are sometimes denoted timing-based approximations [112]. In these approaches, a design is voltage overscaled or frequency overscaled to an operating point where timing constraints may be violated by some circuit paths. At overscaled operating points, the output of a circuit depends not only on inputs, but also on process variation. Design techniques can be used to optimize the quality of results while meeting a power constraint [84], or even to dynamically control the error rate based on the needs of an application [57].

Contributions: The specific contributions we make are as follows:

- We show, for the first time, that results from overscaled approximate computations can reveal the identity of the chip that performed the computation.
- We compare and contrast the identifying ability of the outputs of three popular styles of 32-bit adders.
- We show that random noise does not prevent identification if sufficiently many output vectors are collected, but that a consistent bias such as aging can diminish the effectiveness of identification.

The remainder of this chapter is organized as follows: Section. 2.2 provides related work on approximate computing to give context to our contribution. Section 2.3

explains how approximate computational results can reveal device identity. Section 2.4 addresses methodology. Section 2.5 presents simulation results showing how privacy leakage varies with design, clock frequency, and noise and section 2.6 concludes this chapter.

2.2 Related Work

Many of the efforts toward approximate computing have focused on adders as ubiquitous basic components of digital systems (e.g. [41, 46, 47, 49, 60], among others). More specifically, there has been a lot of research that targets ripple carry adders (RCAs) as an approximate adder of choice because RCAs have a few long paths in the carry chain that are rarely sensitized [49], and this enables a gradual degradation of the quality of results when overscaled. For example, the authors in [60] have targeted RCAs to reduce the error rate within a fixed energy budget and the authors in [41] proposed a biased voltage scaling for probabilistic RCAs that scales the operating voltage according to the significance of bits. Because of the focus on adders in previous approximate computing research, we focus our study on adders as well.

Aside from computational blocks in general and adders specifically, there has also been significant interest in approximate memories. Previous works have proposed DRAM-based approximate memories [72] with unsafe refresh intervals to save energy, fast but inaccurate writes to multi-level non-volatile storage cells [93], and voltage overscaled SRAM [36]. Recently, one paper has shown that data stored in approximate DRAM can be used as a fingerprint to reveal device identity [90]. To the best of our knowledge, this one previous paper is the only work to explore privacy issues in approximate computing systems, and no previous works at all have studied privacy leakages on the computational (i.e. non-memory) side of approximate computing.

The use of process variations to identify devices is similar to the idea of a physical unclonable function (PUF) in security. PUFs are circuits designed to extract identifying

fingerprints from process variations via timing variations [40] or power-up states of SRAM [44, 51]. Recent work has also shown that variations in sensors [21] and wireless transmitters [88] are identifying and can threaten privacy. Many of the efforts toward approximate computing have focused on adders as ubiquitous basic components of digital systems (e.g. [41, 46, 47, 49, 60], among others). More specifically, there has been a lot of research that targets ripple carry adders (RCAs) as an approximate adder of choice because RCAs have a few long paths in the carry chain that are rarely sensitized [49], and this enables a gradual degradation of quality of results when overscaled. For example, the authors in [60] have targeted RCAs to reduce the error rate within a fixed energy budget and the authors in [41] proposed a biased voltage scaling for probabilistic RCAs that scales the operating voltage according to the significance of bits. Because of the focus on adders in previous approximate computing research, we focus our study on adders as well.

Aside from computational blocks in general and adders specifically, there has also been significant interest in approximate memories. Previous works have proposed DRAM-based approximate memories [72] with unsafe refresh intervals to save energy, fast but inaccurate writes to multi-level non-volatile storage cells [93], and voltage overscaled SRAM [36]. Recently, one paper has showed that data stored in approximate DRAM can be used as a fingerprint to reveal device identity [90]. To the best of our knowledge, this one previous paper is the only work to explore privacy issues in approximate computing systems, and no previous works at all have studied privacy leakages on the computational (i.e. non-memory) side of approximate computing.

The use of process variations to identify devices is similar to the idea of a physical unclonable function (PUF) in security. PUFs are circuits designed to extract identifying fingerprints from process variations via timing variations [40] or power-up states of SRAM [44, 51]. Recent work has also shown that variations in sensors [21] and wireless transmitters [88] are identifying and can threaten privacy.

2.3 Identification from Overscaling

Overscaling-based approximate computing relaxes clock period constraints and allows that the long combinational paths of a circuit may not fully propagate within the clock period. In this case, the register at the end of the path may capture intermediate (wrong) results on the clock edge. With shrinking feature size, the effect of process variation has become more significant in recent years. Because of process variation, the critical paths of different chips will have different delays. For example [105] reports 12% frequency variation at 1.1V in 45nm technology and [23] reports 30% frequency variation for sub-90nm technologies. The variable path delays will cause different erroneous outputs in approximate computation.

Example: We now give a concrete example to show how gate delays can lead to different results at overscaled operating points. Figure 2.1 shows an example 8-bit ripple carry adder that has two 8-bit input signals $\{a_7 \ldots a_0\}$ and $\{b_7 \ldots b_0\}$, and a 9-bit output signal $\{c_{out} s_7 \ldots s_0\}$. Assuming that $\{a_7 \ldots a_0\} = 8'b11111111$ and $\{b_7 \ldots b_0\}$ changes from $8'b00000000$ to $8'b00000001$, a carry signal has to propagate all the way from the least significant full adder (FA0) to the most significant full adder (FA7) in order to generate the correct result. We now focus on what occurs after the carry has propagated through the seven less significant full adders and signal c_7 rises on the input to FA7. The rising transition of c_7 will indirectly cause both a falling transition on s_7 and a rising transition on c_{out}; the timing of these transitions will depend on gate delays. Letting the delay of gate i in FA7 (see Figure 2.1) be denoted d_i, when the value of c_7 rises, the output s_7 will fall after time d_2. The critical path to c_{out} goes through gates 3 and 5. Therefore, it takes $d_3 + d_5$ from the time c_7 changes for c_{out} to rise.

The value captured on flip-flops after c_{out} and s_7 will depend on the delays of the gate instances. In the presence of process variation, some gates might be faster or slower on one chip than another. If all gates are slow relative to the clock period,

then the rising transition on c_7 may propagate to neither c_{out} nor s_7 before the clock edge, and the output will be $c_{out}s_7 = 01$. If gates 2,3, and 5 are all fast, then the correct value of $c_{out}s_7 = 10$ will be captured on the clock edge; this is depicted in Figure 2.2a. If gate 2 is slow, and gates 3 and 5 are fast, then output s_7 will not have fallen before the capturing rising clock edge, and the captured value will be $c_{out}s_7 = 11$ (Figure 2.2b). If gate 2 is fast and gate 3 or 5 is slow, then output s_7 will have fallen but c_{out} will not have risen, and the captured output value will be $c_{out}s_7 = 00$ (Figure 2.2c). This example shows that variations in gate delays can lead to different erroneous outputs in approximate computing; this is the reason that overscaled approximate computing may lead to device identification. However if the clock period was determined based on the longest and worst case scenario, none of these would have happened.

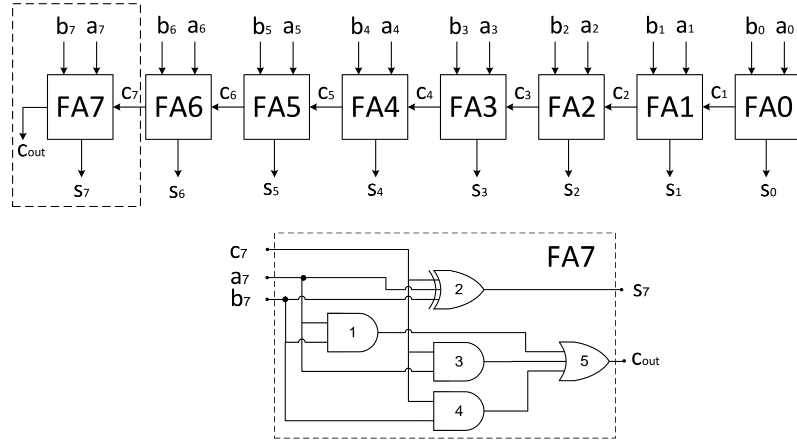
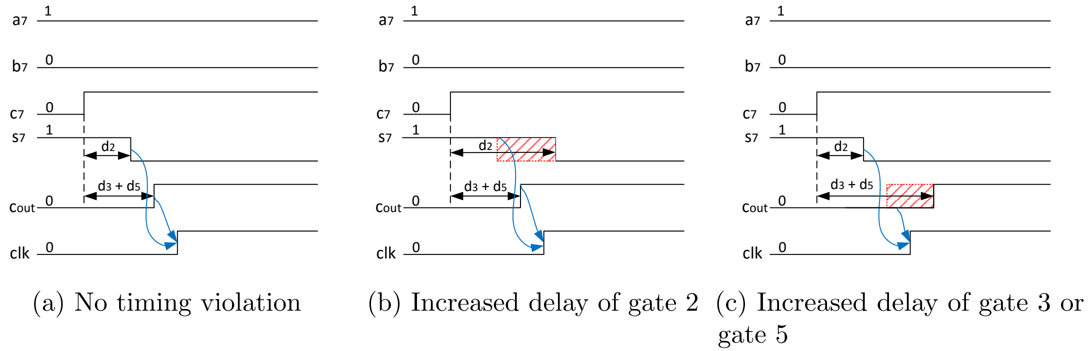


Figure 2.1. An 8-bit ripple carry adder with full-adder blocks.

Entropy of Input Vectors: Note that for the above example, we only considered a portion of a small circuit. For a large circuit, each individual gate may have different delays and there may be many different output results based on different path delays for the same inputs. A good input vector for identification is able to distinguish different chips with different path delays caused by process variation. When applying



(a) No timing violation

(b) Increased delay of gate 2

(c) Increased delay of gate 3 or gate 5

Figure 2.2. Timing diagram for FA7 of the ripple carry adder depicted in Figure 2.1

random input vectors to a circuit, only a small fraction of these inputs may be useful for identification, because the majority of vectors will not sensitize long paths and therefore will produce deterministic error-free outputs. To distinguish the useful vectors from non-useful vectors, we use metric of conditional entropy. When an input vector a_j is applied across a large number of devices at a particular operating point, let the probability of observing output x_i be denoted $Pr(x_i|a_j)$. The entropy associated with the result to input a_j is given by equation 2.1. Although entropy can be estimated from the outputs of adders when viewed as a black box, the entropy associated with different inputs to each adder type implicitly depends on the distribution of path lengths and the path diversity inside of the adder.

$$H(X|a_j) = -\sum_i Pr(x_i|a_j) log_2 Pr(x_i|a_j) \tag{2.1}$$

If an input vector has high entropy on a particular style of adder, it means we get different results for many of the considered chips. In an ideal case, if a vector is able to produce a different result for each chip, it can uniquely identify all chips. However, this is not possible in practice as an input vector usually produces the same results for many chips. Furthermore, noise can diminish the usefulness of high-entropy inputs. Nonetheless, entropy is a useful metric that can provide insights about the identifying ability of each adder, as will be discussed in Section 2.5.1.

2.4 Methodology

We considered three different 32-bit adders for our evaluations: ripple carry adder (RCA), carry lookahead adder (CLA) and Han-Carlson adder (HCA) [48]. Because our experiments require simulating a large number of input vectors on large populations of 32-bit adder circuit instances with different amounts of process variation, using HSPICE simulation alone was found to be impractical in terms of simulation time. Instead, we use HSPICE simulation to extract a number of gate delays for each gate in the library, randomly and then use timed Verilog simulation with the extracted gate delays to simulate the overall 32-bit circuit. We implement each adder style at gate level with Verilog gate primitives and annotate the extracted delays as rise-time and fall-time parameters for the timing simulation. The gate models in HSPICE are 45nm CMOS Predictive Technology Model [11] (PTM) minimum-sized transistors at the voltage of 1.0V. Monte-Carlo simulation is performed 100 times across process variations on V_{th} to provide a distribution of realistic pin-to-pin gate delays for each gate type. When creating an instance of the overall adder circuit, we randomly select gate delay instances from the pre-characterized distributions of each gate type. The timed Verilog models of the adders are simulated using Icarus Verilog (iVerilog).

An overscaled operating point exists when some paths exceed the clock period due to the supply voltage being too low for the applied clock period. Changing the voltage and changing clock period are two different ways of affecting the same amount of overscaling. In our experiments, we control overscaling by changing the clock period while keeping a set of gate delays extracted at one voltage. We choose this approach because it allows us to dial in a target error rate by performing a binary search on the clock period until hitting the desired error rate. If instead trying to achieve a target error rate by searching supply voltage at fixed clock period, we would need a more complicated procedure of re-extracting gate delays many times over at different voltages until hitting a desired error rate. We make comparisons across the different

styles of 32-bit adders by choosing a clock period for each adder style that realizes equivalent rates of erroneous output. For our 32-bit adders, the clock periods that yield 1%, 2% and 5% erroneous outputs are shown in Table 2.1. Note that going from 5% error to 1% error in an RCA requires increasing the clock period by 127 ps, whereas both CLA and HCA require only a 20 ps increase for the same change in error. This occurs because the RCA has many infrequently sensitized long carry chain paths, whereas HCA is a tree adder with many near-critical paths.

To represent persistent and transient non-idealities (e.g. aging and noise) in our timing model, we add random delay components with different scopes. These are used to evaluate the robustness of identification.

- Random Aging: A persistent change, meant to represent issues such as aging, is applied to each gate in the adder in some experiments. It is applied on top of process variations by adding to the delay of each gate a random offset drawn from a normal distribution with 0 mean and standard deviation equal to 10% of the nominal delay. The aging component is chosen independently for each gate in the design, and once applied, the change persists across all vectors simulated on that adder instance.

- Random Noise: A second offset is used to represent noise. Each time noise is added to a gate, it is drawn from a normal distribution with 0 mean and standard deviation equal to 10% of the nominal delay. Noise is uncorrelated across gates, and across vectors, meaning that for each new vector applied to a circuit, the noise offsets are replaced by new values.

Table 2.1. Clock period used for each adder type to achieve desired error rate.

	RCA	CLA	HCA
1% error	653 ps	345 ps	355 ps
2% error	624 ps	340 ps	349 ps
5% error	526 ps	325 ps	335 ps

2.5 Evaluation

We perform a set of experiments to study the extent to which instances of each adder type can be identified by their outputs. We use these experiments to compare the identifiability of the different adder styles, and the impact of noise.

2.5.1 Measuring the Entropy of Vectors

First, for each style of adder, we examine how entropy of input vectors is distributed. We set the clock period for each adder style to achieve a 1% error rate (see Table 2.1), and simulate 200,000 random input vectors on 50 instances of RCAs, CLAs and HCAs. The entropy associated with each input vector is calculated using Equation 2.1. The vectors are then binned according to their entropy and plotted in the histogram of Figure 2.3.

Notably, Figure 2.3 shows that the percentage of vectors with 0 entropy are quite different across adder styles, despite each using a clock period that induces a 1% error rate on a per-chip basis. In the RCA, 97.78% of all input vectors produce the same result on all 50 chips, while in the HCA, the same number is only 90.73%. We investigated this issue and observed that the 0 entropy vectors are always vectors that induce the error-free output on every chip instance[1]. So, in an RCA, a much higher percentage of vectors cause errors on no chips, or in other words sensitize no paths with a delay comparable to or exceeding the clock period. On the other hand, an HCA, which is a tree adder, tends to have a variety of paths with similar nominal delays, and

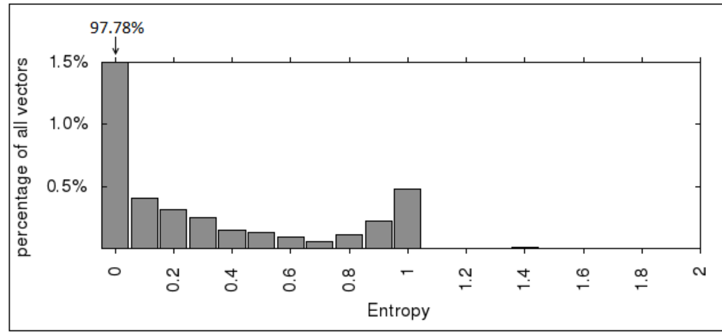
[1]This is perhaps to be expected, as it would be unlikely that an input which produces a wrong output would always induce the same exact wrong output regardless of gate delays

a much lower percentage of vectors are error-free across all chips. Another view of this result is as follows: when considering for each adder type the set of random vectors that caused an error on one or more of the 50 instances, we find that each such vector causes errors in about 71% of RCA chips, versus only 24% for CLA and 10% for HCA adders. If each adder type is operated at the same error rate, the error-causing input vectors will be less unique on the RCA. Note however, that non-unique input vectors does not mean that the output vectors are less unique to each chip; instead, it only means that the inputs that *induce* the erroneous outputs are less unique to each chip.

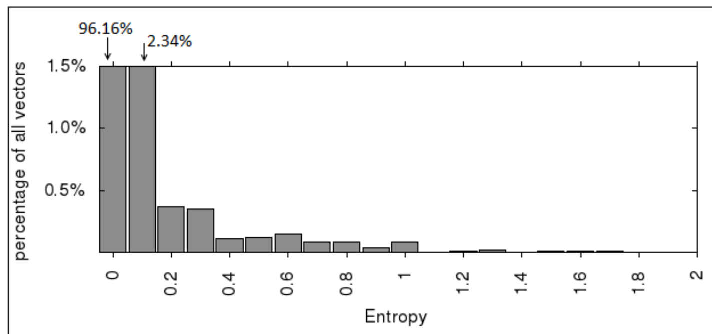
2.5.2 Identification Results

Next we explore identification of chip instances using their outputs. For this experiment, we simulate 40,000 vectors on 50 instances of each adder type operating at their respective clock periods for 1% error (Table 2.1). To measure similarity or lack of similarity between the outputs produced, we use a metric of *Matching Distance*. The matching distance for any two adders of the same type is the number of outputs that are observed differently when the same (40,000) input vectors are applied to both of them (not to be mistaken with *Hamming Distance*, which is the number of bits that differ in two output vectors). The histograms of between-class and within-class matching distances are shown in Figure 2.4. The within-class bars correspond to the matching distance corresponding to two trials of applying the same input vector on the same chip in the presence of noise (see Section 2.4), and the between-class bars correspond to applying the same input vectors on pairings of two different chips. When between-class and within-class overlap less, then one can tell whether two sets of outputs are from the same chip, and can therefore better identify a chip.

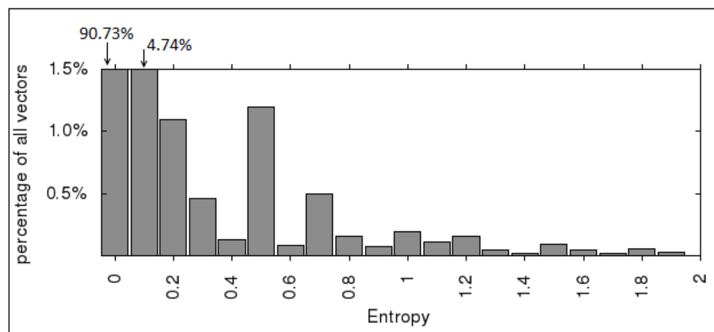
ROC Curve: A chip can always be identified using some matching distance as a decision threshold if the between-class and within-class distances are non-overlapping. A Receiver Operating Characteristic (ROC) curve is used to measure the performance



(a) Entropy of ripple carry adder

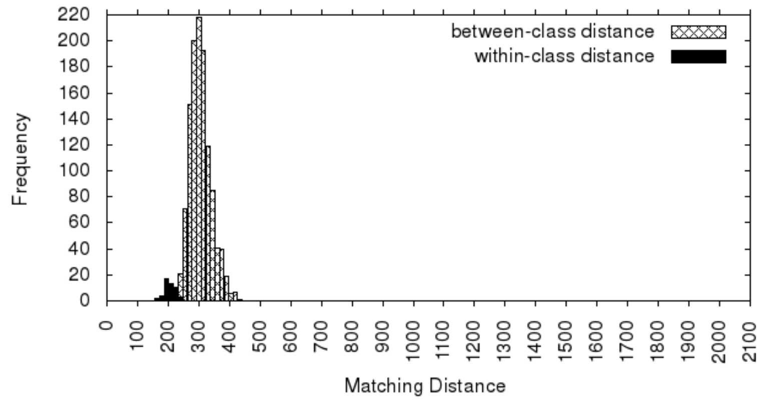


(b) Entropy of carry lookahead adder

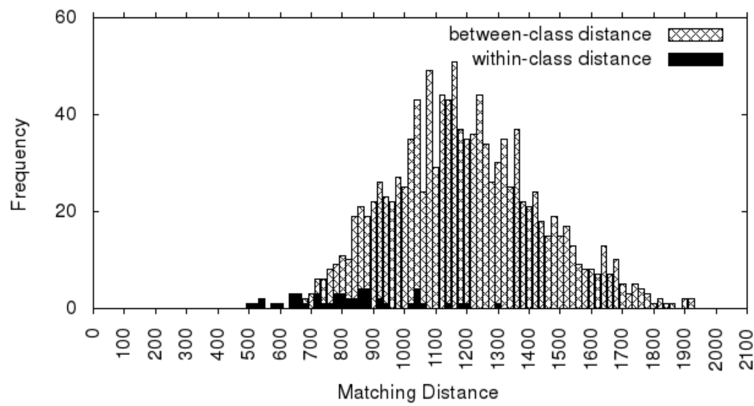


(c) Entropy of Han-Carlson adder

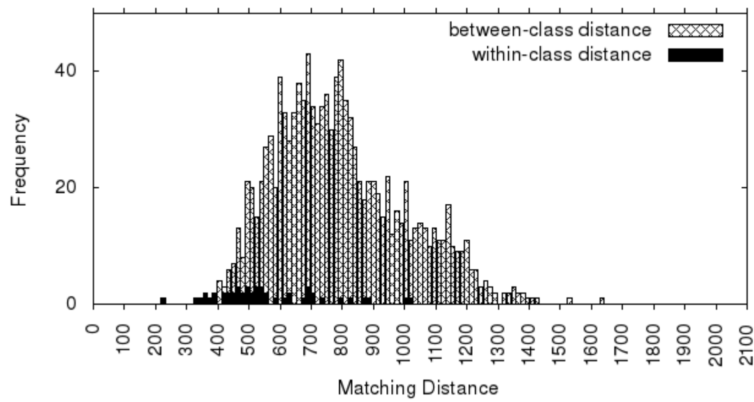
Figure 2.3. Entropy distribution of vectors from Ripple carry, carry lookahead and Han-Carlson adders styles.



(a) Matching distance of ripple carry adder



(b) Matching distance of carry lookahead adder



(c) Matching distance of Han-Carlson adder

Figure 2.4. Matching distance based on outputs produced for 40,000 random input vectors for each adder type.

of chip identification. Each point of an ROC curve corresponds to a single decision threshold and depicts the trade-off between true positives and false positives at that decision threshold. In an ideal case where between-class and within-class distances are separable, the ROC curve will be a step function [55], as this would indicate that there exists some decision threshold that can correctly identify all true positives (within-class pairings) without accepting any false positives (between-class pairings). Figure 2.5 shows the ROC curve for the three adder styles when 40,000 vectors are simulated on 50 instances of each adder using a clock period for 1% error rate. The AUCs for RCA, CLA, and HCA are 0.99, 0.89 and 0.81 respectively. The RCA is easily the most identifiable of the three adder styles in this case.

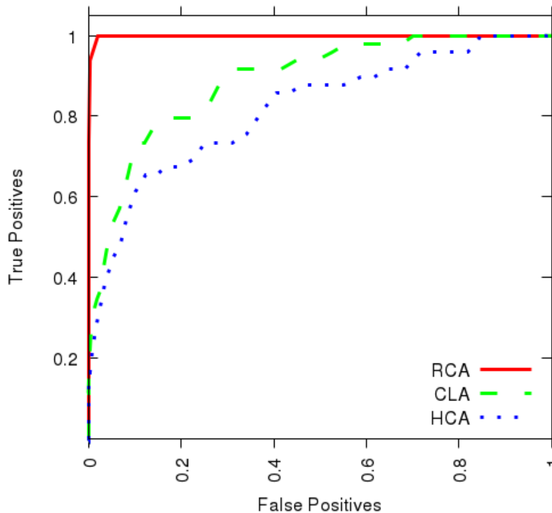


Figure 2.5. ROC curve of three adder styles

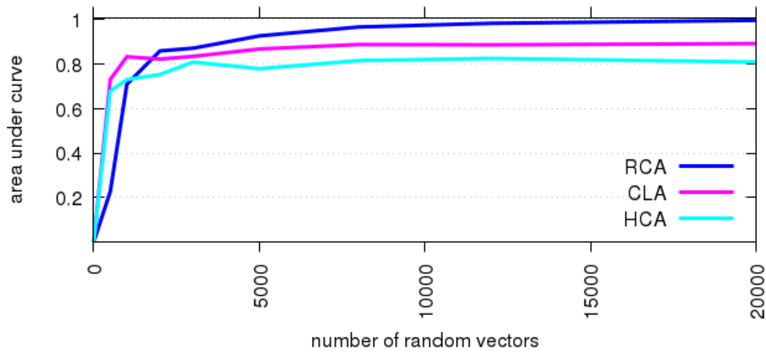
AUC Metric: An ROC curve is a two-dimensional depiction of identification performance. To have a single scalar value for representing the overall identification performance, one can use area-under-curve (AUC), which is defined as the area under an ROC curve. The AUC is a portion of a unit square, so its value can vary between 0 and 1, which are the worst case and ideal case, respectively [37].

2.5.3 Impact of Noise and Number of Vectors

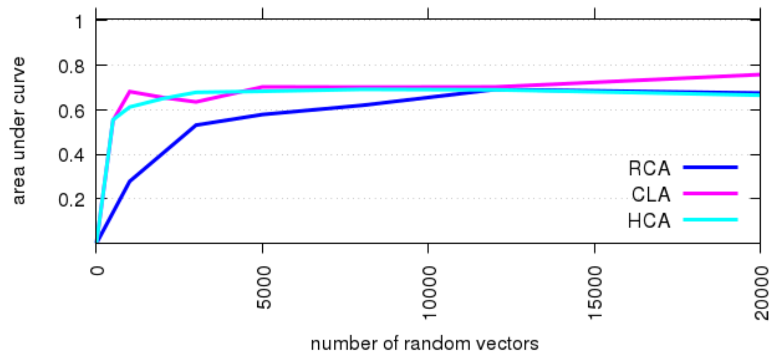
If the number of vectors applied to a circuit is increased, with all else held equal, then the distributions of within-class and between-class distances should overlap less, and the AUC should increase. Therefore, it is expected that the identification success increases with the number of applied vectors. We consider identification performance under both noise and aging when a different number of vectors are applied to the circuits. Figure 2.6a shows performance when only noise is applied. Note that the AUC of the ROC curves in Figure 2.5 would correspond to points on this plot if the x-axis extended to 40,000 vectors. This result shows that noise can largely be mitigated by simply using more vectors. Figure 2.6b shows identification under aging as described in Section 2.4. Figure 2.6c shows identification under reduced aging, where the delay offset representing aging has a standard deviation of 5% of the nominal gate delay, instead of 10% as in the previous figure. These results show that if the delays of gates are changed randomly, it can have deleterious effects on identification, especially in the case of CLA and HCA.

2.5.4 Impact of Error Rate

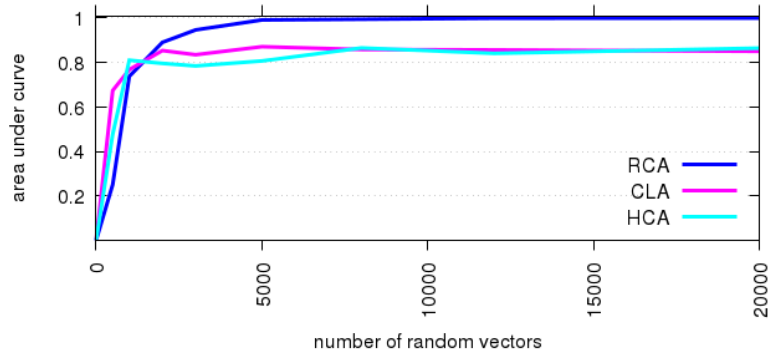
There is usually a trade-off between the number of errors in the outputs and the power/performance improvement in the system. While accepting a higher error rate can be more attractive for efficiency, our results show that a higher error rate can increase the identifiability of a circuit. We chose carry lookahead adder for this evaluation, and in this experiment we set the clock period such that an average error rate of 1%, 2% and 5% are seen on the output results. The results of this experiment are shown in Figure 2.7.



(a) AUC in presence of only noise



(b) AUC in the presence of noise and aging



(c) AUC in presence of noise and reduced aging

Figure 2.6. Increasing the number of applied vectors increases the AUC.

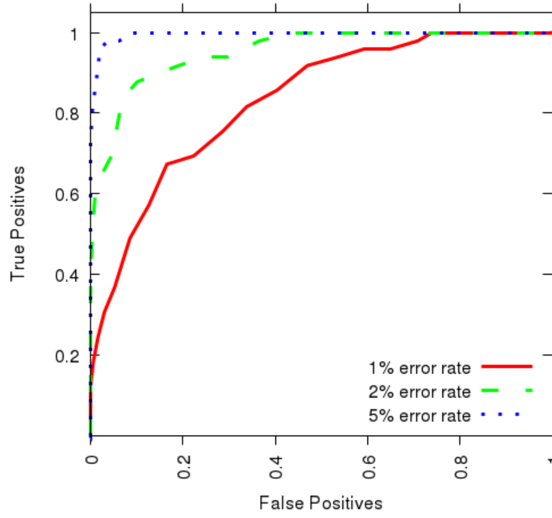


Figure 2.7. ROC curve of carry lookahead adder at different error rates

2.6 Conclusion

The possible privacy implications of voltage-overscaled or frequency-overscaled approximate computations have been demonstrated, for the first time, in this chapter of thesis. We perform an extensive simulation study and show that the ability to provide inputs to a computation unit and observe corresponding outputs can reveal the identity of the approximate computing device that performed the computation. This is a possible privacy risk that designers of future approximate computing systems should consider when evaluating application scenarios. The results of this work have been published in [61].

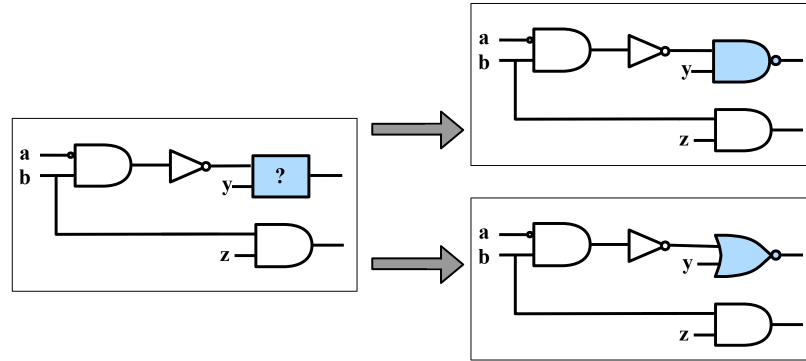
CHAPTER 3

CIRCUIT OBFUSCATION WITH MULTIPLE VIABLE FUNCTIONS

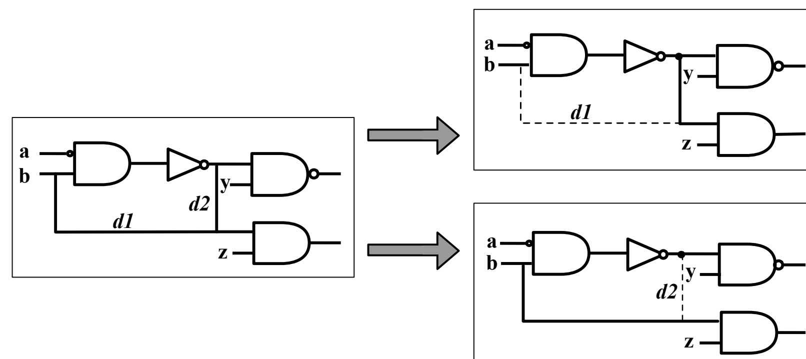
The are many reasons that a chip designer may wish to prevent a reverse engineer from learning the specific function implemented on a target chip. He may, for example, want to avoid IP theft or, in the case of a cryptographic algorithm, prevent an adversary from learning information about the architecture which would allow an adversary to mount side channel attacks.

Partially prompted by the increasing practicality of invasive reverse engineering attacks, there has been several proposals for hiding the true structures of the chip, coined camouflaging. In the case of gate camouflaging, the attacker will only be able to recover the topology of the connections but not the exact gate functions. In the case of interconnect camouflaging, the attacker will only be able to recover the gate functions but not the exact topology of connections between them. An example of the schematic-level uncertainty introduced by camouflaging is given in Figure 3.1.

Figure 3.2 shows an example of camouflaged gate layouts of 2-input $NAND$ and NOR gates. Because the two layouts are different, a reverse engineer can easily determine the gate functions by visual observation. Reverse engineering becomes more difficult when gates use similar layouts that differ only in the contacts between the layers inside the cells. The connections between layers are not visible from top-down imaging, and cross-sectional imaging is impractical for a large chip, making the gate functions not easily distinguishable from each other. Figures 3.2c and 3.2d show a camouflaged layout with dummy contacts. These two nearly-identical layouts



(a) Schematic of a circuit with a camouflaged gate. The circuit structure can be interpreted as the two choices shown on the right.



(b) A similar circuit with camouflaged interconnect. Depending on which wire among $d1$ or $d2$ is conducting, the circuit structure can be seen as implementing the two possibilities shown on the right.

Figure 3.1. Camouflaging of gate-level schematics

correspond to $NAND$ and NOR gates, respectively, but the difference between the two gate functions is determined by which contacts within the cell are made. There can be many ways of creating dummy contacts, but certain techniques such as partially etched vias are not commonly available in standard foundry processes.

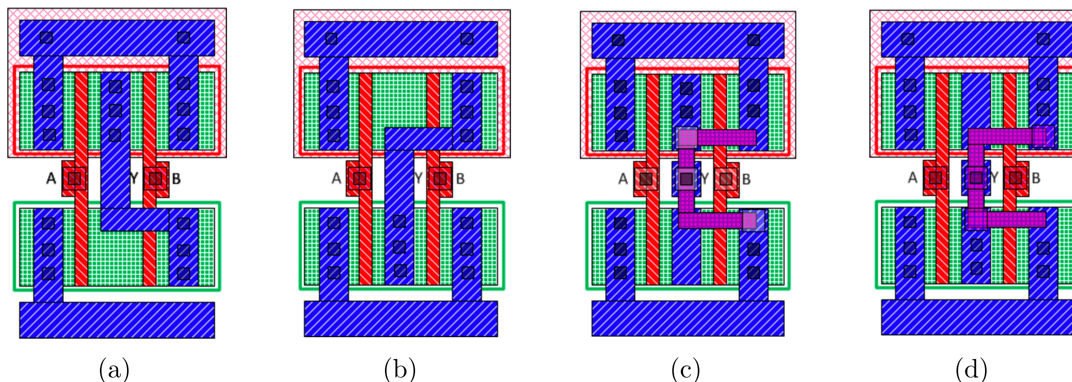


Figure 3.2. Layouts corresponding to standard 2-input (a) NAND and (b) NOR gates. Camouflaging creates look-alike (c) NAND and (d) NOR cells that are difficult to differentiate optically. Layout images are from the work of Rajendran et al. [92].

Given that an adversary won't know the exact function of each look-alike cell, she must consider an exponential set of *plausible* functions that the circuit may implement. We use the terminology "plausible function" of a circuit to denote a function that a circuit or sub-circuit could implement given its use of camouflaged cells. Starting with a synthesized circuit, a designer replaces ordinary cells with camouflaged look-alike cells, and in doing so implicitly creates the exponential set of plausible functions that are guaranteed to contain the true function as well as many other (incorrect) functions. Yet, even a set with exponentially many plausible functions may not fool an attacker who knows that only specific functions are *viable* for the chip's application. For instance, for an obfuscated arithmetic function, it is usually easy for an attacker to extract the correct functionality. Previous works have not addressed how to obfuscate against such an adversary, and have implicitly assumed that the attacker sees all plausible circuit functions as viable functions.

In this work, we consider how to obfuscate a circuit against an adversary that has prior knowledge of a fixed set of viable functions that might be implemented in an obfuscated design. As a case study with practical relevance, we consider a proprietary block cipher. The cipher is based on PRESENT [19], a popular lightweight cipher

that is ISO-standardized [53]. In some application scenarios, it can be advantageous to replace the original PRESENT S-Box with a proprietary function. For instance, military and pay-TV systems are known to employ proprietary encryption algorithms. In addition to PRESENT S-boxes, we also consider the well-known DES S-boxes that are larger in size.

In the setting that we consider, an adversary is not able to query inner circuit values directly, and is seeking to reverse engineer the logic function of a circuit using the following capabilities:

- She has knowledge of the cell library used in the design, including camouflaged look-alike cells.
- She can identify the cells and their connections by imaging the delayered circuit and matching against the components from the library. Her knowledge of the library allows her to infer the plausible functions of each look-alike cell instance, but she does not specifically know which of the functions is implemented by each cell instance.
- She has pre-existing knowledge of a specific set of viable logic functions $F = (f_1, \ldots, f_n)$ for the circuit. In the example we will use, her pre-existing knowledge comes from an assumption that the circuit must implement a cryptographically strong S-box.

The goal of the attacker is then to use the information gained from reverse engineering to guess which viable function is implemented by the circuit.

The goal of the designer in this scenario is to thwart the attack by obfuscating the circuit in a way that prevents the attacker from ruling out any of the viable functions. Specifically, the designer tries to create a single circuit with a fixed set of interconnections for which the plausible functions of each cell will make all viable functions of the overall circuit appear plausible to the attacker.

Our approach for designing circuits that can obfuscate multiple viable functions is motivated by the low probability that any viable function will be plausible when camouflaging is performed randomly. The number of different n-input m-output Boolean functions is doubly exponential (2^{m2^n}), whereas the number of plausible functions is, at most, exponential in the number of camouflaged cells. Therefore, it is improbable that the viable functions will be found in the set of plausible functions unless they are intentionally made to be plausible. This implies that random camouflaging is insufficient for obfuscating viable functions. In this work, we go beyond random camouflaging to present an automation strategy that the designer can use to achieve his goal of making plausible all of the viable functions.

The specific contributions made are as follows.

- A description of the obfuscation problem in which the adversary has partial knowledge about circuit function but lacks the ability to query the direct outputs of the circuit.

- A novel design automation strategy using synthesis, heuristic optimization, and technology mapping to obfuscate circuits in a way that makes a set of chosen functions all appear plausible.

- Evaluation of the approach on cryptographic S-box circuits, showing an area reduction of up to 48% in DES S-boxes and up to 38% in PRESENT S-boxes compared to an approach that does not employ this method.

3.1 Related Work

There have been several proposals for look-alike cells in which it is difficult for the adversary to infer the gate function from its appearance [27]. Camouflaged gate libraries use hard-to-observe structural techniques to differentiate the gate functions [26, 27, 92, 104], or functionality can be controlled without structural differences via transistor

doping [16, 28, 54, 78, 98], or using conducting and non-conducting interconnects in a way that cannot be distinguished by the attacker once the chip is delayered [25]. The attacker model in these works allows that the adversary can identify the obfuscated cells by optical inspection during reverse engineering, but do not allow her to observe the particular aspect of each cell instance that determines its functionality.

When gate camouflaging technologies are deployed in a circuit, the uncertainty about the functions of gate instances leads to a number of plausible functions for the overall circuit that is exponential in the number of camouflaged gates. In the scenario where intermediate values can be read out of registers through a scan chain, SAT-based attacks [79] and defenses [114, 117] are applicable. To protect against reverse engineering by querying the circuit, intermediate values should be inaccessible [118], especially in a security-critical design. This mitigates the threat of SAT attacks.

Although SAT-based attacks can be prevented by making circuit values inaccessible, a circuit function can be reverse engineered without observable values when some information is known about its function. For example, an attacker can check whether the plausible functions contain a particular function of interest (e.g. a viable function) by checking satisfiability of a QBF problem that is similar to equivalence checking, but with unconstrained side inputs that select which of the plausible functions is realized by the circuit [100]. Because the unconstrained side inputs can choose any plausible function, the result of this check indicates either that the viable function is in the set of plausibly implemented functions, or that the viable function is not plausible and can be ruled out. Note that the attacker is able to perform this check *without being able to observe or control any values in the circuit.* This attack can be prevented if the viable functions of the circuit are a subset of the plausible functions of the circuit. If this condition is met, then an attacker checking whether a viable function is plausible will always find that it is, and thus learn nothing about which viable function is actually implemented.

In another context outside of security, there has been work that considers the problem of creating polymorphic circuits that implement two different functions, depending on the operating conditions such as supply voltage or temperature. One such work uses Cartesian Genetic Programming (CGP) to evolve polymorphic circuits [39], and later work proposes to speed up the fitness function evaluation of CGP with SAT-based equivalence checking [95]. These techniques, while promising, do not appear to be scalable as the number of functions grow, and may not converge to a solution even when the number of polymorphic functions is small. Avoiding the daunting task of trying to evolve a circuit that plausibly implements up to 16 different functions, we rely in this work on an algorithmic synthesis-based approach that is guaranteed to produce a solution.

3.2 Setting

Although our technique is general, we demonstrate the work on the crucial problem of obfuscating S-boxes. The viable functions in this setting are the different cryptographically strong S-box functions, as described below. Our objective is then to design a circuit in which all these strong S-box functions are plausibly implemented.

3.2.1 Illustrative Example: S-Box circuits

We evaluate two kinds of S-boxes for our work. Our primary application is 4-bit S-boxes that are used in lightweight block ciphers such as PRESENT [19], which uses a substitution-permutation structure with sixteen S-boxes in each round. Although the S-box functions are generally specified as part of the algorithm, Leander and Poschmann [71] give 16 families of different optimal 4-bit S-box functions that all have equivalent security. Each such S-box is a 4-bit-input 4-bit-output function that requires around 30 gate equivalents to implement. Using proprietary, i.e., non-standardized, S-Boxes can be advantageous in certain applications. For instance, mounting side-

channel attacks against ciphers with unknown S-Boxes can be a considerable challenge. Also, other key-extraction attacks are of limited use if the cipher that is being used is not known.

We also evaluate our approach on non-proprietary 6-bit-input 4-bit-output S-boxes that are used in DES. Each of these S-boxes is around 150 gate equivalents in area. Although the DES S-boxes are standardized and not proprietary, we use 8 different DES S-boxes for the purpose of having a second related test case for our design technique.

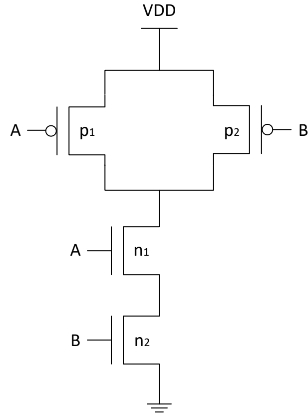
3.2.2 Cell Library

Although our technique is compatible with any library of camouflaged cells, we use cells that are constructed by modifying the doping of nominal library cells. By modifying doping to turn transistors ON and OFF, a cell can be made to implement the positive and negative co-factors of its nominal function with respect to each input.

For example, consider the camouflaged 2-input NAND as shown in Figure 3.3a. The nominal function of the cell is $f = \overline{AB}$. In a variant where p_2 is always ON and n_2 is always OFF, the cell implements $f_{\overline{B}}$, which is constant '1'. On the other hand, if p_2 is always OFF and n_2 is always ON, then the cell is implementing f_B which is $\overline{A}$. The cell can be similarly modified to implement $f_{\overline{A}} = 1$, $f_A = \overline{B}$, and $f_{\overline{AB}} = 0$. Figure 3.3b shows the truth table of all possible functions that can be achieved by changing the transistor doping of a 2-input NAND gate. We use the same approach to create camouflaged versions of the other library cells as well.

3.3 Problem Formulation

We propose a three-phase approach for synthesizing circuits that can plausibly implement a chosen set of functions. We first use ordinary logic synthesis to create a merged logic circuit with the capability to implement all the functions. We then add



(a) An example 2-input NAND gate

Gate inputs		Plausible functions				
A	B	$f_0 = \overline{AB}$	$f_1 = \bar{A}$	$f_2 = \bar{B}$	$f_3 = 1$	$f_4 = 0$
0	0	1	1	1	1	0
0	1	1	1	0	1	0
1	0	1	0	1	1	0
1	1	0	0	0	1	0

(b) The truth table of functions that can be realized by a 2-input NAND gate with non-standard doping

Figure 3.3. The library cell for a 2-input NAND gate

heuristic optimization to decide how to impose the functions onto each other in a way that maximizes logic sharing in synthesis. Lastly, we perform technology mapping to cover the synthesized circuit using the plausible functions of the camouflaged look-alike cells, to reduce cost while preserving security.

3.3.1 Phase I: Multi-Function Synthesis

To have a general circuit that can implement viable functions $(f_0, f_1, \ldots, f_{n-1})$, we write RTL for a design that contains all of the functions with shared input signals, and add multiplexers at the outputs to choose between the outputs of the different functions. Figure 3.4 shows the high-level schematic of this merged circuit for n functions, each with four inputs and four outputs. The select inputs to the multiplexers choose which function's output will be the overall output of the circuit, and therefore, for appropriate assignment to the select inputs, the merged circuit is equivalent to any of the viable functions.

The merged design containing all viable functions is then synthesized to produce a gate-level design where the select signals are inputs that may be used anywhere in the circuit, and not only right at the outputs. We use ABC [4] for synthesis to

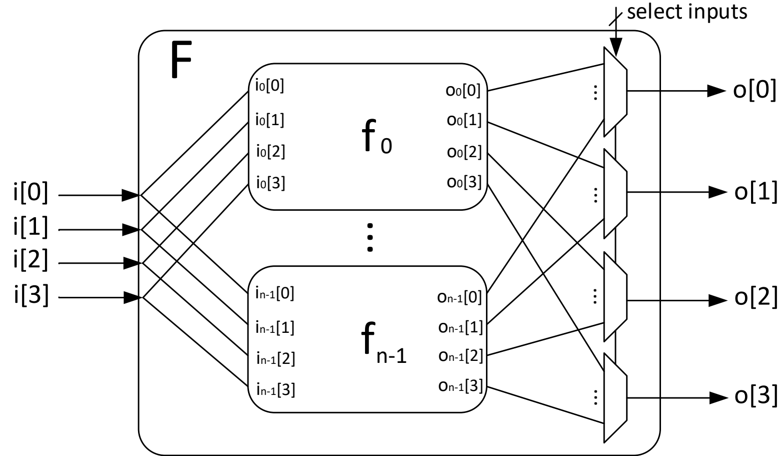


Figure 3.4. High level schematic for the circuit merging n different 4-input 4-output functions, such as 4-bit S-boxes

enhance logic sharing and minimize area with our own script comprising multiple *refactor*, *rewrite* and *balance* commands. Because ABC has limited input syntax, we use Yosys [10] to map RTL into a blif netlist that can be read by ABC. ABC maps the blif to a set of logic gates comprising inverters, buffers, and 2-4 input NAND, NOR, AND, OR gates.

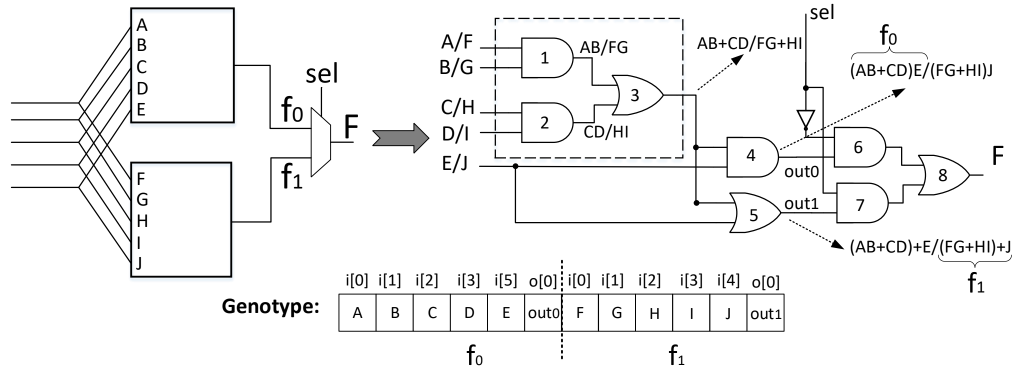
3.3.2 Phase II: Maximizing Logic Sharing

Circuit synthesis of a merged design will inherently try to share logic across the viable functions in order to minimize area. However, the potential for logic sharing depends on the input and output pin assignments of the merged functions. Assuming that an adversary doesn't know which specific signals are carried on particular input or output wires, he must consider a function to be plausible as long as there is some input and output interpretation that causes the obfuscated block to plausibly implement that function. The designer can exploit this degree of freedom to choose the input and output correspondence across the viable functions in a way that will maximize logic sharing.

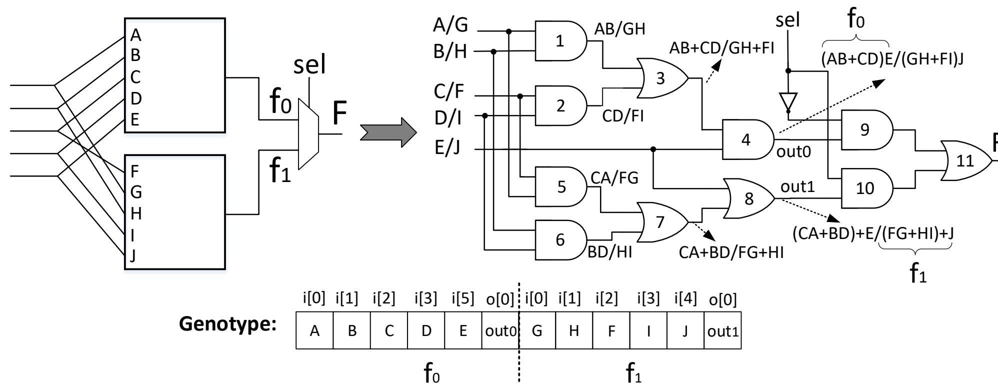
Figure 3.5 shows two different ways of mapping the functions $f_0 = (AB + CD)E$ and $f_1 = (FG+HI)+J$ onto each other. A designer that wants to show both functions as plausible must decide which input of f_0 corresponds to each input of function f_1. The mapping in Figure 3.5a is preferable because it allows the sub-circuit surrounded by a dotted line to be used in both functions f_0 and f_1. However in Figure 3.5b, the input placement does not allow the same extent of sub-circuit sharing between functions f_0 and f_1, and more gates are needed to implement the function. This example shows that assigning input position of each function can increase opportunities for logic sharing between functions and can hence reduce redundant logic to save area. The same observation about effective pin assignment also holds for outputs when the respective functions have multiple outputs.

Considering the first function to have fixed input/output positions and letting the input/output position of all other functions change with respect to this one, then there will be $(inputs!)^{|F|-1}.(outputs!)^{|F|-1}$ possible pin assignments, where $|F|$ denotes the number of viable functions. When the number of viable functions is large, it is infeasible to find the best pin assignment by exhaustive search. Furthermore, a random search may not yield a good solution. To address this issue, we find effective pin assignments using genetic algorithm with the Python Package DEAP [7]. The fitness function used to evaluate the quality of a pin assignment is the synthesized circuit area as reported by ABC. Therefore, we are using repeated logic synthesis in our exploration of pin assignments to try to find a pin assignment that will minimize area by enabling a high degree of logic sharing across the functions.

The genotype of genetic algorithm is a vector that specifies the pin assignments of the viable functions. For inputs, the genotype determines which input pin of each viable function will share the same input pin of the overall merged circuit. For outputs, the genotype specifies which output pins of each viable function will connect to the function-selecting multiplexer of each output in the merged circuit. The fitness



(a) effective input placement for sharing



(b) ineffective input placement that allows less sharing

Figure 3.5. An example showing the importance of input positioning for logic sharing

of genotypes is evaluated using the area reported by synthesis as explained above. Note that synthesis-reported area overestimates the final area because our subsequent technology mapping step reduces area. Nonetheless, area is a useful objective because it encourages configurations that maximize sharing. Genotype instances with high fitness (low area) are propagated using mutation and crossover. In mutation, new instances are created from existing instances by randomly swapping input or output pin assignments within a single function. In crossover, two genotypes are merged to create a new genotype that inherits the pin assignments of some viable functions from one individual and inherits the remainder from a second individual. Figure 3.6 depicts crossover by showing three different genotypes for pin assignment of three

viable functions (f_0, f_1, f_2). In individual 0's genotype, the single input pin $(i[0])$ of the merged circuit is shared by input $i_0[0]$ of f_0, by $i_1[2]$ of f_1, and by $i_2[1]$ of f_2. For the output pins of viable functions, the genotype similarly specifies which viable function outputs are grouped as inputs of each single multiplexer that produces an output of the merged circuit. A new genotype is created from the crossover that inherits the pin assignment of functions f_0 and f_1 from individual 0 and inherits the pin assignments of f_2 from individual 1.

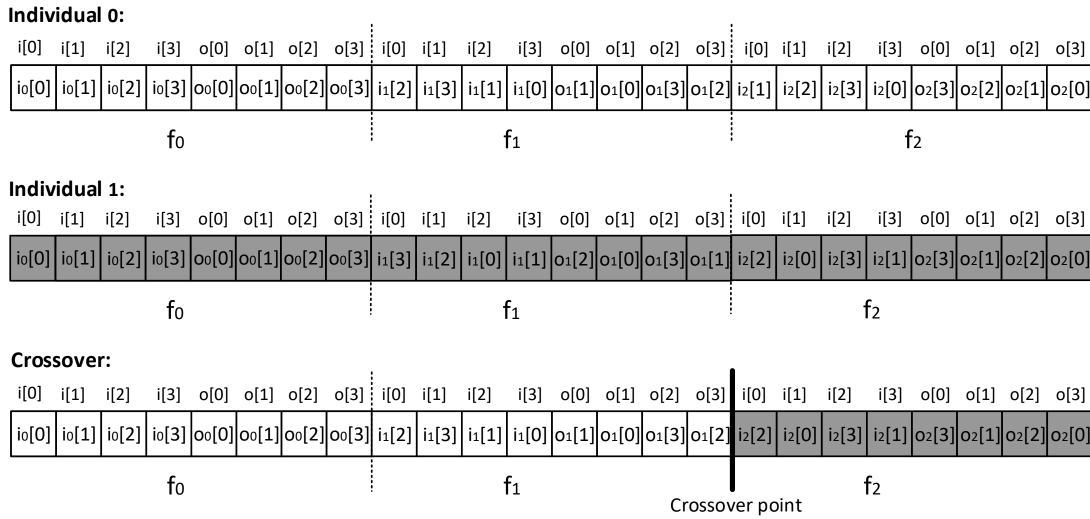
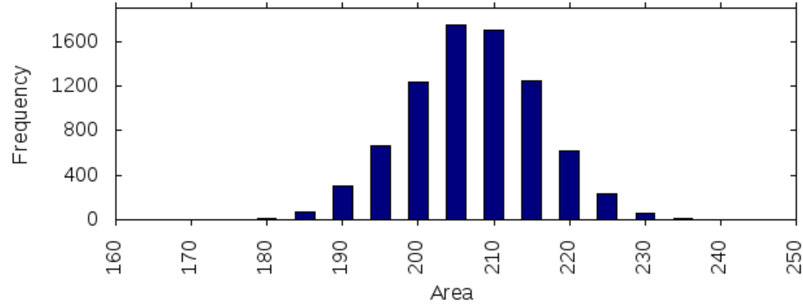
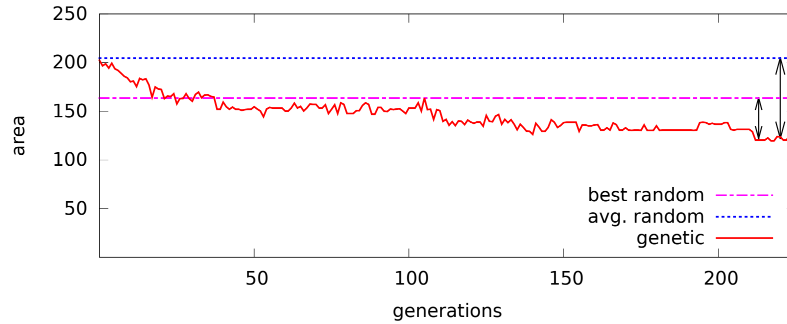


Figure 3.6. Example connectivity array used for genetic algorithm

Figures 3.7a and 3.7b together show that the genetic algorithm is able to find solutions that use less area than what can be achieved by trying random configurations. Figure 3.7b shows how the area improves across iterations of genetic algorithm. The reported areas are in units of GE (gate equivalents) which is the circuit area normalized to the area of a NAND2 gate in the same technology. The x-axis shows the number of generations in the genetic algorithm, with each generation creating and evaluating a number of individuals. For comparison, we also assess a number (9726) of random pin assignments that is equal to the number of individuals evaluated during the genetic algorithm process; the distribution of areas from the random individuals is shown



(a) Area distribution when using random input pin assignments



(b) Area from genetic algorithm surpasses best random

Figure 3.7. Synthesized circuit area of 8 merged PRESENT S-boxes when pin assignment is random or chosen by genetic algorithm.

in Figuew 3.7a. The area of the average and best random solutions are drawn as horizontal lines on Figew 3.7b to show visually that the genetic algorithm method is clearly finding pin assignment solutions that surpass what can be achieved by generating the same number of configurations randomly.

3.3.3 Phase III: Technology Mapping to Deploy Cells

The synthesized merged circuit has a number of logical ”select” inputs that choose between the viable functions. This circuit gets mapped to a circuit with camouflaged gates such that all viable functions in the synthesized circuit become plausible functions in the mapped circuit. One could accomplish this by adding a stealthy mechanism to connect each select signal to supply or ground; the attacker, without knowing

the values of the select signals, would not be able to rule out any viable functions. However, instead of assigning values to the select signals, we use technology mapping to reduce the area cost of the circuit and eliminate the select signals. As will be explained, the key to this mapping is ensuring that, locally for any subcircuit with camouflaged cells, the plausible functions of the subcircuit include all corresponding functions of the synthesized circuit for any assignment to its select inputs. Meeting this condition ensures that all viable functions that were plausible in the synthesized circuit will remain plausible in the mapped circuit.

As is common in technology mapping [66], our approach decomposes the circuit graph into trees and uses dynamic programming tree covering to map the trees into cells. Each tree describes a fanout-free subcircuit with a single output that implements some Boolean function over the leaf nodes of the tree. The significant difference between our approach and ordinary technology mapping by tree covering is that in ordinary technology mapping, a subtree can be mapped to a cell if the cell's single function is *equivalent to* the subtree's single function. In our approach, since we have to consider multiple functions depending on the value of the select signals, a subtree can be mapped to a cell if the cell's plausible functions *contain* all of the desired functions for the subtree.

To allow tree covering to be used on the synthesized circuit, we first create a forest of trees from the circuit by splitting it at all fanouts. If any trees have only select signals as leaf nodes, we duplicate the tree and prepend a copy to every other tree that uses its output as a leaf node. Then the original tree is deleted as an independent tree. This step is performed because trees with select inputs for all nodes will incur cost if they are technology mapped, but add no cost when mapped as parts of other trees because when they can be absorbed into the covering of subsequent gates.

The procedure ABSFUNC (Algprithm 1) determines the functions that need to be covered by the mapped version of any tree from the synthesized circuit. This

procedure abstracts the values of any select inputs that are leaf nodes in the tree. The output of the procedure is the set of Boolean functions must be plausible in the mapped version of the tree. Note that select inputs will appear only as leaf nodes of trees because they are primary inputs of the circuit.

The tree-covering procedure for mapping each tree to camouflaged cells is described in Algorithm 2. The algorithm uses dynamic programming starting from the leaf nodes of the tree and working toward the output. Whenever a node is considered for mapping, minimum-cost mappings will have already been discovered for all nodes in its transitive fanin. To cover a node, different-sized subtrees having that node as output are candidates to be mapped to a new camouflaged cell. Each candidate subtree will have different leaf nodes from the other candidate subtrees. If a subtree is mapped to a cell, the cost of that covering is the cost of that cell added to the cost of the optimal coverings of all of its leaf nodes (line 10 of Algorithm 2). The lowest cost covering is chosen for each node in the tree until the entire tree is covered.

Algorithm 1 Perform abstraction of select inputs on a logic function described by a fanout-free circuit t. The result after abstraction is a set of Boolean logic functions F where the domain of each function in F is the set of leaf nodes of t that are not select inputs.

1: **function** ABSFUNC(t)
2:     ▷ input t is tree topology circuit of logic gates
3:     ▷ let f denote the Boolean function of output node of t
4:     $F \leftarrow \{f\}$ ▷ $f : 2^{|Leaves(t)|} \mapsto \{0,1\}$
5:     **for each** $S_i \in$ select signals of t **do**
6:         **for each** $f' \in F$ **do**
7:             $F \leftarrow F \setminus f'$
8:             $F \leftarrow F \cup \{f'_{\overline{S_i}}, f'_{S_i}\}$ ▷ replace f' with cofactors
9:         **end for**
10:     **end for**
11:     **return** F
12: **end function**

Algorithm 2 Technology mapping to cover a tree with camouflage cells to eliminate the select inputs while preserving as plausible all functions of the output node that could occur under any assignment of the select inputs.

```
 1: function TREE-COVER(t)
 2:   cost(n_i) ← ∞   ∀ nodes n_i ∈ t
 3:   for each node n_i ∈ t, in topological order do
 4:     for each subtree t_s with output n_i and depth < 3 do
 5:       ▷ leaves of t_s are already-covered nodes in t
 6:       F(t_s) ← ABSFUNC(t_s)                         ▷ functions to preserve
 7:       for each camouflage library cell g_j do
 8:         if plausiblefunctions(g_j) ⊇ F(t_s) then
 9:           ▷ cell g_j contains all functions of t_s
10:           c ← cost(g_j) + Σ_{n_k ∈ Leaves(t_s)} cost(n_k)
11:           if c < cost(n_i) then
12:             cost(n_i) ← c                           ▷ new opt. cover for n_i
13:             ▷ Cover n_i by mapping t_s to cell g_j
                  and using optimal covers for leaf
                  nodes of t_s
14:           end if
15:         end if
16:       end for
17:     end for
18:   end for
19:   return mapped circuit for tree
20: end function
```

3.3.3.1 Example

We demonstrate the tree covering algorithm (Algorithm 2) using the simple tree shown in Figure 3.8a. The three nodes B, D, and E must be covered, in that order.

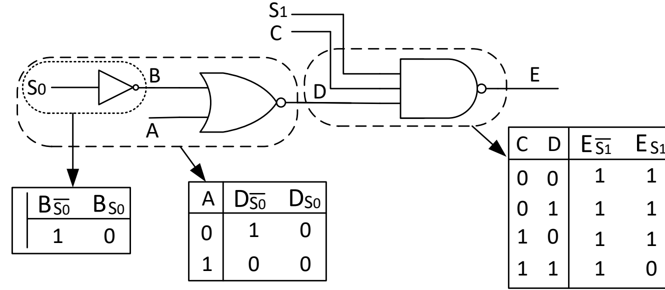
Covering B: The only subtree that outputs node B is the one with leaf node S_0. The abstracted function of this subtree is the set containing the constant 0 (as would occur if S_0 is 1) and constant 1 (if S_0 is 0), so this subtree gets mapped to a cell that has both such functions as plausible. The subtree gets mapped to INV cell because that has the lowest cost among all cells that plausibly implement constant 0 and constant 1, and therefore the cost of covering B is 2/3 (in terms of gate equivalents).[1]

Covering D: The algorithm considers two alternative subtrees that have node D as their outputs. One has leaf nodes B and A, and has no select inputs, so its abstracted function is simply the NOR2 function. Mapping this subtree to NOR2, the cover of node D would be the NOR2 in addition to the INV that covers B, at a total cost of 5/3. A second subtree to consider for node D has leaf nodes S_0 and A, and its abstracted functions are as shown in the center table of Figure 3.8a. These functions are INV and constant 0, and both are plausibly implemented by a single INV gate with A as its input. Because this covering has a cost of 2/3, it is chosen as the optimal covering of node D.

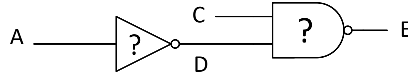
Covering E: One subtree with output node E is the one with leaf nodes S_1, C and D. The abstracted function of this subtree contains the NAND2 function (if S_1 is 1) and the constant-1 function (if S_1 is 0); both are implemented by the camouflaged NAND2 (see Figure 3.3), so this subtree can be mapped to the NAND2, and the cover for E would be the NAND2 in addition to the optimal covering of node D, at a total

[1] the select input S_0 remains in the circuit at this point, but will be removed later when the next gate gets covered. This is always the case when the outputs are constants because any gate that can cover the constants can also eliminate the gate and cover its inputs for a lower total cost.

cost of 5/3. All larger subtrees that have E as their output have abstracted function sets that are not contained within the plausible functions of any cell in the library.



(a) The original circuit as could be produced by synthesis, and the logic functions that need to be preserved.



(b) Mapped circuit with camouflaged gates

Figure 3.8. Example used for describing technology mapping

3.4 Evaluation

To evaluate our proposed method, we use the 16 different 4-bit S-box functions from Leander and Poschmann [71] to create obfuscated designs that plausibly implement 2, 4, 8 or all 16 of the S-box functions in a single circuit. Additionally, we create obfuscated designs that plausibly implement 2,4, or all 8 of the 6-bit-input 4-bit-output DES S-boxes. We use genetic algorithm as discussed in Section 3.3.2 and generate random pin position assignments equal to the number of total individuals that are evaluated in genetic algorithm. We then use the technology mapping algorithm from Section 3.3.3 to map the resulting circuits into our camouflaged library cells. To validate the correctness of our implementation, we verify using ModelSim that the resulting circuits can implement each of the viable functions when appropriate gate functions are supplied. Table 3.1 reports the synthesized area for the best case and

average case of random pin assignment, as well as the area when genetic algorithm is used (GA), and the area when genetic algorithm is followed by technology mapping (GA+TM); all areas are given in units of GE (gate equivalents).

As can be seen, when comparing our final area to the synthesized result for the best randomly discovered pin assignment, our techniques provide an area improvement of up to 38% for the PRESENT S-box and up to 48% for the merged DES S-box circuit. The area savings from our approach generally increases with the size of the circuit. The modest incremental cost of going from 8 to 16 PRESENT S-boxes is due to the limited size of the circuit. Note that our savings are conservative, as the area cost of the randomly generated solutions do not include the additional costs that would be needed to stealthily connect the select inputs to supply or ground.

Figure 3.9 shows the layout for the camouflaged circuit that plausibly implements 8 4-bit S-boxes, generated from Cadence SoC Encounter using the Nangate 45nm Open Cell Library [12]. The technology mapped circuit contains no select inputs and all camouflaged gates have the layouts of standard logic gates, giving the adversary a large space to explore.

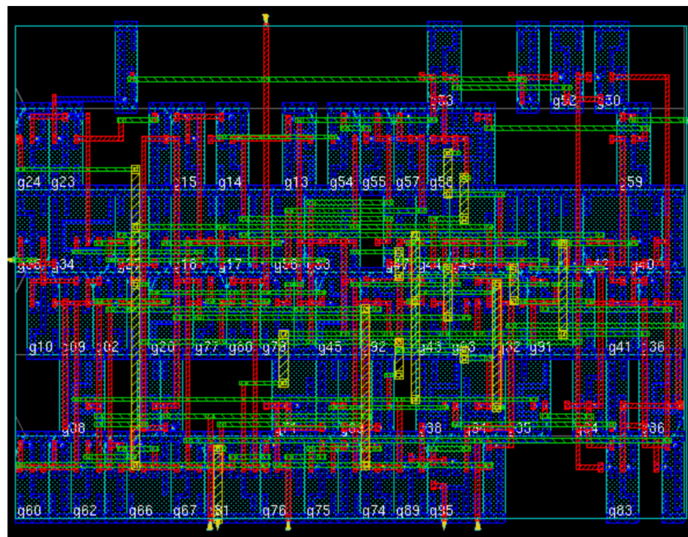


Figure 3.9. The layout for 8 S-box merged camouflaged circuit

Table 3.1. Area comparison for merged S-box circuits

Circuit	#S-boxes	Random avg	Random best	GA	GA+TM	Improvement(%)
PRESENT	2	54	42	41	39	7
	4	108	84	74	65	23
	8	205	164	118	101	38
	16	248	213	183	141	34
DES	2	257	217	200	195	10
	4	496	447	257	242	46
	8	923	805	473	416	48

3.5 Summary of Results

In this chapter, an automation technique for designing circuits that can plausibly implement a number of chosen functions was proposed. Our procedure comprises synthesis and optimization of pin assignments to maximize shared logic between the functions, and a technology mapping step that deploys camouflaged cells while ensuring that all desired functions are plausible in the final circuit. For the problem of S-box design, this technique saves up to 38% area in PRESENT S-boxes and 48% in DES S-boxes. This approach can find wide application in a number of practical scenarios where the adversary has partial information about what functions would be viable in an obfuscated design.

CHAPTER 4

SAT-BASED REVERSE ENGINEERING OF OBFUSCATED CIRCUITS

4.1 Introduction

Gate camouflaging is a technique that has attracted the attention of chip designers in past years. Camouflaging seeks to hide the true structures of the chip so that imaging-based reverse engineering cannot easily recover the details of the implemented design. The purposes of camouflaging include IP protection and preventing targeted attacks. The related work section of this document described some of the different camouflaging mechanisms that exist in academia and industry.

A number of attacks exist against camouflaging including the SAT attack which is based on Boolean satisfiability solving. In this attack, a reverse engineer uses an uncertain model of the design, together with a functional instance of the chip as an oracle, to discover a set of tests that will reveal the exact logic function of the design. The SAT attack extracts the correct function of the design but is unable to make any claim regarding whether it has recovered the same gate-level schematic of the obfuscated design, or another gate-level schematic that is functionally equivalent to the obfuscated design. In this work, we present a stronger SAT attack for small circuits that makes the following contributions:

- We show how an attacker with probing and fault injection capability can use SAT-based reverse engineering to guide his decisions about which faults to apply and which nodes to probe.

- We propose a new SAT-based reverse engineering formulation that can solve for unknown connections while restricting the search to acyclic networks and avoiding combinational loops that can thwart SAT attacks.

- We show that probing can be used to make inference about connections even when gate functions are unknown.

- We show that fault injection and probing provide additional discriminating factors in reverse engineering that can help SAT attacks to recover schematics that are equivalent to the target on a gate-by-gate basis, instead of merely functionally equivalent in traditional SAT attacks.

4.1.1 Related Work

Imaging-based invasive reverse engineering works by decapsulating the chip, imaging and removing each layer in succession, and then using the images to reconstruct the circuit schematic. Among other applications, reverse engineering is used for competitive analysis in the IC industry, and was used to break the weak cryptography in the Mifare Classic RFID tag [86]. Torrance and James give an overview of the state of the art in invasive reverse engineering [107]. A multi-layered defense model is presented in [89] that incorporates different countermeasures in the device to provide aggregated protection against various attacks. At architectural level, memory timing side channel attacks are also possible that can be addressed with memory-level obfuscation methods [58, 59].

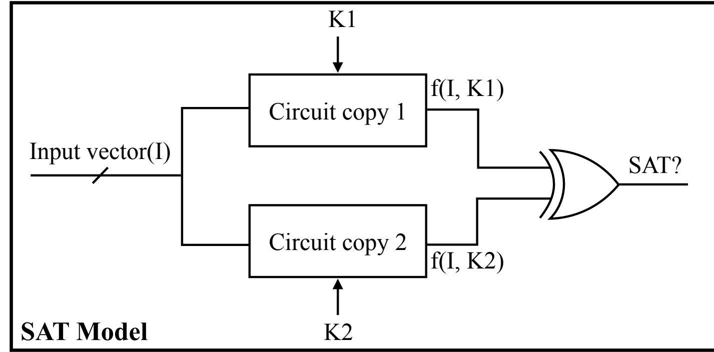
4.1.2 SAT Attacks

An attacker model for reverse engineering circuits with camouflaged gates is given by Rajendran et al. [91]. The logic function of a camouflaged circuit should remain secret when the attacker has knowledge of all non-camouflaged gates and can apply inputs to the circuit and observe outputs. Techniques from oracle-guided

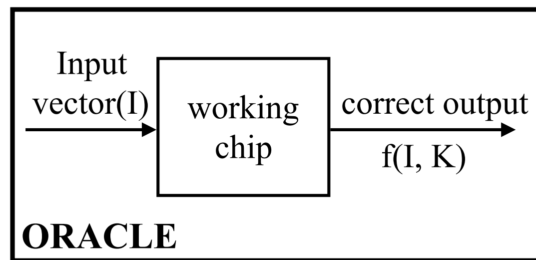
synthesis [56] are used in SAT-based attacks to reverse engineer gate camouflaging or logic encryption [79, 101]. SAT attacks are based on the principle of finding discriminating input vectors, which are input vectors that can eliminate at least one additional circuit function hypothesis once the corresponding output vector is known. Once no further discriminating vectors can be found, it means that no further circuit functions can be ruled out by any tests, and therefore the current set of discriminating inputs is sufficient to uniquely identify the circuit function.

Figure 4.1a shows the basic schematic of a SAT attack. The main idea of the SAT attack is finding distinguishing input patterns (DIPs) that can differentiate between two circuit functions that both seem possible from an attacker's perspective. The distinguishing input patterns are determined by iterative SAT solving and the corresponding outputs for each DIP are found by applying the DIP to the Oracle and observing its output. The SAT solver terminates the process after no more DIPs can be found, meaning that the current set of DIPs is sufficient to recover the exact function of the circuit.

It is important to note that a circuit reverse engineered by oracle-guided synthesis is only guaranteed to be functionally equivalent to the obfuscated circuit, and there is no assurance that it will match the obfuscated circuit on a gate-by-gate basis. Ensuring gate-by-gate equivalence to the obfuscated circuit is generally impossible because the attack only has information about the inputs and outputs. Designs recovered through oracle-guided synthesis are therefore unsuitable for certain classes of side-channel attacks or fault injection attacks that require knowing the states of all combinational circuit nets. In this chapter, we propose a SAT-based de-obfuscation technique that assumes very little knowledge about the obfuscated circuit connections or gates, yet still attempts to reconstruct the exact gate-level schematic of the obfuscated circuit.



(a)



(b)

Figure 4.1. Using SAT based approach to solve an obfuscated circuit with unknown key value K; (a) The model that includes a MITER for two circuit copies with different keys $K1$ and $K2$; (b) The Oracle is a working chip with correct key K being internally applied.

4.1.3 Attacker Model

The attacker model we consider in this work represents an adversary that is trying to reverse engineer a circuit from the backside. This scenario may arise in some chips that have anti-tamper mechanisms that prevent delayering to learn the interconnections of each metal layer. From the backside, the adversary has a very limited knowledge of the circuit as listed below:

- **Connections:** All connections in the circuit are unknown.

  This means that any gate input in the circuit could be connected to the output of any other gate in the circuit.

- **Gate inputs/outputs:** Each gate has a single output, and the output pin of the gate can be identified, yet the adversary cannot see what the gate output connects to. The adversary can know how many inputs each gate has, but cannot know which signals (primary inputs or outputs of other gates) are driving them. If the number of inputs to each gate cannot be determined, the attacker can be conservative and overestimate the number of inputs to each gate.

- **Gate functions:** Our model considers that the attack may know nothing about the gate functions. That is, a gate with n inputs can implement any of 2^{2^n} possible functions, as it can produce a 0 or 1 output for any of the 2^n input combinations. However, the model has the potential to incorporate further knowledge about the gate library.

The assumed attacker capabilities in this work are as described below:

- **Circuit inputs/outputs:** Attacker has a working circuit instance, and can apply the desired inputs to the circuit and observe the outputs. Therefore, primary inputs to the combinational logic block are controllable, and primary outputs are observable. The circuit instance used to correctly map input vectors to output vectors is called the "oracle". In our scenario, we assume that the attacker has full control over the circuit. In case the circuit is part of an encryption hardware, the attacker can control the input to the encryption hardware and knows (or is able to set) an internal secret key. This enables calculating any intermediate values that might occur during computation (the primary outputs of our target circuit). Later, we will exemplary target a PRESENT S-Box as the target for reverse-engineering. Based on the secret key and the plaintext, the attacker can calculate both the input and the output of the S-Box although they are not visible as primary inputs or outputs.

- **Probes:** At some points in the work, the attacker is allowed to probe the value of arbitrary gate outputs. In this setting, the attacker still has no knowledge of connectivity and hence doesn't know what else is being driven by the node that is probed. This represents laser probing. Due to the nature of probing, it is not possible to probe the value of gate inputs.

- **Fault Injection:** At some points in the work, the attacker is allowed to inject faults using a laser. Due to the structure of CMOS circuits, the attacker can either target the pull-up-network or the pull-down-network, forcing the output of the circuit to 0 or 1. However, the attacker does not necessarily know whether the node was a 0 or 1 before the fault was injected so the fault might be ineffective. Both laser probing and fault injection use in principle a very similar setup.

  Note that although this model is very conservative in terms of reverse engineer's knowledge, it has the potential to incorporate further knowledge about the gate library to simplify the reverse engineering process. These less constricting scenarios may include an attacker that tries to reverse engineer an invasively camouflaged circuit from the front-side where he knows a little about gate library functions or their connections, or reverse engineering the circuit from the backside but with a knowledge about plausible function of the circuit, like our assumption from Chapter 3.

4.2 SAT Formulation for Unknown Gates and Connections

As previously mentioned, our attacker model assumes no knowledge of the circuit except for the existence of the gates and the number of their inputs. The attacker will use a SAT attack to determine the functions of these gates and the connections between them. In SAT attacks, the adversary translates uncertainty

about the circuit to the state of certain variables, and then uses observations from the oracle circuit to constrain the values of those variables.

We demonstrate the modeling of connections and gate functions using the example shown in Figure 4.2. In this example, an unknown 2-input gate has output node C and thus is denoted as gate C. The gate exists within a circuit having five nodes (A, B, C, D and E). In this model, the uncertainty about the logic function of gates and uncertainty about wiring connections are both translated into Boolean configuration variables (shown as white dotted-line boxes) that are connected to multiplexers. The values of the configuration variables are unknown, and the SAT solver's task is to find them.

4.2.1 Configuration Variables for Unknown Connections

Since nothing about the connection of gates are known to the attacker, multiplexers are added that are responsible for selecting which node in the circuit is connected to each input of the gate. For example, in Figure 4.2, since the gate has output C, the connection multiplexers choose from the other four nodes of the circuit (A, B, D and E) to determine which is connected to each of the gate's inputs. Therefore in a circuit with N nodes, the connection multiplexers are $(N-1)$-to-1 input multiplexers, as they can select any other node in the circuit except for that gate's own output (node C). In some cases, as will be shown later, certain connections can be ruled out and the number of multiplexer inputs would reduce accordingly.

To keep track of the connectivity between gates, as will be needed later to ensure that the solver only considers acyclic networks, we define transition relation predicates for all pairs of gates. If there is a connection from output of gate A (node A) to one of the inputs of gate C (that has output node C), the predicate $R(A, C)$ will be 1 and otherwise it will be 0. In other words, $R(A, C)$ corresponds

directly to certain assignments of the configuration variables to the connection multiplexer for gate C. In Figure 4.2, predicate $R(A,C)$ is true if and only if the configuration variables for the connection multiplexer connect the output of gate A to an input of gate C; therefore, $R(A,C)$ is true whenever $sel_1 sel_0 = 00$ or $sel_3 sel_2 = 00$.

4.2.2 Configuration Variables for Unknown Functions

The second type of multiplexer employed is for choosing the function of the gate based on the selected inputs from the connection multiplexers. The function multiplexer can be regarded as implementing the truth table of the gate function, choosing which combination of input values should result in which binary value on the gate's output. For a gate of n inputs, the function multiplexer would be a 2^n to 1 multiplexer. Note that our model puts no restrictions on the function of the gates. That is, the function of the gate can be any Boolean function. However, if the attacker has knowledge of the gate library used, he can put restrictions on the configuration variables that determine the gate's function. For example in Figure 4.2 if the attacker knows that the 2-input gate could only be $NAND$ or NOR, then he can restrict the multiplexer's input values to "1110" (for $NAND$ gate) and "1000" (for NOR gate) by adding clauses to the SAT problem to disallow all other combinations.

4.3 Learning from Voltage Probing

Adding more constraints and knowns to the SAT problem can make it easier to solve. One approach that can help the attacker with reverse engineering is a semi-invasive technique called laser voltage probing (LVP) [74, 113]. In laser voltage probing, the target transistors are illuminated and the signal values are inferred based on the measured emitted light. Two broad classes of voltage

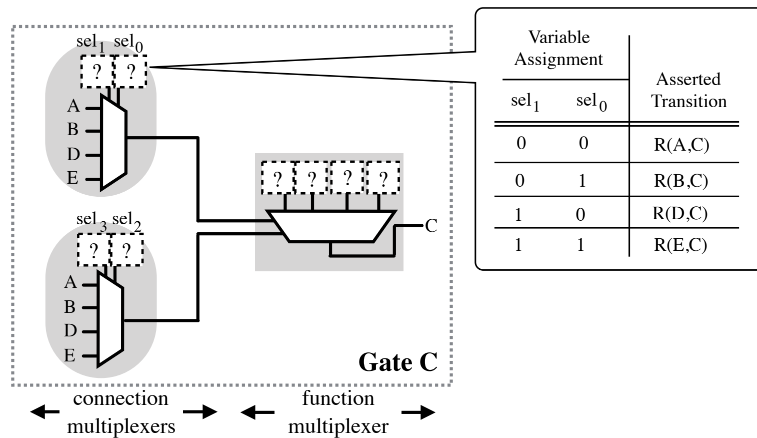


Figure 4.2. An example of the proposed gate model. Depending on the values of the configuration variables, this model allows each gate input to be driven from any node, and allows the gate to implement any possible logic function over its inputs.

probing are frontside and backside. The frontside of the chip is the side of metal layers while the backside is the side of the substrate. With the growing number of metal layers on the frontside, backside probing may seem more promising as it keeps the metal layers intact and preserves the proper functionality of the circuit [67]. Preparing the chip for frontside probing requires decapping the chip by removing epoxy and blocking metal layers to access the internal signals or transistors while backside probing only requires simple thinning and polishing from the back [20, 110].

Having access to the value of internal signals can also help make inference about the possible connections between gates. Even when gate functions are unknown, it is known due to the nature of circuits that each gate instance must implement a deterministic Boolean function; in other words, any gate must always map the same gate input value to the same gate output value. Access to internal values allows an attacker to check whether some candidate connections would violate this condition. Any connections that cause functional consistency of a gate to be violated can be ruled out from further consideration as the inputs to that gate.

input vector	circuit nodes A	B	C	X
0000	1	1	0	0
0001	0	1	0	1
0010	1	1	1	0
0011	0	0	1	0
0100	0	1	0	1

(a) Probed node values.

AB	X
00	0
01	1,1
10	
11	0,0

AC	X
00	1,1
01	0
10	0
11	0

BC	X
00	
01	0
10	0,1,1
11	0

(b) Gate truth table under different connections.

Table 4.1. Example showing that probed values can rule out certain connections. Among the three possible node pairings that could be the inputs of the gate producing node X, one of the three is non-deterministic and can be ruled out.

As a demonstration of how probing can eliminate some candidate connections, consider the example of Table. 4.1 that shows the values of selected nodes when five different primary input values are applied to a circuit. Assume in this case that the attacker knows that node X is the output of a 2-input gate and nodes A, B, and C are other nodes in the circuit. Without knowing the connections of the circuit, the attacker knows only that the inputs to the 2-input gate that produces X are either (A, B), (A, C), or (B, C). The different combinations of gate input values shown in Table. 4.1b are a result of applying different primary input vectors to the circuit and can be observed with laser voltage probing, along with the corresponding output value of the gate. It is possible that multiple primary inputs induce the same combination of gate input values; however, the value of the gate's output node "X" should remain the same when different primary inputs induce the same resultant gate inputs. Looking at these truth tables in Table 4.1b, the attacker can see that it is impossible for the gate input connections to be (B, C), because X takes different values in the three primary input vectors that induced (B, C) to have the values (1,0). Input combinations (A, B) and (B, C) both imply a consistent (deterministic) function for the gate,

so neither of these can be ruled out. Note that our pair notation is not ordered; in other words, $(n_i, n_j) = (n_j, n_i)$.

Using probed values to rule out infeasible input combinations leads to, for each gate, a set of feasible input pairs. If the set of nodes in the circuit is denoted as N, for each node $n_x \in N$ that is the output of a 2-input gate a set of feasible input pairings $(F(n_x))$ can be calculated as shown below, where n_i^j is the value of node $n_i \in N$ when the jth input vector is applied to the circuit.

$$F(n_x) := (n_y, n_w) \in N^2 \mid \big((n_y^i, n_w^i) = (n_y^j, n_w^j)\big) \Rightarrow \big(n_x^i = n_x^j\big) \tag{4.1}$$

Due to the implementation of the SAT formulation, it is easier to allow or disallow single wires instead of pairs of wires. Therefore, we allow as the possible inputs to each gate n_x, the set of all nodes that appear in any of the feasible node pairs in $F(n_x)$. In principle, the input from logic probing is redundant to the information that will be available in the SAT formulation. In other words, the same functional inconsistencies being exploited to eliminate connections would eventually lead to conflicts in the SAT formulation that would prevent the solver from choosing the infeasible connections. However, limiting the number of feasible inputs for each gate as a pre-processing step is a simple way to reduce the size of the SAT problem. To keep the pre-processing simple, we only apply the function consistency check on 1-input and 2-input gates so that we would be able to consider all combination of wires as their inputs. Checking function consistency for all possible combinations of wires quickly becomes time-consuming when the gates have more than 2-inputs.

We apply the proposed pre-processing approach to an 8-bit AES S-box circuit with 501 gates, among which 194 are 2-input gates. Figure 4.3 shows the reduction in the number of feasible input pairs for 50 of the 2-input gates that

are chosen at random. As more primary inputs are applied to the circuit, many pairs of wires can be ruled out as infeasible input pairings for each gate. Overall, The pre-processing approach reduced the number of possible input pairs of 2-input gates from 128,778 to an average of 335, a reduction of more than 99.7%.

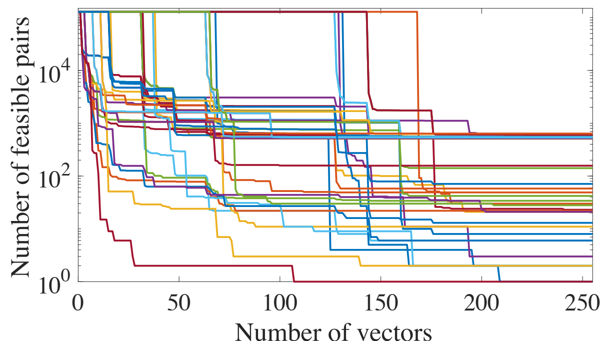


Figure 4.3. Number of feasible input pairs for 50 different gates with respect to the number of primary inputs applied to an 8-bit AES S-box.

4.4 Learning from Fault Injection

Our formulation incorporates the use of fault injection in the reverse engineering process. Just as input/output observations and probing observations provide information that allows an attacker to discriminate between different circuit functions, the results of fault injection experiments provide the attacker with another source of discriminating information. The use of fault injection is important when trying to reverse engineer gate-level schematics because primary input/output observations and probing can be insufficient to uniquely recover the implementation. In our setting, the attacker targets specific nodes as instructed by the SAT solver, but performs each fault injection on a node without knowing the function of any gates or their connections.

In laser fault injection, the attacker can use a setup that is very similar to that used for probing [94]. However, instead of measuring the reflected light

as in probing, he chooses wavelength and energy of the laser pulse so that the photoelectric effect occurs. When focusing the laser beam at a transistor node, an electric current is generated. The induced current might charge or discharge the output of the gate, depending on whether the targeted transistor is a PMOS in the gate's pull-up network or an NMOS in the gate's pull-down network. The ability to inject such single bit errors has been experimentally verified down to 45nm feature size [96]. Given that the duration of laser faults can exceed the clock period, they can be modeled as stuck-at faults in the circuit model.

Masking is an important consideration in fault injection. When the attacker tries to force a 0 or 1 value onto a node for some applied input vector, the induced value will have no effect if it matches the fault-free value of the same node. Similarly, even if the induced value does change the value of the targeted node, it is possible that the changed value may not propagate to the outputs, depending on the connections and gate functions of the remainder of the circuit. Cases where an induced fault changes the output are perhaps the most informative in reverse engineering. In these cases, the attacker learns that the fault-free value of the node is opposite the induced value, and learns the specific output value that is caused by the fault. The information learned from different fault injection outcomes is listed in Table 4.2. For a circuit with 2^m input vectors and x nodes that each can be faulted to induce a 1 or 0, the total number of fault scenarios that can be considered in reverse engineering is $2 * 2^m * x$, and each of the scenarios will correspond to one of the outcomes in Table 4.2. We show in the formulation described in Section 4.5.2 that inference from fault injection can be integrated into a SAT-based reverse engineering framework, and this leaves the deductions shown in Table 4.2 to be made by the SAT solver.

Table 4.2. Each fault injection attempt comprises an applied input vector, a target node being faulted, and a specific induced value at the target node. When a fault is applied, a corresponding output observation is made that is either $o_{correct}$ (if it matches the non-faulted circuit output) or o_{faulty} (if it differs from the non-faulted output). The table summarizes the information that is revealed by each outcome.

condition	output	information learned
SA-1	o_{faulty}	fault-free value of target node is 0 AND fault must propagate to observed outputs
SA-0	o_{faulty}	fault-free value of target node is 1 AND fault must propagate to observed outputs
SA-1	$o_{correct}$	fault-free value of target node is 1 OR fault does not propagate to outputs
SA-0	$o_{correct}$	fault-free value of target node is 0 OR fault does not propagate to outputs

4.5 Extended SAT Formulation

We have previously shown how to model each gate based on the attacker's knowledge about the circuit, as discussed in Section 4.2. In this section, we first show how to enforce gate levelization in our model to restrict the solver to loop-free circuits, and then we show how to incorporate the additional information from voltage probing and fault injection into the SAT problem so that it can be used by an attacker that has these capabilities.

4.5.1 Restriction to Acyclic Topologies

The basic SAT formulation given in Section 4.2 allows for cycles in the wiring connections that would not occur in combinational circuits. The possibility of cycles is problematic because cycles allow circuit nodes to become undefined state variables (that are not determined by the circuit inputs). For a simple example, consider the cycle shown in Figure 4.4, in which node A can have either a 0 or 1 value. When the solver considers a wiring connection such as this one, regardless of the applied inputs the solver has the freedom to assign

any value to node A that will satisfy a SAT formula. This allows the solver to repeatedly find erroneous discriminating inputs, which in reality are not useful in the reverse engineering process.

To avoid this problem, we modify the SAT formulation to disallow cycles while still allowing arbitrary acyclic topologies. Our solution for disallowing cycles is to enforce that the SAT solver only finds solutions in which the topology can be levelized (i.e. topologically sorted). We solve for the circuit's levelization as part of the same SAT formulation that solves for the gate functions and connections. To do this, we add auxiliary variables (to denote levels) and levelization constraints to the SAT problem. Our proposed levelization enforcing approach is not only helpful for our problem and assumptions, but also can help making any SAT attack feasible when there is an uncertainty in connections that would otherwise make a loop in the circuit.

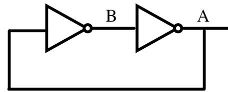


Figure 4.4. Cycles in a circuit

In the conventional levelization definition, levels increase from inputs to outputs, such that the level of each gate must exceed the levels of all gates that provide its inputs. In our formulation, the levels increase from outputs to inputs, but otherwise the levelization notion is the same. Any gate connected to a primary output should be assigned level 1, and the level of every gate must be exceeded by the levels of the gates providing its inputs. In other words, the level of any gate should be higher than the level of all its fanout gates.

4.5.1.1 Encoding Constraints

For each gate g_j in a circuit with n levels, we define a bit-vector of auxiliary variables $(l_0(g_j), l_1(g_j), \ldots, l_n(g_j))$ to encode the level of the gate. The level of the gate is encoded in a thermometer code style, with a number of 0 values followed by a number of 1 values. If bit $l_i(g_j)$ is 0, then gate g_j exceeds level i. If bit $l_i(g_j)$ is 1, then the level of gate g_j is less than or equal to i. Therefore, the level of the gate can be said to be the left-most bit position in which the value is 1. For example in Figure 4.5, the level of g_2 is 2 because $l_2(g_2)$ is the left-most bit position with value of 1. In any legal thermometer coded value, every 0 bit in the vector other than the first must be preceded by another 0 bit, and this is enforced by the encoding invariant shown in Table 4.3. The first and last bit of the level encoding vector must be 0 and 1 respectively for all gates.

Table 4.3. The levelization constraints enforced in the SAT problem to avoid combinational loops

Encoding Constraint	$\forall_{i>0}, g_j : \overbrace{\neg l_i(g_j)}^{level>i} \Rightarrow \overbrace{\neg l_{i-1}(g_j)}^{level>i-1}$ $\forall g_j : \neg l_0(g_j) \wedge l_n(g_j)$
Ordering Constraint	$\forall_{i>0}, g_j, g_k : \left(\overbrace{\neg l_{i-1}(g_j)}^{level\geq i} \wedge R(g_k, g_j) \right) \Rightarrow \overbrace{\neg l_i(g_k)}^{fanin\ level\geq i+1}$
Uniqueness Constraint	$\forall_{i>0}, g_j : \overbrace{\neg l_i(g_j)}^{level\geq i+1} \Rightarrow \overbrace{\bigvee_{\forall g_k} (R(g_j, g_k) \wedge \neg l_{i-1}(g_k))}^{exists\ fanout\ at\ level\geq i}$

4.5.1.2 Ordering of Levels

For the circuit to be levelized, each gate g_j at level i or greater must get its inputs from gates at level $i+1$ or greater. Using transition predicate $R(a, b)$

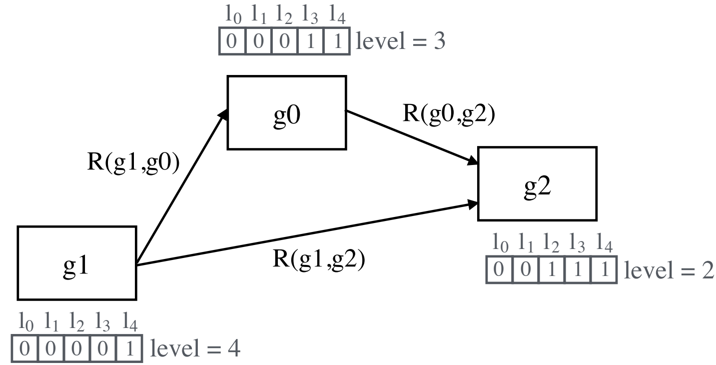


Figure 4.5. Sample levelization encoding in a circuit

(see Section 4.2) to denote a connection from output of gate a to an input of gate b, any legal level-ordering between nodes a and b must obey the ordering constraint in Table 4.3. For example in Figure 4.5, $l_1(g_2) = 0$ (meaning that gate g_2 is level 2 or higher) and we have $R(g_0, g_2) = 1$ and $R(g_1, g_2) = 1$, indicating that both g_0 and g_1 fan out to g_2. Therefore, the ordering constraint enforces that $l_2(g_0)$ and $l_2(g_1)$ must both be 0, meaning that gates g_0 and g_1 are at level 3 or higher.

4.5.1.3 Uniqueness of Levelization

Our ordering constraint prevents cycles, but does not minimize the number of levels, and allows for the skipping of levels as long as ordering is not violated. To minimize the number of levels used, we add an additional constraint that ensures a unique (minimum) levelization. This constraint, shown in Table 4.3 enforces that, for each gate g_j at level i (or higher), at least one of the gates it fans out to must be at level $i-1$ (or lower). When applied to the example of Figure 4.5, given that gate g_1 is at level 4 ($l_3(g_1) = 0$ and $l_4(g_1) = 1$), this constraint requires that either $l_2(g_0)$ or $l_2(g_2)$ should be 0, meaning that one of them should be level 3 or higher. The ordering constraint has already restricted them not to be level 4 or greater, so it follows that g_0 or g_2 must be at exactly

level 3. In this way, the uniqueness constraint, when added to the ordering constraint, ensures that each gate's level is only one greater than the level of one of its fanout gates.

4.5.2 Adding Voltage Probing to SAT problem

In this work we extend SAT attacks to incorporate voltage probing. Voltage probing has the effect of selectively making internal nodes of the circuit visible to the attacker, which in the attacker's model is equivalent to selectively making internal nodes appear as part of the observable output vector. We create an input signal that selectively activates the observability of this node so that, when the SAT solver finds a discriminating input, that input now indicates to the attacker whether any internal signals should be probed when the vector is applied to the Oracle. The corresponding output vector and probed value are fed back into the SAT attack as the additional constraints on the circuit configuration variables.

Probing is illustrated in Figure 4.7 using a model of a circuit with a single internal node C. The newly added input signal $Probe_C$ selects whether the value of node C can be considered when finding a discriminating input vector. Whenever the solver decides to assert the input signal $Probe_C$, then a discriminating vector is one that can produce different values on the primary outputs or the now-observable signal C. Whenever the solver does not assert $Probe_C$, then the approach reverts to the standard SAT attack that discriminates between possible configurations based on the primary outputs only.

4.5.3 Adding fault injection results to SAT problem

In this work, we extend SAT attacks to also account for the attacker's ability to inject faults. To incorporate fault injection results into the SAT problem, the attacker can add the structure shown in Figure 4.6 to all nodes (except primary inputs) in his model, and then allow the SAT solver to guide his fault injection trials as will be

shown. In this structure, $injectFault_x$ is a primary input that selects whether or not a fault should be injected onto node n_x. Primary input signal $FaultVal$ determines the value that is forced onto the selected node. Node n_x_fe is the fault-enabled version of the node, which is either the value computed by the circuit for n_x, or the value forced onto the node by fault injection. Because n_x_fe is the value that fanout gates would see in the circuit, it is this signal that is fed to the connection multiplexers (see Section 4.2) at the inputs of all other gates.

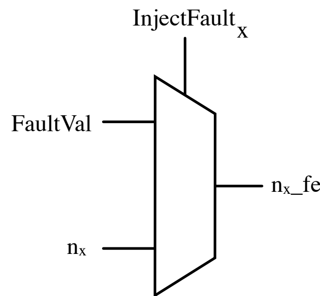


Figure 4.6. The multiplexer that is used to model fault injection for each node n_x.

Figure 4.7 shows how the fault injection mechanism is used as part of the overall circuit model when trying to reverse engineer a simple circuit with three gates. Discriminating inputs produced by the solver now provide the reverse engineer with an input vector to apply to the circuit, as well as a node to fault, and a faulty value to inject on that node. The attacker applies these conditions to the oracle circuit, finds the resulting output vector, and feeds the conditions back into the SAT solver as constraints. The ability to have additional discriminating information through fault injection, can allow an attacker to better distinguish between circuit implementations.

4.6 Results

We have evaluated our approach for two small circuits. The first is ISCAS'85 benchmark circuit c17 comprising 6 gates, 4 internal wires, 5 primary inputs, and

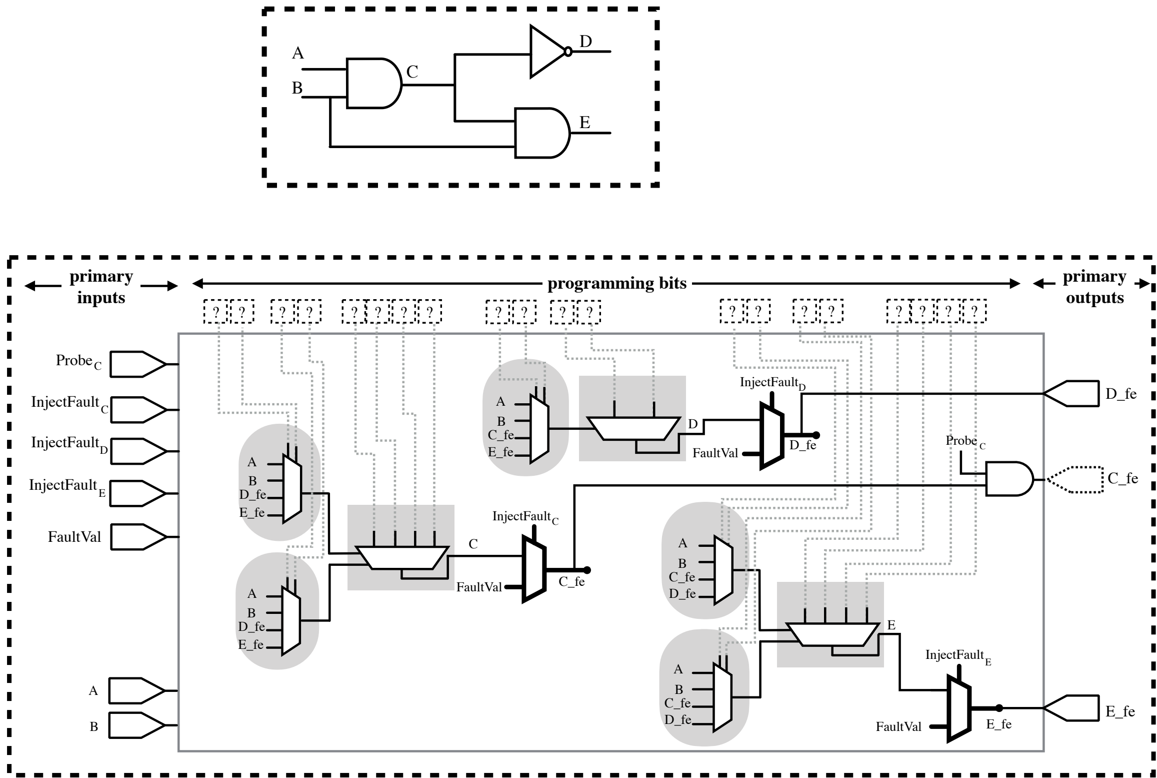


Figure 4.7. The circuit shown on top would be modeled as the one shown on the bottom. The multiplexers in thick lines are used to incorporate fault injection and the AND gate controlled with $Probe_C$ signal is used to add voltage probing into the SAT problem.

2 primary outputs; the second is an S-Box from the PRESENT block cipher [19] comprising 20 gates, 16 internal wires, 4 primary inputs, and 4 primary outputs. For our oracle, we simulate the circuit and perform fault simulation using ModelSim. The attack is performed on a fully-camouflaged netlist where all gates and connections are unknown and modeled as explained in Section 4.5. We perform the SAT attack using a modified version of a publicly available program from existing work [119].

For each circuit, we evaluate the effect of changing different parameters and enabling and disabling features. The results for runtime and number of iterations (i.e. the number of discriminating inputs found) before the algorithm terminates are shown in Tables 4.4 and 4.5, and explained in the following subsections. The

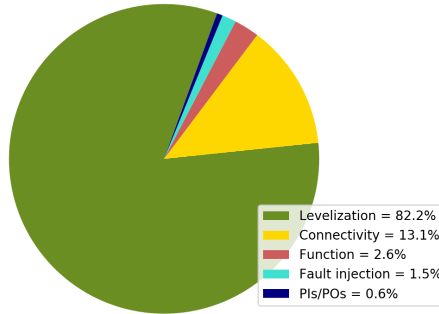


Figure 4.8. Proportion of CNF variables being used toward each component of model in ISCAS'85 c17 circuit.

algorithm terminates when no more discriminating inputs can be found, and at that time, the recovered solution is compared to the original netlist to see whether they are equivalent on a gate-by-gate basis.

4.6.1 Distribution of SAT variables

In all cases, the levelization constraints are found to be necessary for the SAT attack algorithm to terminate successfully (see Section 4.5.1 for details). The majority of the variables and clauses in the SAT problem are used to implement the constraints (see Table 4.3) that enforce levelization. For circuit c17, Figure 4.8 shows the proportion of SAT variables that are used in each of the following aspects of the formulation: The function variables that help with solving gate functions, the connection variables that are created to solve the gate connections, the levelization variables that are created to enforce levelization in the circuit, the fault injection variables that are used to add fault injection capabilities to the model, and the variables related to circuit's primary inputs and outputs and constraints thereof.

4.6.2 Effectiveness of fault injection and probing

As can be seen in Tables 4.4 and 4.5, the problem is not solved within hours if probing and fault injection are not used. When probing is enabled but fault injection is not, the algorithm converges to a solution in a timely manner, but the solution is

not unique, and in the case of the S-Box, it doesn't match the structure of the target circuit that is being reverse engineered. Therefore, probing alone doesn't fulfill our objective of finding solutions that are structurally equivalent to the original netlist on a gate-by-gate basis.

Using fault injection along with voltage probing in reverse engineering makes it possible to find a unique solution for both circuits. In case of the S-box circuit with fault injection and probing, 788562 variables and 5089036 clauses as part of the SAT problem. This solution is identical to the target circuit in all connections and all gate functions. Note that the formulation that includes both probing and fault injection requires more iterations (more discriminating inputs). This occurs because of the large space of fault injection tests and the need to rule out every possible circuit configuration that is not exactly the same as the target, instead of merely ruling out the configurations that are not functionally identical to the target.

4.6.3 Adding Additional Constraints

Our experiments also study the impact of applying the function consistency approach discussed in Section 4.3 as a preprocessing step. In this approach, it is assumed that all nodes are probed for all input vectors. Each of the 2-input gates has 90 possible input pairings in c17 and 253 pairings for the S-Box circuit. Once function consistency is enforced, an average of only 2.3 pairs and 10.6 pairs remain feasible per gate for c17 and S-box circuits, respectively. We then modify the circuit model to disallow a connection between any two gates if that connection does not exist in one of the feasible pairs. We find that this additional preprocessing step has only a modest improvement on runtime, and in some cases the runtime increases slightly. Note that putting constraints on the connections not only reduces the number of connectivity variables, but also will reduce the number of levelization variables; and therefore can

Table 4.4. Results for c17 circuit

fault injection	probed nodes	func. consistency	CPU time(s)	iterations	unique?	CPU time (with gate function limitation)
✗	✗	✗	timeout*	-	-	timeout*
✗	✓	✗	5.990	12	Yes	6.425
✗	✓	✓	0.717	9	Yes	0.864
✓	✗	✗	874.894	34	Yes	328.623
✓	✓	✗	9.178	24	Yes	8.086
✓	✓	✓	1.3293	14	Yes	1.068

* Timeout is considered after 16 hours.

Table 4.5. Results for the 4-bit PRESENT S-Box

fault injection	probed nodes	func. consistency	CPU time(s)	iterations	unique?	CPU time (with gate function limitation)
✗	✗	✗	timeout*	-	-	timeout*
✗	✓	✗	1029	16	No	485
✗	✓	✓	1198	16	No	409
✓	✗	✗	timeout*	-	-	timeout*
✓	✓	✗	611	54	Yes	438
✓	✓	✓	416	44	Yes	308

* Timeout is considered after 24 hours.

have a significant impact on reducing the complexity of SAT problem as both of these are the main contributors on the total number of variables.

Additionally, we also consider the scenario in which an attacker knows the set of gate functions that might be used. In our case, we do this by restricting all gates to implement only functions that are among the 37 gate functions in our gate library, which consists of 2 1-input, 7 2-input, 18 3-input and 19 4-input gates. As shown in the rightmost column of Tables 4.4 and 4.5, we find that this additional constraint can offer a speedup, and could be a viable strategy for the attacker if he has some partial information about gate functions that he is attacking.

4.7 Conclusions

This chapter of the thesis introduced a SAT-based invasive reverse engineering technique that uses probing and fault injection for deobfuscating a circuit. Starting with no knowledge about the gate functions or how they are connected, this approach provides the reverse engineer with a specific set of fault injection and probing experi-

ments to perform on the obfuscated circuit that will allow him to eventually resolve all of its unknown gate functions and connections. Moreover, a new function consistency approach for voltage probing is proposed that can resolve unknown circuit connections without knowing the logic function of any gates. Unlike existing SAT attacks, we show that our approach can recover the exact gate-by-gate netlist of the obfuscated circuit.

CHAPTER 5

SECURE KEY STORAGE IN HARDWARE RESILIENT TO REVERSE ENGINEERING AFTER FABRICATION

5.1 Introduction

This chapter of the thesis deals with the problem of storing a secret key in hardware that is resilient to reverse engineering after fabrication. One of the applications of a hardware secret key is providing a secure hardware storage for storing the secret key of a cryptography core. Additionally, the hardware secret key can be used for the purpose of hardware obfuscation while adhereing to Kerckhoff's principle [1] in separating the secret key from the design. In our proposed key storage approach, the attacker is allowed to know everything about the design except for the characteristics of certain transistors that determine the key. The advantages of separating the secret from the design's hardware as a secret key will be the following:

1. Modularity: Involving an arbitrary logic circuit in obfuscation comes with extra difficulties for design and test. However, if the key is separate from the logic, the interactions between obfuscation and logic are minimal and well-defined. This allows for a detailed exploration of key obfuscation without regard to its impact on logic.

[1] An important idea in Cryptography is Kerckhoffs' principle – that security should rely only on the key, and it should not matter whether an adversary knows the algorithm. In other words, a system should remain secure even if everything except the key is public knowledge. A common restatement of this principle is to avoid "security through obscurity."

2. Generality: A key is perhaps the most general type of information to hide on a chip, and an obfuscated key can also be used in a straightforward way to obfuscate logic [116].

3. Use of error correction: Most importantly, a separate obfuscated key storage allows for the use of error correction, which is not possible when logic is obfuscated directly. We will show that error correction is crucial for allowing the threshold differences to be small enough to fool the attacker without compromising reliability.

5.1.1 Related Work

5.1.1.1 Threshold Voltages to Prevent Reverse Engineering

The modification of threshold voltages can make transistors permanently on/off or can adjust their characteristics in a way that will change the function of a logic gate [28, 35, 85]. Design automation for logic obfuscation can be used to deploy the aforementioned modified transistors as basic library cells [63]. Note that all obfuscations techniques of this type induce the same logic function on all chip instances; this provides an attractive high-value target for a determined attacker. Although threshold voltages cannot be learned from optical analysis during reverse engineering, techniques do exist which can measure thresholds. Sugawara et al. [102] show that it is possible to invasively read out information about transistor doping of the type utilized by Becker [16].

Similar to obfuscated logic that can be attacked invasively, data stored in flash memory or antifuses can be attacked and read out invasively when a chip is not powered. This is especially troublesome in the case of high-value master keys that must be stored on many instances of a chip. To mitigate the threat of invasive readout, Valamehr et al. present a scheme to protect data from invasive attack [109]; in their work, an attacker is required to read out a large number of cells correctly in order to

extract the key, but the information read by the attacker is still digital information. By contrast, the approach we will present uses distributed analog secrets to prevent attack, in order to offer even stronger defense by not having any sensitive digital artifact that can be attacked at rest.

5.1.1.2 Physical Unclonable Functions for secure keys

PUFs are device-tied secrets that are dependent on the unique process variation of the device. They are repeatable (can be regenerated via device evaluation under the same situation), yet unpredictable (one cannot tell what the secret would be, without evaluating the PUF). PUFs are generally used for two applications: device authentication and key generation [50]. Several PUF schemes have been proposed over the years, such as Ring-Oscillator PUFs [103] that uses variations in the delay of identically implemented ring oscillators, SRAM PUFs [44, 51] that exploit the inherent threshold variation of the cross-coupled SRAM cells and butterfly PUFs [70] that use contention on cross-coupled latches to generate output bits. There also exist a number of FPGA-specific PUF designs, such as [14] and [108] that leverage the difference in feedback paths of two identical SLICEs in Xilinx Zynq-7000 FPGAs.

To improve the randomness and correct the inherent noise from a PUF response, the output of the PUF cannot directly be used as a key and helper data or fuzzy extractor is used. The helper data is made publicly available and should reveal limited information about the secret key. However, it has been shown that helper data can be exposed to manipulation attacks [15, 30, 31] and hence leak information about the secret key. Note that our approach does not require a public helper data and is inherently immune to these attacks.

5.1.2 Proposed Approach

Figure 5.1 shows the overall key generation process from the designer and attacker perspective. The design consists of a block containing cells with modified threshold

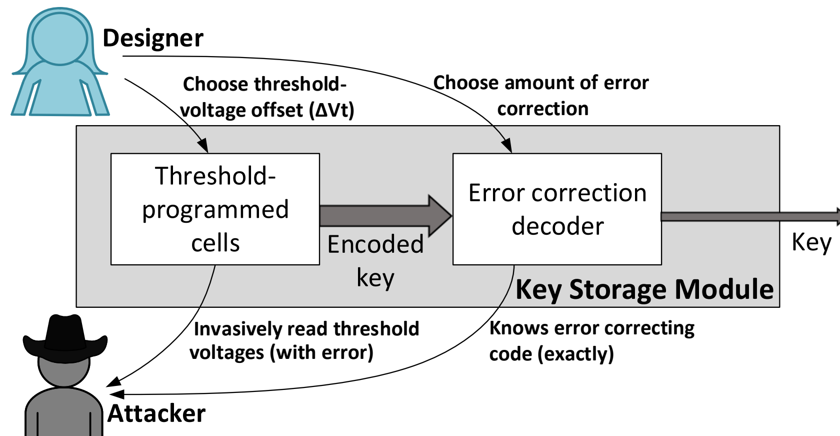


Figure 5.1. The overall design approach, both from the designer's and attacker's perspective

voltages, which is used to generate the encoded secret key when it is needed. The output of this block is given to an error correction block, which decodes the encoded secret key in a way that tolerates errors. The designer can choose the threshold offset of cells to store the encoded secret key, and can choose the parameters of the error correction codes to ensure key reliability. The attacker is allowed to know the error correction used, and knows the mechanism that the designer uses to configure the cells; the attacker can even invasively measure the threshold voltages of the cells, but does so imperfectly. If the attacker is able to get enough information about the cells, then he will be able to guess the encoded key accurately enough to produce the secret key by applying the known error correction scheme to it.

The specific contributions of this work are as follows:

- We present the first approach that combines threshold-based obfuscation and error correction.
- Our proposed key generation approach does not require a helper data, which makes it immune to helper data manipulation attacks.

- We show that this approach leads to quantifiable protection against invasive readout using a very conservative attacker model that only assumes some amount of imprecision when invasively measuring device threshold voltages.
- We give a CAD flow for deploying the proposed approach in a way that can achieve various tradeoffs between reliability, security, and cost.

5.2 Sketch of Approach

Figure 5.2 shows our overall design flow. An engineer can use this framework to implement obfuscated keys that achieve desired tradeoffs of cost, reliability, and security. Each step of the flow is described in detail in the following sections. In the first step the designer chooses a cell type to use for storing the obfuscated constants, and chooses a threshold voltage offset to use for biasing the cells to generate codewords of the encoded key; from this, a cell reliability model is extracted. In the second step, the designer uses the cell reliability model and the chosen key reliability criteria to decide which error correcting code strengths are compatible with the circuit design. A candidate design then exists, and in the third step its security against invasive readout attack is quantified. Depending on whether the security level is deemed adequate, the design can be revised. Examples of revision can be to trade cost against security by decreasing threshold offset and increasing error correction, or trade reliability against security by using a weakened error correction.

5.3 Threshold-Based Key Storage Elements

Threshold voltages of transistors are commonly chosen for power-performance tradeoffs, but recent works have shown that modifications to threshold voltages can also be used to determine the logical function of cells. The basic idea behind these works is to use multiple classes of transistors with different threshold voltages. The modification of threshold voltages can permanently make transistors on/off or can

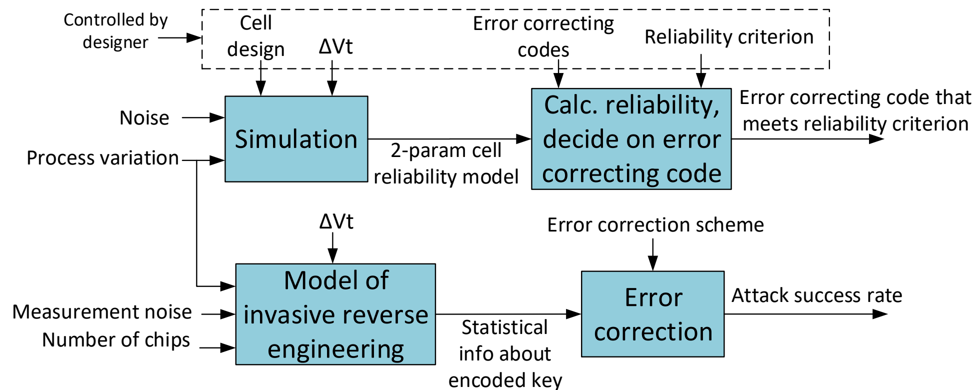


Figure 5.2. Overall CAD flow of the work

adjust their relative characteristics to cause the circuit to implement a specific function [28, 35, 63, 85].

There exist a number of ways in which threshold voltages can determine digital values produced in a circuit. In another context, intrinsic variations in threshold voltages have been used to create device-tied identifiers [73, 99] or secret values in PUFs. It has been previously shown that the power-up state of an SRAM cell depends on the intrinsic threshold voltage differences between transistors which are caused by process variation [51]. In the same way that intrinsic differences in threshold voltages can randomly bias cells toward generating specific values, intentional differences in threshold voltages can bias cells toward specific values in a way that is common across chip instances.

Consider a designer that wants to modify the 6-T SRAM cell of Figure 5.3 so that it will generate a certain value each time it is powered up.

Without loss of generality, we assume the desired state is the 1 state ($Q = 1, \overline{Q} = 0$) while noting that the 0 state works the same way due to the symmetry of the cell. For simplicity, we assume the designer wants to induce the desired state by changing the threshold voltage of only one transistor in the cell. Then the question arises regarding which transistor should be changed, and by how much should its threshold be changed.

The designer can change the threshold voltage of one transistor in the cell in the following ways to bias the cell toward the 1 state: *1)* increase the magnitude of threshold voltage on $N1$; *2)* decrease the magnitude of threshold voltage on $N2$; *3)* increase the magnitude of threshold voltage on $P2$; or *4)* decrease the magnitude of threshold voltage on $P1$. Regardless of which transistor threshold is modified, a larger magnitude change will make the cell more reliably biased, but will also give the attacker a better chance of correctly measuring the threshold difference invasively during reverse engineering. The designer, therefore, seeks to maximize the reliability that can be obtained for a given amount of threshold offset.

To determine which transistor should be modified, we evaluate the 1-probability of the SRAM cell versus its threshold offset. For any threshold offset, the 1-probability shows the fraction of cells that are biased toward producing the desired 1 state after process variations are added. The evaluation is based on 1000 Monte Carlo simulation instances of an SRAM cell in HSPICE using 45nm CMOS Predictive Technology Model [11] (PTM) with a nominal threshold voltage of 469mV for NMOS and -418mV for PMOS transistors. We consider the standard deviation of threshold voltage distribution to be 30mV. The result of this comparison is shown in Figure 5.4. It can clearly be seen that biasing the threshold of PMOS transistor results in a higher 1-probability. Therefore, we conclude that adding a threshold offset on the PMOS transistor is a more effective way to influence the value generated by the cell, compared to the same threshold offset on an NMOS transistor. Based on this analysis, the threshold-based cell programming that we use is to increase the magnitude of $P2$ to induce a 1 value in the cell (amongst the three other choices). Because of the symmetric structure of an SRAM cell, a 0 is stored in a complementary way, by increasing the magnitude of $P1$.

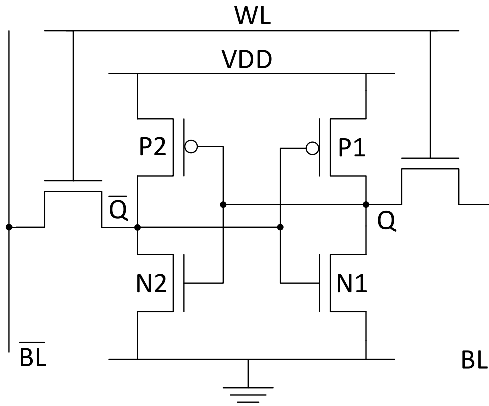


Figure 5.3. A simple 6T SRAM cell. The cell is biased toward the 1-state by increasing the magnitude of transistor P2, and biased toward the 0-state by increasing the magnitude of transistor P1.

5.3.1 Fabrication Questions

We have shown that it is more beneficial in terms of value stability to choose PMOS over NMOS transistors. Therefore, our technique requires two different kinds of PMOS devices with two different thresholds to choose from. One is nominal, and we assume in several places that the second is a threshold of our choosing. Multi-threshold processes are common, but different fabrication processes will typically offer fixed choices for thresholds. There are no technical barriers to having the second threshold be arbitrary, and for the sake of exploring the achievable limits of obfuscation, we will assume fabrication cooperation that allows us to freely choose a threshold. For a slightly more granular approach, a designer can choose among a discrete number of thresholds that are available in existing commercial processes.

5.4 Reliability of Threshold-based Keys

In cryptography, even a single key bit upset may cause discernible consequences, and threshold-programmed values are inherently unreliable due to phenomena such as noise and process variation. For this reason, our scheme uses error correction in

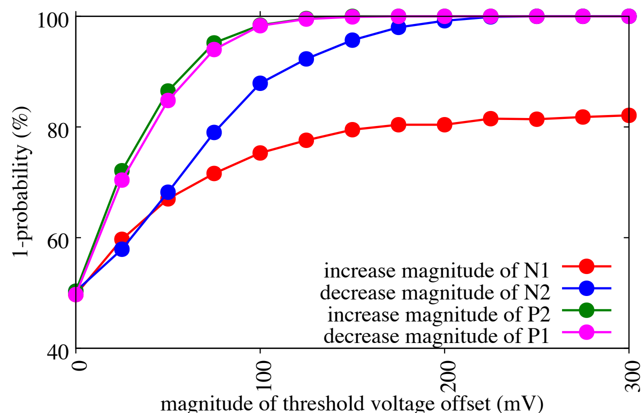


Figure 5.4. The 1-probability against different threshold voltage offsets for a PMOS and NMOS transistor

addition to the threshold-programmed cells. The parameters of error correction and the cell threshold offsets must be chosen together to ensure that the key meets a reliability criterion; a larger threshold offset improves cell reliability and allows weaker error correction to suffice, while a smaller threshold offset will require a correspondingly stronger error correction. Due to variations across chips, the chips will not all have the same key failure rates. Any reliability criterion must therefore specify both a key failure rate and a fraction of chips that must have key failure rates below that number. The reliability criterion that we use is that at least 99% of chips must have a key failure rate of less than 10^{-6}. The error correcting code selected for any threshold offset must cause this criterion to be satisfied.

Because key failures are such infrequent events, it is not possible to check whether a design meets the given reliability criterion using random simulation alone, so we rely on a careful combination of simulation, modeling, and statistics. The scheme we use to check reliability is a two-step process. The distribution of cell error probabilities is first captured in a two-parameter abstracted model. The model of cell error probabilities is then used within a procedure that calculates the distribution across chips of the key failure rate for different error correction schemes.

5.4.1 Distribution of Error Probabilities across Cells

Each cell is biased to produce a single 0 or 1 bit of a codeword as chosen by the designer, but due to noise and variability, it may not produce this desired value in a given trial. In fact, due to process variations, some cells may almost never produce the desired value, while other cells will produce it sometimes or almost always. Circuit simulation is used to learn the distribution of cell error probabilities for a given threshold offset.

Our baseline data for cell reliability is generated using HSPICE simulation of SRAM cells in 45nm Predictive Technology Model (PTM). We created 512 SRAM cell instances with variation on transistor threshold voltages according to PTM, and evaluated each cell in the presence of transient noise 300 times. Noise is captured in the simulations of each instance by doing a single-sample Monte Carlo transient noise analysis with the $.TRANNOISE$ command. From these simulations, a set of empirical cell error probabilities is obtained.

Noting that an SRAM cell with an intentionally offset threshold voltage is similar to a biased PUF, we adopt a modeling approach from PUFs to compute an expression that describes the distribution of cell error probabilities. The most straightforward approach for modeling the behavior of cells is to use a fixed error rate model, where each cell used for key generation is assumed to have the same probability of error. The problem of this simple model is that it cannot accurately capture the behavior of a cell as in reality, some cells are more subject to failure than others. To have more accuracy in our evaluations, we use The heterogeneous error rate model proposed for PUFs by Roel Maes [76]. The model assumes two sources of variation in a cell: The process variable (M) that models the persistent impact of bias and process variations, and the noise variable (N_i) that accounts for the cumulative effect of all noise sources during evaluation. Both variables are normally distributed. The process variable has an unknown mean and variance, while the noise variable is modeled as having 0-mean

and unknown variance. These three unknowns reduce to two unknown parameters λ_1 and λ_2 in the model (Equation 5.1); $\phi(x)$ and $\phi^{-1}(x)$ represent the cumulative distribution function of standard normal distribution x and its inverse, respectively. Parameters λ_1 and λ_2 are chosen by fitting Equation 5.1 to the empirical CDF of cell error probability from circuit simulation using Levenberg-Marquardt algorithm. Figure 5.5 shows the fitting of the model to simulation data for various threshold offsets. Having an expression for the distribution of cell error probabilities (Pe) allows us to sample from this distribution in order to obtain representative cell error probabilities.

$$\mathbf{cdf}_{Pe}(x) = \phi(\lambda_1 \phi^{-1}(x) + \lambda_2) \tag{5.1}$$

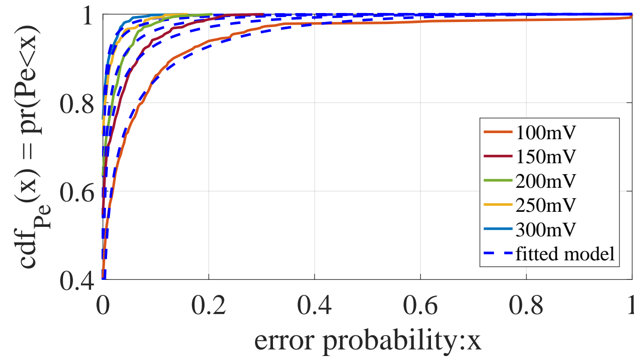


Figure 5.5. Cumulative distribution function of error probabilities from the simulation data and their relative fitted curves for different magnitudes of voltage offsets

5.4.2 Distribution of Key Failures Across Chips

For any threshold voltage offset, using the known distribution of cell error probabilities, we can compute the distribution of key failure rates that will be achieved using different error correcting codes, and can check which codes satisfy our reliability criterion. We focus on BCH codes, which is a class of codes with different block sizes and numbers of correctable errors in each block. We denote a certain BCH code as `BCH[n, m, t]`; where n is the block size, m is the number of useful information bits

per block after error correction, and t is the number of correctable errors in each block. In our setting, n is the number of SRAM bits used to store a portion of the encoded key, m is the number of key bits generated from decoding the n bits, and t is the maximum number of SRAM bit errors that can be tolerated. If an error correcting code is able to correct t bits, the block fails if more than t bits are erroneous.

The number of blocks required to generate a key using a given BCH code will depend on the desired key size (k) and the number of useful information bits from each block in that BCH code (m). The number of blocks needed for key generation is therefore $\lceil \frac{k}{m} \rceil$. The key generation fails if at least one code block that contributes to the key fails. If $P_{Fblock,i}$ is the probability of failure in block i, then the key failure probability is given by Equation 5.2.

$$P_{Fkey} = 1 - \prod_{i=1}^{\lceil \frac{k}{m} \rceil} (1 - P_{Fblock,i}) \tag{5.2}$$

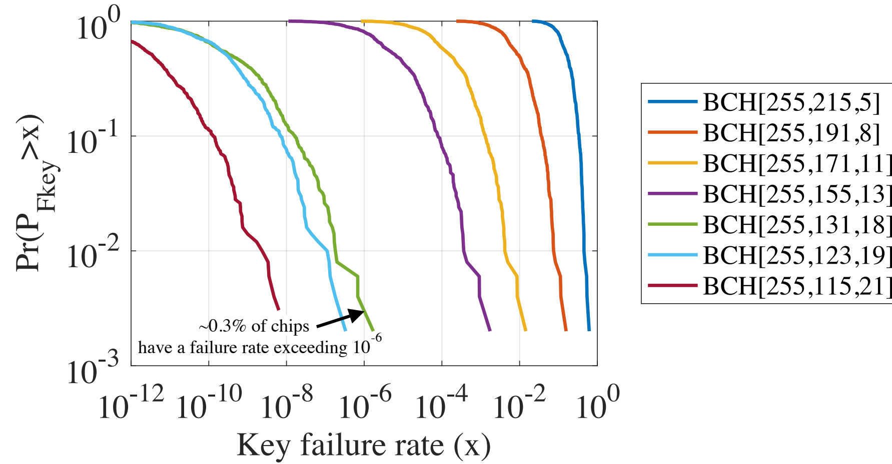
For each block of `BCH[n, m, t]` code, the probability of producing an erroneous result is the probability that the number of errors in that block exceeds t. With a heterogeneous error rate model of cells, each block in a chip will have a failure rate that depends on the unique error rates of its cells. Hence, we cannot use binomial distribution to find failure rate of each block and instead, we use a more general case of binomial distribution, called "Poisson-binomial distribution". The distribution is a discrete probability distribution to calculate the summation of Bernoulli trials that are not necessarily identically distributed. Given a set of n non-uniform cell error rates $P_e^n = (p_{e,1}, p_{e,2}, ..., p_{e,n})$ in a block, the probability of having less than t errors is calculated using cumulative distribution function of Poisson-binomial distribution $F_{PB}(t; P_e^n)$ as shown by Maes [76] and given by Equation 5.3; this describes the probability of correctly decoding the block. Therefore, the failure rate the same block is given by Equation 5.4.

$$F_{PB}(t; P_e^n) = \frac{t+1}{n+1} + \frac{1}{n+1}\sum_{m=1}^{n} \frac{(1 - e^{\frac{-j2\pi m(t+1)}{n+1}})}{(1 - e^{\frac{-j2\pi m}{n+1}})} \cdot \prod_{k=1}^{n}(p_{e,k}e^{\frac{j2\pi m}{n+1}} + (1 - p_{e,k})) \tag{5.3}$$

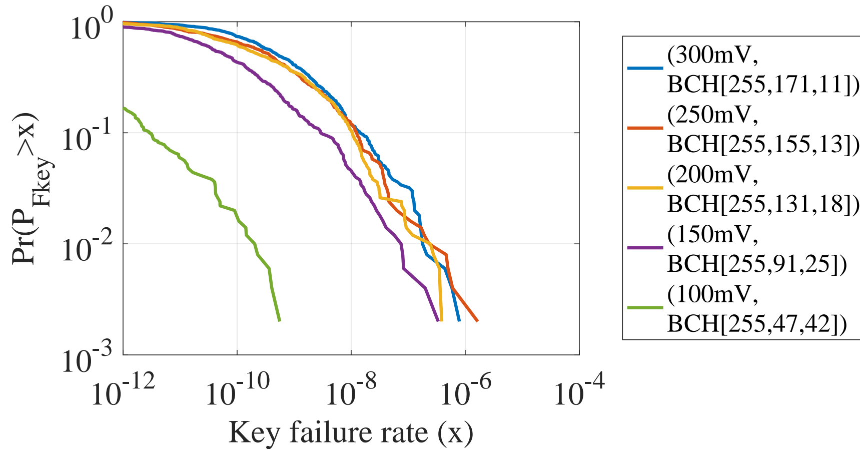
$$P_{Fblock} = 1 - F_{PB}(t; P_e^n) \tag{5.4}$$

We now describe the steps to use the equations given above for evaluating key reliability with a given BCH code and given threshold offset. First, we sample cell error probabilities from the fitted $\mathbf{cdf}_{P_e}$ (Equation 5.1) using inverse transform sampling to obtain a set of n representative cell error probabilities (P_e^n); because the error probabilities are fitted to simulation results, this accounts for circuit-level reliability. We repeat the sampling for the number of required blocks, and then for each one calculate the block failure rate (P_{Fblock}) using Equation 5.3 and Equation 5.4, and use the block failure rates to compute the key failure rate using Equation 5.2. This calculated key failure rate is for one chip instance with a specific combination of threshold offset and BCH code. Repeating the whole calculation multiple times produces the distribution of key failure rates, and we use that distribution to evaluate whether the combination of threshold offset and BCH code satisfy our reliability criterion of at least 99% of chips having key failure rates of less than 10^{-6}.

Among all the BCH codes that will satisfy our reliability criterion for a given threshold offset, we use only the lowest cost BCH code, which is the one that corrects the fewest errors among all sufficiently reliable codes. A designer can choose from different combinations of threshold voltage offsets and error correcting codes to reach the desired reliability for the key. Figure 5.6a shows the key read failure rate for threshold offset of magnitude 200mV, evaluated for different BCH codes. As can be seen, the least expensive code that meets our reliability criterion is `BCH[255,131,18]`.



(a) Key failure rates of different BCH error correcting codes, for threshold offset magnitude of 200mV



(b) Key failure rates of all (threshold offset,BCH code) pairs that meet the reliability criterion

Figure 5.6. Key failure rates of different design options

Figure 5.6b shows, for each threshold offset, the distribution of key failure rates that occurs when the minimal BCH code meeting the reliability criterion is used. Table 5.1 shows the area of each of these combinations in order to generate a 128-bit key that satisfies the reliability requirement of at least 99% of chips having key failure rate of less than 10^{-6}. For our area overhead evaluations, we used SRAM cell area of $0.345\mu m^2$ as reported in [83] and synthesized the BCH decoders using NanGate

Table 5.1. Evaluation of equivalent-reliability designs. Each pairing of threshold offset and BCH code are chosen such that the BCH code is the lowest cost code that will satisfy the reliability criterion for that threshold offset.

$\Delta_{vt}(mV)$	100	150	200	250	300
BCH code parameters (n, m, t)	[255,47,42]	[255,91,25]	[255,131,18]	[255,155,13]	[255,171,11]
Number of cells to store encoded key	765	510	255	255	255
Cells area overhead (μm^2)	264	176	88	88	88
BCH decoder area (μm^2)	61403	40723	31428	24835	21602
Total area (SRAM cells + BCH decoder (μm^2))	61667	40899	40899	31516	21690
Attacker success for a single chip (RS_{key})	8.99e-36	1.45e-28	5.26e-13	6.90e-11	7.66e-08

45nm Open Cell Library [12]. The cost of each option is provided in terms of area in μm^2 units. Given that equivalent reliability can be obtained by these different combinations of threshold offset and BCH code, one must consider the implications of choosing among the equivalent-reliability design alternatives. As we will show in the next section, each of these approaches comes with some tradeoff of cost and security. Using a higher threshold voltage offset makes reverse engineering easier, but using a stronger error correcting code comes with more expense in terms of area and power consumption.

5.5 Resistance Against Invasive Readout

If the designer uses the proposed technique to store a key, the first question that comes to mind is how resistant this key is to reverse engineering attacks. As explained in the previous sections, our approach benefits from the use of error correcting codes to correct the impact of noise and manufacturing issues on key values. The strength of this code is chosen in accordance with the threshold offset (Δ_{Vt}); a smaller threshold offset will require stronger error correction to reach its desired reliability. Selecting a small threshold offset makes it harder for a reverse engineer to distinguish between the different measured threshold voltage values, but the stronger error correction can also help the attacker to correct errors in his invasive measurements. This makes it

difficult for a designer to increase security without compromising reliability and leads to a space of trade-offs between reliability, security, and cost that must be considered during design. In this section, we will evaluate the resistance of each design option against reverse engineering.

5.5.1 Attacker Model

We conservatively assume that an attacker knows everything about the encoded secret key except for the key value that the designer has encoded. The attacker knows which cells store the encoded values, and knows that the secret key bits are encoded into the cells by increasing the magnitude of threshold voltage on either transistor $P1$ or $P2$ to encode a 0 or 1 bit. The attacker also knows the parameters of the BCH error correction that is used.

Using this knowledge to reverse engineer the encoded values, the attacker has to somehow guess enough bits correctly that applying the error correction to his guess will produce the key. For example, if the designer added a BCH error correcting block capable of correcting t errors, the attacker's guess of the encoded key must be within t bits of the value that the designer intended to store. The attacker learns about encoded key bits by invasively measuring the threshold voltages of $P1$ and $P2$ to guess whether the cell stores a 0 or 1 value.

Since threshold voltage cannot be learned through conventional methods such as delayering and imaging, most works on multi-threshold obfuscation regard the threshold voltage as being perfectly secure. However, there are still methods such as spreading resistance profiling (SRP) [81], scanning capacitance microscopy (SCM) [68], scanning spreading resistance microscopy (SSRM) [29] and Kelvin probe force microscopy (KPFM) [69] to measure the concentration of dopant atoms in the channel and hence reveal the threshold voltage. However, these methods still have low read

accuracy and high overhead. We evaluate the key stealthiness even for high threshold read accuracies that may not be feasible yet with today's technology.

Regardless of the technique used to invasively measure transistor threshold voltages, there will be some imperfection to the measurements. Measuring the threshold of transistors and their relative values can be a difficult task since the measurement precision of threshold voltages may not be perfect, and even the task of preparing the chip for measurement can be difficult. There are two sources of inaccuracy that limit the attacker's success in reverse engineering the obfuscated key:

1. **Manufacturing Variations**: Process variations cause the threshold voltages of manufactured transistors to differ from the nominal values intended by the designer. The effect of process variation on threshold voltage of each transistor has a distribution of $\mathcal{N}(0, \sigma_{var}^2)$. This is the same process variation model used in circuit simulation in Section 5.3.

2. **Measurement Error**: Regardless of the type of measurements performed by the attacker to read out the threshold voltages, some inaccuracy is inevitable. Measurement error causes the reverse engineer to measure a threshold voltage that differs slightly from the true threshold of the transistor. We model measurement error as $\mathcal{N}(0, \sigma_{err}^2)$.

An attacker can successfully find out the value of an SRAM cell if he correctly guesses the relative threshold value of PMOS transistors that was intended by the designer. If an unlucky process variation messes up the intended relative relation between values of threshold voltages, the attacker still captures a wrong value even if he correctly measures the voltages. As mentioned before, a designer uses error correcting codes to compensate for these probable errors. Knowing which BCH code is used to generate the key, the attacker can use the same algorithm for error correction,

whether to correct the effect of measurement inaccuracies or structural flaws caused by process variation.

Consider the attacker's view of a cell that is designed to store a 1. The magnitude of threshold voltages of $P2$ and $P1$ are $\mathcal{N}(vt+\Delta_{vt},\sigma_{var}^2)$ and $\mathcal{N}(vt,\sigma_{var}^2)$ respectively, because an increased threshold on $P2$ is the mechanism used to create a 1-value. The threshold voltages of $P1$ and $P2$ as read by the attacker with measurement error are $\mathcal{N}(vt+\Delta_{vt},\sigma_{var}^2+\sigma_{err}^2)$ and $\mathcal{N}(vt,\sigma_{var}^2+\sigma_{err}^2)$ respectively. The attacker should guess that the cell stores a 1 value if he measures a higher threshold voltage on $P2$. The difference between the measured threshold voltages of $P2$ and $P1$ follows the distribution of $\mathcal{N}(\Delta_{vt},2\sigma_{var}^2+2\sigma_{err}^2)$, and when this difference is positive, the attacker guesses a value for the cell that is the same as what the designer intended for the cell. The probability (P_{re}) that the attacker will infer the wrong value for the cell is then the cumulative distribution function of $\mathcal{N}(\Delta_{vt},2\sigma_{var}^2+2\sigma_{err}^2)$ evaluated at point $x=0$ (Equation 5.5).

$$P_{re}=\mathbf{cdf}_{\mathcal{N}(\Delta_{vt},2\sigma_{var}^2+2\sigma_{err}^2)}(x=0) \tag{5.5}$$

Figure 5.7 shows the probability, for different values of Δ_{vt} and σ_{err}, of an attacker inferring a value that disagrees with the value intended by the designer. As would be expected, this probability of misreading a cell is higher when the threshold offset (Δ_{vt}) is small, or the standard deviation of measurement error (σ_{err}) is large.

5.5.2 Attacker's Success Rate for Key Readout

To correctly guess the key, the attacker has to guess the encoded key bits with a number of errors that is within the error correcting capacity of the BCH code. Having the probability of cell read error (P_{re}) from Equation 5.5, the number of errors in a block is binomially distributed, and the probability of the attacker successfully reading out a single block of a `BCH[n,m,t]` error correcting code is given by Equation 5.6.

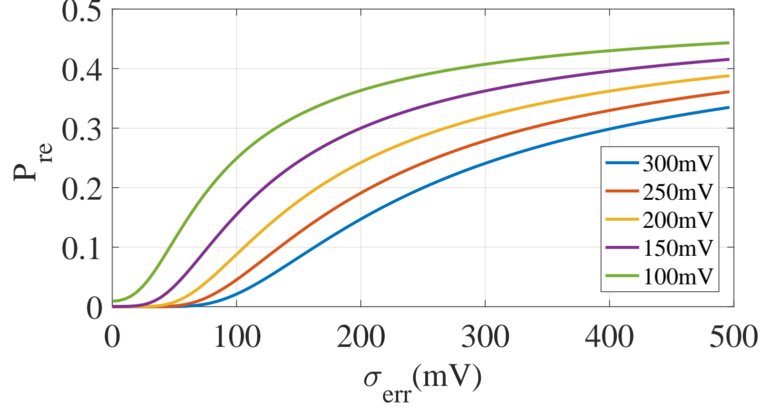


Figure 5.7. For different values of Δ_{vt}, plot shows the probability (P_{re}) that an attacker reads a value for a cell that differs from the value programmed by the designer, as a function of the attacker's measurement error (σ_{err}).

$$P_{RSblock} = \sum_{i=0}^{t} \binom{n}{i} (P_{re})^i (1 - P_{re})^{n-i} \tag{5.6}$$

Given that multiple error correction blocks may be required to generate the entire key, the attacker will only succeed in reading out the key when all blocks are read correctly. The probability of the attacker reading out the key successfully is denoted P_{RSkey} and calculated as shown in Equation 5.7. Table 5.1 reports the attacker success rate (P_{RSkey}) for different threshold offset magnitudes when $\sigma_{err} = 200mV$.

$$P_{RSkey} = \prod_{i=1}^{\lceil \frac{k}{m} \rceil} P_{RSblock,i} \tag{5.7}$$

5.5.3 Cost of Readout by Attacking Multiple Chips

When the same key is encoded in multiple chips, an attacker can choose to attack multiple chips in order to improve accuracy by averaging out deviations in measurement error and process variations. In this case, the attacker sees the differences between the transistor threshold voltages in a cell as $\mathcal{N}(\Delta_{vt}, \frac{2\sigma_{var}^2 + 2\sigma_{err}^2}{C})$, where C is the number of chips measured. Changing the normal distribution of Equation 5.5 to account for this reduced variance leads to a reduction in P_{re} which benefits the attacker. Note

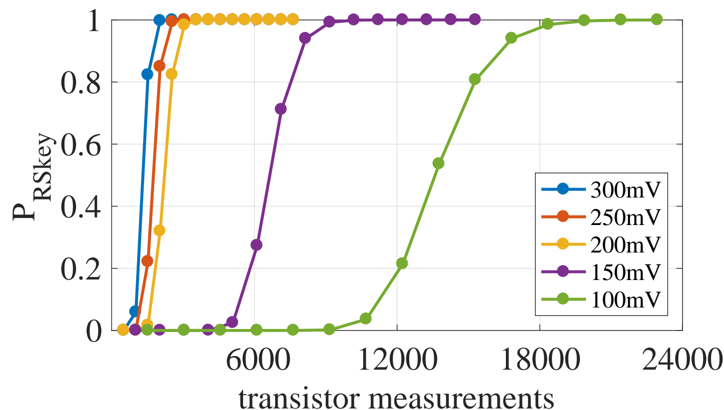


Figure 5.8. Effect of multiple chip measurements on reverse engineering success rate for different threshold offsets with σ_{err} = 200mV

that taking measurements from additional chips is preferable over taking multiple measurements of the same chip, which only reduces measurement noise but not process variations. Depending on the costs of preparing a chip for measurement, there could be advantages to re-measuring a single chip, but we do not consider that here.

Figure 5.8 shows the relation of reverse engineering success rate with the number of individual chips used for measurements for each threshold offset. As an example, one can observe that when the threshold offset (Δ_{Vt}) is 100mV, the attacker has to measure about 13770 transistors to have more than a 53% chance of extracting the key. This requires measuring two transistors from all 6885 cells that store the encoded key on 9 instances of the chip. However, it should be noted that although having more chips increase the attacker's success rate, it also comes with an extra cost of measuring multiple threshold values.

As mentioned before, parameters of a BCH code are denoted as $[n, m, t]$ where n is the block size, m is the size of useful data after error correction and t is the size of correctable errors in a block. For a key of size k that uses BCH blocks of size $[n, m, t]$, a total of $\lceil \frac{k}{n} \rceil$ blocks are used. Therefore, there are $m * \lceil \frac{k}{n} \rceil$ input bits for BCH blocks that are provided by threshold-biased SRAM cells. The reverse engineer needs to

measure the threshold of two PMOS transistors for each cell, making a total number of $2m * \lceil \frac{k}{n} \rceil$ transistor threshold measurements per chip in order to extract the key in this setting.

If the reverse engineer tries to increase the key read reliability by measuring the cell values from C chips, it will increase the number of transistor threshold measurements to a total of $C * 2m * \lceil \frac{k}{n} \rceil$. In this way, using a smaller value of Δ_{vt} combined with stronger error correction has two advantages. By storing information more diffusely, it requires more measurements to be made on each chip, and requires more chips to be attacked before the key can be guessed.

5.6 Design Tradeoffs

Having shown analysis of reliability and security for different design scenarios, we now discuss how a designer can maximize her advantage over the attacker for effective security tradeoffs. While most changes will impact both reliability and security, some will represent more effective tradeoffs for the designer to consider.

5.6.1 Loosening Reliability Constraints

Error correcting code choice is constrained by our reliability criterion which specifies a maximum key failure rate for chips in the first percentile of reliability. In other words, we've specified that 99% of chips must satisfy some reliability bound. If we allow weaker error correction to be used, then the failure rate of chips in the first percentile of reliability will increase. Yet, at the same time, the attacker's success rate for extracting the key will decrease.

To compare the key reliability of the design to the key read success rate of an attacker, Figure 5.9 shows the security versus reliability tradeoff offered by different error correcting codes. This plot analyzes a scenario with a threshold offset (Δ_{vt}) of 200mV, and a low measurement error (σ_{err}) of 100mV. The leftmost point shows the

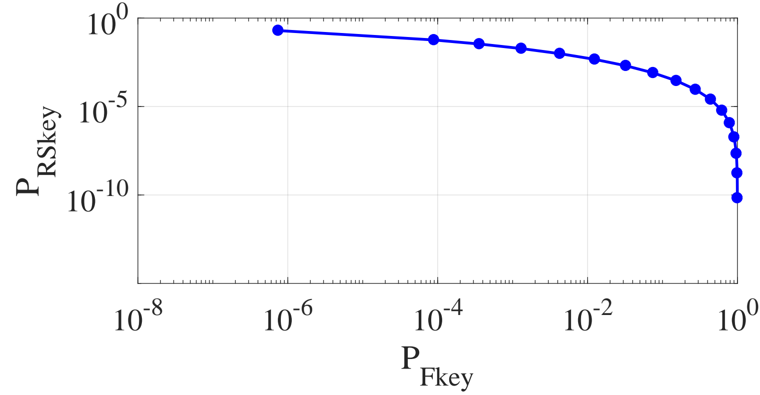


Figure 5.9. Tradeoff between attacker's key read success rate (P_{RSkey}) and key failure rate (P_{Fkey}) that can be achieved by using different error correcting codes.

attacker's high success rate if the BCH code used is strong enough to ensure that 99% of chips have an error rate less than 1E-6, as was used before. If different BCH codes are used, the plot shows how the failure rate of first-percentile chips increases and the attacker success rate decreases, with increasingly weaker BCH codes. This curve represents a set of tradeoffs that a designer can make. Allowing a higher failure rate in key generation may be desirable in some scenarios if higher level error correction mechanisms occur. Note that this particular scenario is one in which the attacker is already able to make highly precise measurements, and that the achievable tradeoffs can be even better in other cases.

5.6.2 Majority Voting

Majority voting using multiple values obtained from each cell provides a way for the designer to mitigate the effects of on-chip noise. This an interesting tradeoff for the designer because on-chip noise, which is detrimental to key reliability, does not present any difficulty to the attacker since his read-out is not based on observing digital values from a functional chip. Therefore, majority voting is an attractive way to improve the reliability of cell values and allow a weaker BCH code to be used, which has the effect of making the attacker's task more difficult without compromising

key reliability. In other words, the designer can strategically replace some amount of algorithmic error correction that helps the attacker, with an amount of circuit-level error correction that does not help the attacker.

5.7 Conclusions

This work presents a methodology for storing obfuscated master keys with quantifiable security against an attacker that knows everything about the design except for the values of the secret key bits. The underlying technique is to combine threshold-based secrets with error correcting codes to allow secrets to be stored diffusely, which gives the designer an advantage over attackers that try to read out the secrets with some amount of imprecision. The proposed methodology enables designers to achieve different tradeoffs of area cost, key reliability, and security against invasive readout.

CHAPTER 6

SECURE KEY STORAGE IN HARDWARE WITH PROGRAMMING BY DIRECTED AGING

6.1 Motivation and Background

The secret key storage approach proposed in Chapter 5 has several fundamental limitations:

1. It assumes that the foundry supports multi-threshold fabrication with arbitrary choice of threshold voltages, which can impact the fabrication cost.

2. It assumes that the foundry is trusted. The designer adjusts the threshold voltage of SRAM transistors based on the secret key that needs to be encoded in the SRAM cells and sends the design to the foundry. Hence, the foundry knows the relative threshold voltage of transistors and can infer the secret key. This can cause security issues in case the foundry is untrusted.

3. It requires that all chip instances will store the same readout resisting key.

The aforementioned limitations show the need for a secure key storage mechanism that eliminates the support of a foundry and enables the IP integrator or users to encode their own secret data. In this chapter, we show that a reliable threshold encoded key can be created by imposing directed accelerated aging on transistors within SRAM cells and build a hardware prototype of such a secure key storage system. Although we envision a system that would ultimately be implemented on a single integrated chip, in experiments, we use commercial off-the-shelf SRAM to represent

the SRAM of the integrated solution, and hardware designs on FPGA to perform the processing that the integrated solution would include.

We first give an overview of our proposed method to secret key storage in Section 6.2. We characterize the power-up values of SRAM cells and conduct experiments to observe the effects of aging and recovery on SRAM cells in Section 6.3. In Section 6.4, we evaluate the key reliability under different error correction parameters and choose the ones that can ensure the design key reliability criterion. We then design and build a hardware prototype of the system and measure over time the reliability of generated keys in Section 6.5. Finally in Section 6.6, a new model-based method for evaluating the security of key against invasive readout is presented that obviates the need for threshold voltage measurements of the experimental data and can adapt to different strengths of measurements from the attacker's side.

6.1.1 Transistor Aging

After a chip is manufactured, some physical parameters of transistors such as threshold voltage change over time with usage. The changes usually manifest themselves as degradation in the performance of integrated circuits. For example, the authors in [13] discuss the delay faults that occur because of the chip speed degradation, and propose a hardware design to predict the circuit failure due to aging.

Although aging is usually considered to be an undesirable phenomenon that impacts performance and reliability, previous work has shown that it can be helpful for some applications, such as reinforcing the value of PUF cells to increase their reliability [17, 18, 80].

The most prevalent types of transistor aging are NBTI (Negative Bias Temperature Instability) and HCI (Hot Carrier Injection) [75]. NBTI aging occurs when a negative bias is applied to PMOS transistors, and interface traps and oxide-fixed-charge are generated by the electrochemical reaction of holes and Si-H bonds at the $Si - SiO_2$

interface [13]. The threshold increase that is induced through aging depends on factors including:

1. The gate-source bias voltage
2. The time for which the bias is applied
3. The working temperature of the chip
4. Characteristics of the technology node, such as gate oxide thickness [13]

Therefore, in cases where the aging effect is desired, e.g. improving PUF reliability as mentioned, it can be accelerated by increasing the temperature of the chip and the gate-source voltage of transistors.

6.2 Proposed Method

Fig. 6.1 shows an overview of the proposed idea. The potentially untrusted foundry is only responsible for providing the user with the necessary infrastructure to store a secret key, but has no knowledge about the key value. In fact, the fabricated infrastructure is general and can be used to store any key value. This infrastructure consists of the SRAM cells that store the key through modifications in their threshold voltages, and an error correction block. The key is programmed after chip fabrication through directed accelerated aging. The aging biases transistors such that the secret key is generated from the power-up values of the SRAM cells. The final secret key is read from the output of the error correction block.

Since the key is programmed by user/IP integrator after fabrication, chip instances need not hold the same key, and different chips can be programmed to store different keys. The unavailability of multiple chips that store the same key can eliminate the possibility of an attacker measuring multiple chips to improve the key readout accuracy, as was suggested in Section 5.5.3. Additionally, similar to the key storage approach in Chapter 5, this approach does not require storage of a secret data to generate the key, which makes it immune to the helper data manipulation attacks.

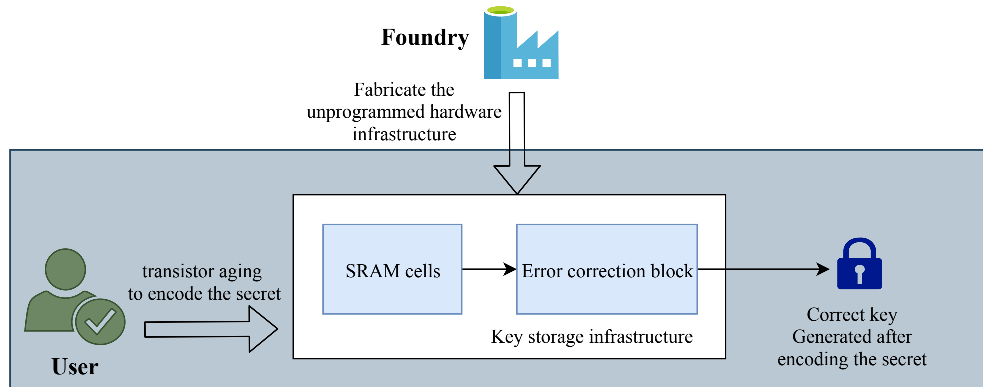


Figure 6.1. Proposed method of storing the secret key. The key storage infrastructure consists of SRAM cells and error correction block that are built by an untrusted foundry. The trusted user/IP integrator encodes the secret data through directed aging of the SRAM cell transistors.

Fig. 6.2 shows the setup that is used for performing experiments on SRAM bias and key reliability in Sections 6.3, 6.4 and 6.6. We use an Arduino Mega to write the required values to SRAM for the directed accelerated aging, and for reading out the SRAM power-up values for key generation. All the analyses were done by different Python and Matlab scripts on the offline data that was read from the SRAM chips.

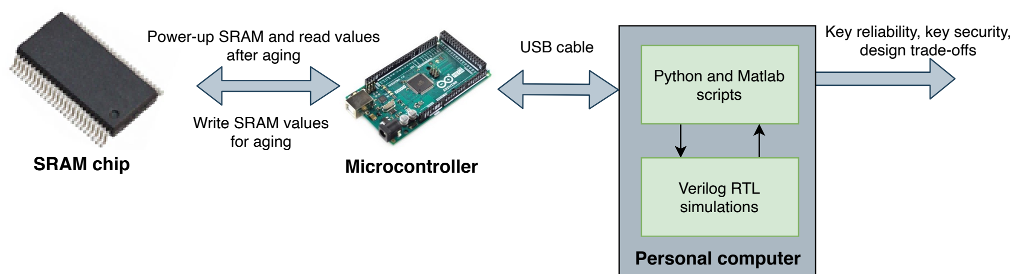


Figure 6.2. Setup to induce aging and perform key evaluations after aging

6.2.1 Error Correction

As previously shown in Chapter 5, if the threshold voltage offset is small, the accumulative effects of process variation and environmental noises can cause the

power-up value of an SRAM cell to be different from its intended value. While considering threshold voltage offset values ranging from 300mV to 100mV, it was shown in Tab. 5.1 that the reliability increases with the threshold voltage offset. The threshold voltage biasing from directed accelerated NBTI aging is less substantial than the chosen thresholds from Chapter 5, and is generally smaller than 40mV [18]. The small amount of threshold voltage change caused by aging requires a strong error correction scheme in order to ensure the key reliability. Different error correction strategies to address the key reliability include:

1. Using BCH codes: We previously discussed BCH error correction in Chapter 5. BCH decoders usually have a significant area overhead, especially the stronger ones that are capable of correcting a large number of errors. Moreover, the small threshold voltage changes caused by aging can result in a large number of errors, such that even the strongest BCH codes would not ensure the desired reliability.

2. Using Repetition codes: Another error correction scheme that can be used is repetition codes, which is achieved by replicating a single bit value over a number of cells. A simple example of a repetition decoder is majority voting, which decides on the final value based on the value produced from the majority of cells. Because of the small area overhead of SRAM cells, the area overhead for creating redundancy is usually low. However, repetition codes typically provide modest reliability improvement compared to BCH codes.

Using repetition codes alone in our scheme would require a large number of replicated SRAM cells to store each bit. On the other hand, a complex BCH decoder with the capability to correct a large number of errors can be expensive in terms of area. A concatenation of these two error-correcting methods will most efficiently meet the reliability criterion, as discussed in previous work [24, 82].

Fig. 6.3 shows how a secret key is stored using our approach. The user determines what values should be written into the cells by encoding their secret key using the BCH encoder. This process is independent of the hardware infrastructure and can be performed in software. The encoded key is then redundantly stored in hardware through directed aging of cells, as will be discussed in the next section. For an M-bit repetition-based error correction, M SRAM cells are biased towards each single bit value of the encoded key, making M replications of the encoded key as shown in blocks of *Rep* 1 through *Rep* M. If the encoded key is of size K, the length of each of these replicated blocks would be K bits, and a total of $M \times K$ SRAM cells are used for storing the key. The reading procedure comprises powering up the SRAM cells, calculating the output of the repetition block from the cells that store the same bit values redundantly, and passing the result to the BCH decoder. The M-bit repetition decoding block and the BCH decoder block together comprise the error correction block and are responsible for correcting the errors in the power-up value of SRAM cells. The key will be generated on the output of the error correction block.

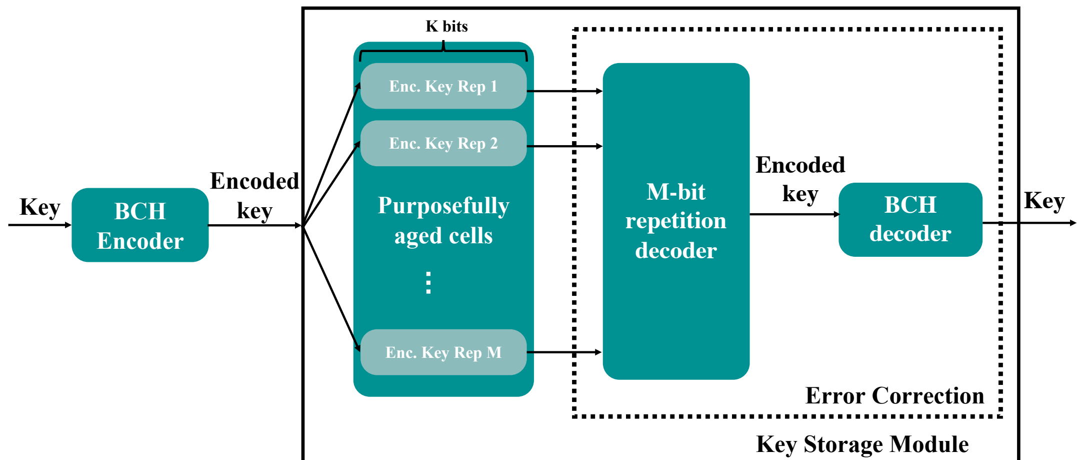


Figure 6.3. Key generation scheme

6.2.2 Inducing Bias in SRAM cells

In our experiments of Sections 6.3, 6.4 and 6.6, NBTI-induced accelerated aging is caused by exposing the devices to a temperature of 85°C in a TestEquity 115A Temperature Chamber [9] for one hour. We use two different models of SRAM chips for these experiments: 23LC1024 from Microchip Technology Inc. [3] and N01S830HA from ON Semiconductor Corporation [8]. Both of these chip models are 1Mbit serial SRAMs. The SRAM chips are powered with 5V supply, which is within the normal range of supply voltage specified in their datasheets. During the accelerated aging, each SRAM cell is programmed to hold the value opposite to the one we wish to generate later at power-up as shown by previous work [51]. To induce aging, the memory chip is placed in the thermal chamber, then once it reaches the desired temperature the chosen values are written to it. The SRAM chip remains powered for one hour in the thermal chamber, during which each cell holds its chosen value. After one hour, the SRAM memory chip is powered off and removed from the thermal chamber.

6.3 Experiments on the SRAM Bias

Our system writes values into the initial state of SRAM using directed accelerated aging. In this section, we examine through experimentation (1) The extent to which we can bias the power-up state of SRAM using NBTI-induced accelerated aging, and (2) How these SRAM devices retain this given bias. The experiments of this section were done in collaboration with an undergraduate student, Peter Stanwicks, that I helped advise in Summer 2018.

6.3.1 Non-uniform SRAM Bias

Before the devices received directed aging, multiple observations of the power-up state were recorded for each device. These initial measurements showed that SRAM

cells exhibit different power-up value probabilities, depending on their address. To investigate this observation, the average Hamming weight of each byte was measured before the SRAM received any bias. The average Hamming weight of a byte is a value between 0 and 8, reflecting the number of bits that hold a value of 1 within that byte. We measured the average Hamming weight for each byte over 25 trials on 9 different chips.

Fig. 6.4 shows the heatmap of the average byte Hamming weights for a single SRAM chip. The y and x dimensions of the heatmap correspond to the most significant bits and least significant bits of addresses, respectively. The colors closer to white correspond to an average byte Hamming weight that is closer to 0, meaning that the bits within the byte are more likely to power-up to 0. A color closer to red is a higher average byte Hamming weight (closer to the value of 8) and corresponds to bits with a higher tendency to power-up to 1. Similar patterns are observed in both models of SRAM devices.

Based on the observed pattern on average byte Hamming weights of SRAM cells, they can be classified into two different groups. The classification threshold is considered to be the midpoint of the lowest (value 0) and highest (value 8) possible values for the average byte Hamming weights, the value 4. If the byte's average Hamming weight is less than 4 the byte is considered in the low group, and if it is greater or equal to 4 the byte is considered in the high group. The distribution of both these two groups can be seen in Fig. 6.5. This categorization will be later used to increase the accuracy of our model-based key reliability evaluation in Section 6.4.3.

The power-up state of an SRAM cell is affected by both its inherent bias and the induced aging. The different pre-aging Hamming weights of SRAM cells can interfere with and mask the effect of directed accelerated aging. In other words, cells with high pre-aging Hamming weight would provide stronger ones if aged towards holding logic-1 and would provide weaker zeros if aged towards holding logic-0. Therefore to

increase the reliability of the keys, the pre-aging values of the SRAM cells should be taken into account when making inferences about their values. We propose a new repetition decoding method that takes the pre-aging Hamming weights into account, which will be further discussed in Section 6.4.1.

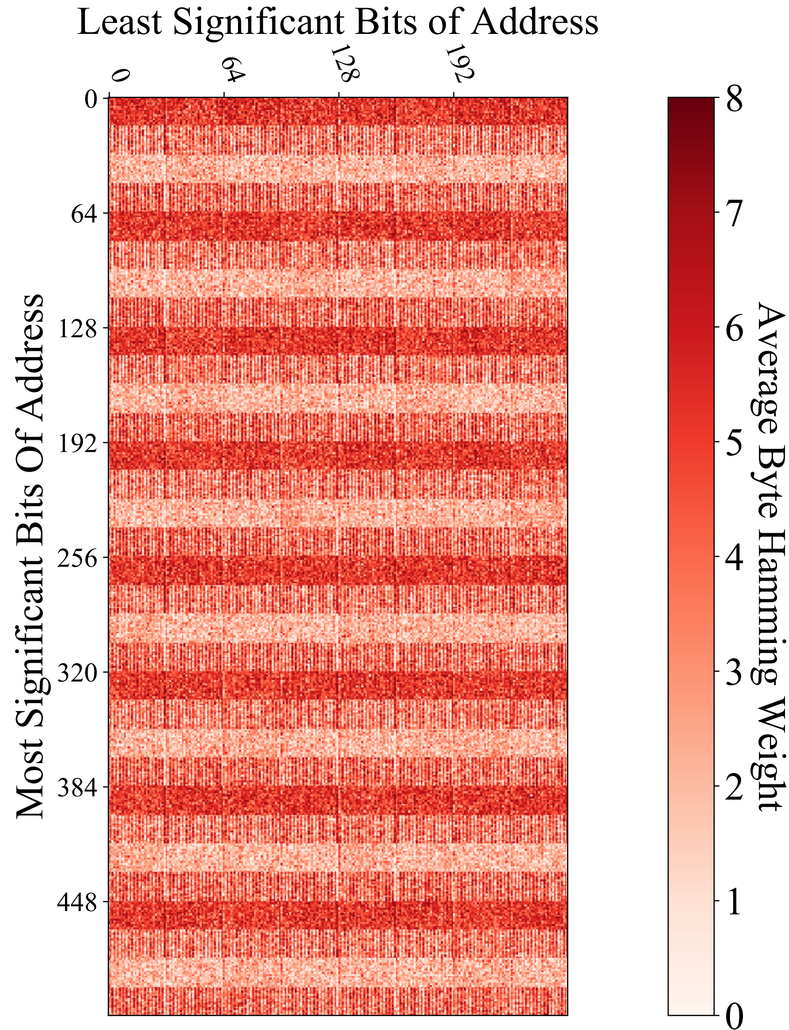


Figure 6.4. Heatmap of SRAM memory before receiving directed aging, sorted by address. Each dot represents the Hamming weight of a single byte as a gradient of 0 to 8. Colors closer to red show a higher byte Hamming weight, meaning the bits are more likely to power-up to "1". The lower byte Hamming weights, where bits have a high tendency to power-up to "0" are shown with whiter dots.

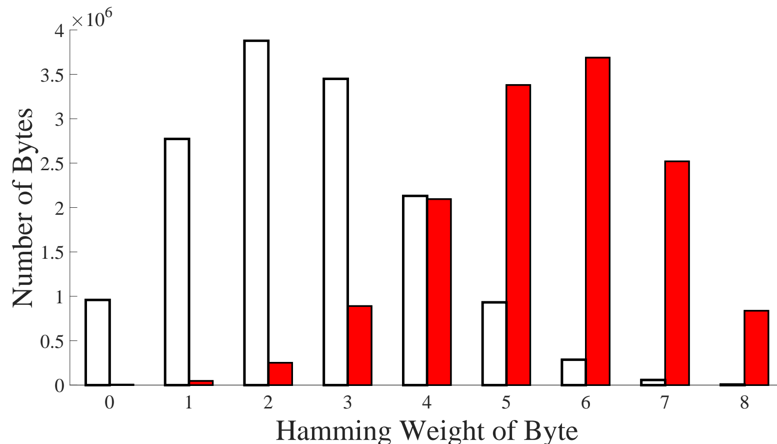


Figure 6.5. Histogram showing the two distributions of byte groups before biasing. The average Hamming weight of a byte is 2.50 in the white distribution and 5.47 in the red distribution.

6.3.2 SRAM Recovery

Over time, some amount of biasing from the aged cells is reverted through a phenomenon called aging recovery. Recovery can threaten the reliability of the biased values and introduce a challenges to utilizing directed aging for encoding reliable secret values. To our knowledge, there is no existing accurate model on the effect of recovery in terms of both active usage and time in SRAMs. The recovery can be observed by comparing the Hamming weight measured at different times following the directed aging.

To investigate the effect of recovery, the chip is first aged as explained in Subsection 6.2.2. To maximize recovery, an active recovery approach [45] is applied to the SRAM cells, in which data that opposes the directed aging is written to the chip and held for a specified duration (30 seconds in our experiments). The chip is then powered off for a specified amount of time t_{off}, and the process repeats. The procedure of this experiment can be seen in Fig. 6.6. The SRAM chips are periodically powered-up to generate their fingerprints, written values opposing their aging, and then powered-off again 30 seconds later. Experiments are performed for four different

t_{off} values: 10s, 100s, 1,000s and 10,000s, each on three different chips. All recovery measurements are performed at room temperature, 22°C. Fig. 6.7 shows the percentage of difference between the average Hamming weights measured after directed aging and the average Hamming weight that was measured before directed aging. Four different cases were considered while studying the effect of recovery: (1) SRAM cells with a "high" Hamming weight natural tendency that are further aged towards 1 (displayed in dark red) (2) Cells with a "high" tendency that get biased towards 0 (light red) (3) Cells with a "low" tendency that get biased towards 1 (dark blue); and (4) Cells with "low" tendency that get biased towards 0 (light blue).

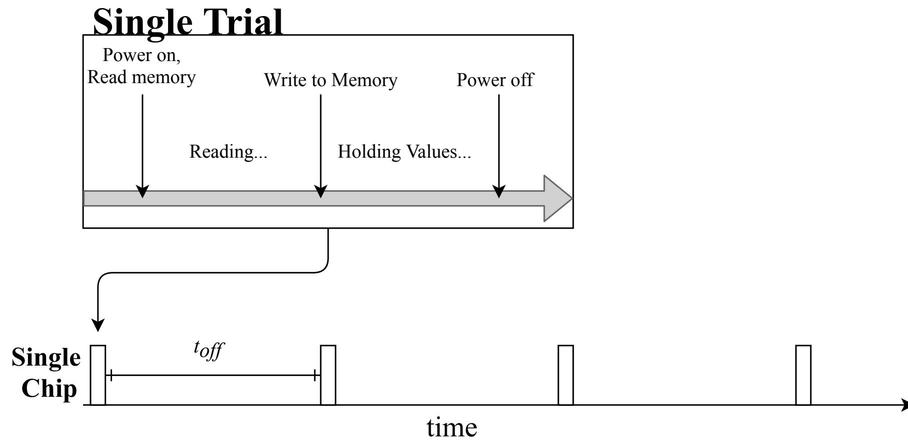


Figure 6.6. Process used to test recovery after directed aging. Parameter t_{off}, the time between power on trials, is varied in different experiments.

Since the natural power-up tendency of each SRAM cell affects the post-aging Hamming weight of their relative repetition blocks, it is best to consider each high and low Hamming weight region (as shown previously in Fig. 6.5) independently when studying the effect of recovery. The experiments are repeated on three different chips, and each line in this figure represents one of them. As can be observed from this figure, the recovery effect manifests itself as the percentage of difference in average Hamming weights getting closer to their relative unbiased values (shown as a black line at center) over time.

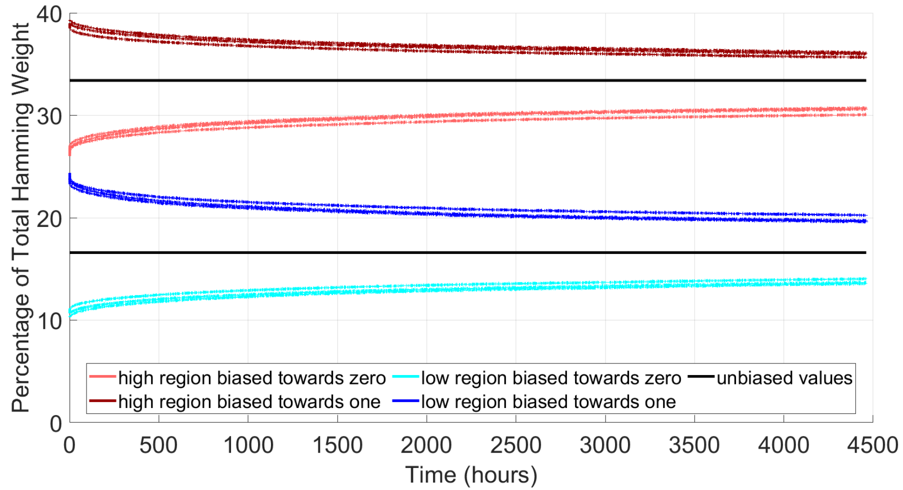


Figure 6.7. Recovery of three SRAM memory devices over 4000 hours with power-on cycles every 1000 seconds. Y-axis shows the percentage of difference between the pre-aging average Hamming weights and the after-aging average Hamming weight.

6.3.2.1 Recovery as a Function of SRAM Usage

We have investigated the effect of chip power-up cycles and the elapsed time on the amount of recovery after aging. Fig. 6.8 shows a comparison between recovery as a function of use time and recovery as a function of the number of power-up trials, specifically examining the sections of memory that received bias towards logic-1. We introduce a metric called percent bias that can be calculated according to Eq. 6.1. In calculation of bias at time t ($Bias^t$), b_j^t represents the actual value of bit j that is observed at time t after the aging, and HW_j is the pre-aging Hamming weight of bit j and is a representation of the natural power-up tendency of the bit. j iterates through every bit in memory with capacity C bits, which is 1M bits in our experiments. All values of bias are normalized with respect to the maximum amount of bias, which is seen at the start of each experiment.

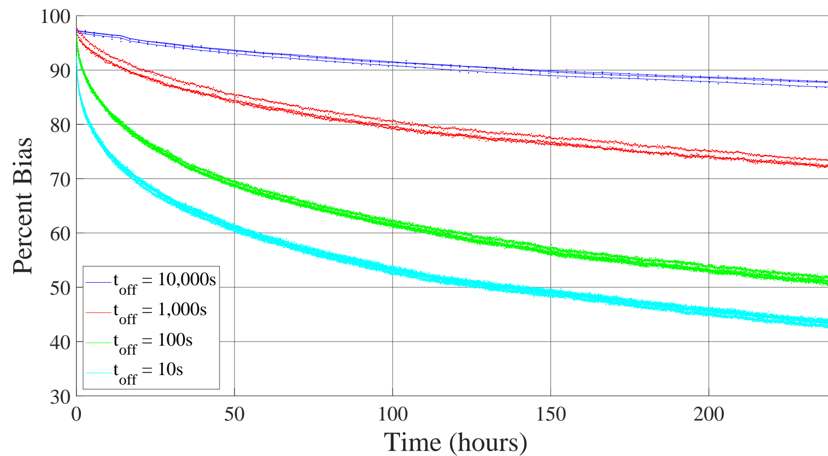
Fig. 6.8 illustrates the effect of recovery using the Percent Bias value from Eq. 6.1. A lower value of Percent Bias indicates a greater amount of recovery. Fig. 6.8a depicts the amount of recovery with respect to elapsed time, showing that the chips that were

powered on more frequently recovered more than chips that were powered on less frequently. For example, the rate of recovery is highest for chips with $t_{off} = 10s$ and lowest for chips with $t_{off} = 10,000s$. Fig. 6.8b shows the Percent Bias changes with respect to the number of power-up trials. It can be seen that the amount of recovery is comparable across chips regardless of the different t_{off} values. This indicates that the amount of recovery is related to how long the SRAM chip is powered, and not to the total elapsed time since the aging. To minimize the amount of recovery, power-gating techniques can be used to deactivate and power-off the SRAM between the key generations to minimize the amount of recovery that acts against the reliability of the programmed key. This finding implies that it is possible to store keys in thresholds for a long duration provided that SRAM cells are powered up only when key generation is required and are powered off at the rest of the time. We use this point in Section 6.5 when designing the hardware prototype.

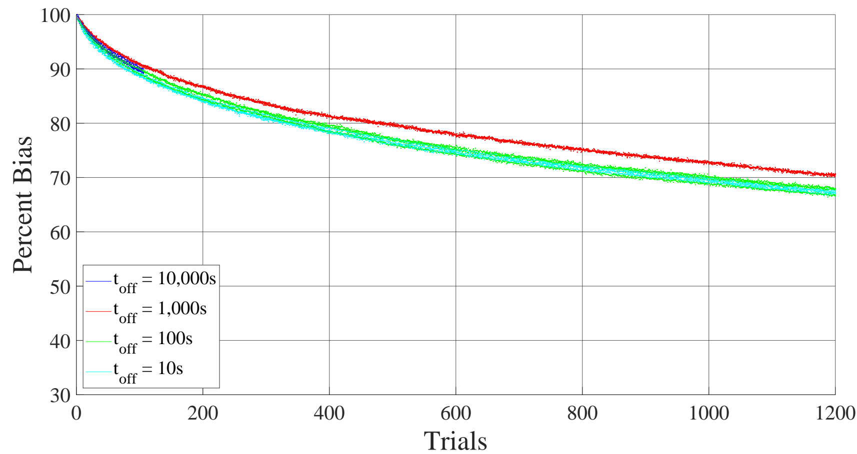
$$\text{Percent Bias}^t = \frac{|\sum_{j=1}^{C}(b_j^t - HW_j)|}{|\sum_{j=1}^{C}(b_j^0 - HW_j)|} \tag{6.1}$$

6.4 Characterizing the Key Generation Scheme

As explained in Section 6.2, the value that is generated at the SRAM power-up goes through an error-correcting phase, which consists of a repetition decoding block and a BCH decoder block to generate the key. In this section, we observe how key reliability changes with different amounts of repetition and the number of correctable errors in a BCH error correction block. We then use the two-parameter model to find the design parameters that meet the design's reliability criterion. The parameters we need to determine are size of the repetition and strength of the BCH decoder in terms of the maximum number of errors that can be corrected in each block. We measure the hardware costs for implementing the system under different design parameters in terms of area and perform simulation-based analysis to estimate the amount of



(a) Percent of Bias Remaining vs. Overall Time Elapsed for 23LC1024 SRAM



(b) Percent of Bias Remaining vs. Number of Power-on Trials for 23LC1024 SRAM

Figure 6.8. Plots show same data against different x axes. Recovery depends primarily on number of uses since the aging was applied, and not on the elapsed time.

induced bias in transistors. For the rest of this chapter, we put our focus on a single model of SRAM chip (23LC1024) and perform our analysis based on the data collected from this type. A similar analysis should be performed on any SRAM model that would be used in our system, regardless of whether it is a standalone chip or SRAM block on the Integrated key generation circuit.

6.4.1 Repetition Decoder

Based on the observation that the expected Hamming weight of each SRAM cell depends on its address (see Fig. 6.4), the effect of directed aging can be more accurately detected when taking into account the expected Hamming weight of the same cells before they received aging. Therefore when generating the key values, we put some weight into the natural power-up tendancy of SRAM cells. We take into account the Hamming weight of the unbiased SRAM cell when decoding the repetition block. Considering a repetition block of M bits, the expected Hamming weight of block i (HW_i) can be calculated as shown in Eq. 6.2. In this equation, $hw_{i,j}$ is the expected Hamming weight of $j-th$ bit in block i.

$$HW_i = hw_{i,1} + hw_{i,2} + ... + hw_{i,M} \tag{6.2}$$

We deduce the power-up value from repetition block i, corresponding to the i_{th} bit of secret data (b_i), based on Eq. 6.3. In this equation, $x_{i,j}$ denotes the after-aging power-up value of the SRAM bit at position j in block i. The summation of post-aging bit values in repetition block is compared to HW_i, which is the expected Hamming weight of block i calculated with Eq. 6.2, to find b_i.

$$b_i = \begin{cases} 0\ ; & if\ (\sum_{j=1}^{M} x_{i,j}) \leq HW_i \\ 1\ ; & otherwise \end{cases} \tag{6.3}$$

6.4.2 Key Reliability of Empirical Data

We first evaluate the key reliability from the empirical data, that is, the data from standalone SRAM chips collected using the microcontroller. The average key reliability is defined as the ratio of the correctly generated keys to the total number of key generation trials. We chose different 128-bit keys to be stored in each SRAM chip. To investigate the effect of repetition size on the key reliability, each key passes

through a fixed BCH decoder, which is of type `BCH[255,215,5]` for this experiment, but a varied repetition size. The size of the key is 215 bits, from which 128 bits are used. The total number of SRAM cells required to store each encoded key is therefore $M \times 255$, where M is the repetition size and 255 is the size of BCH encoder's output. We have chosen M to be the upper bound of all repetition values that we wanted to experiment, which enabled us to select different sized of repetition for the later experiments by using subsets of the data. A total of 8 different encoded keys are written in the same way on 9 SRAM chips, to be sure of including the effect of process variation in our experiments. The SRAM cells are biased to write the desired values as explained in Section 6.2.2. Fig. 6.9 shows the key reliability versus the size of the repetition block. Each line in this plot corresponds to the average key reliability over 200 key generation trials at different stages of chips' lifetime: right after the aging and zero key generations, and after 200, 400, 600, and 800 key generations. The power-up trials were conducted in intervals of 10 seconds ($t_{off} = 10s$). The figure shows that the key reliability degrades with the total number of SRAM power-ups and implicates the necessity to account for recovery and the expected lifetime of the device when choosing error correction parameters. The figure also shows how increasing the repetition size can improve the reliability and counteract the recovery effect.

6.4.3 Key Reliability Model

As previously mentioned in Chapter 5, the infrequent occurrence of bit errors can cause a challenge in calculating the key reliability. This challenge can be addressed by using an accurate model, which provides us with bit error rates for virtually an infinite number of cells to perform our key reliability calculations. In this subsection, we perform a model-based key reliability evaluation using the experimental data from SRAM cells. We used the same data as of Section 6.4.2, this time only considering the values between the 900th and 1000th power-ups to allow a certain amount of

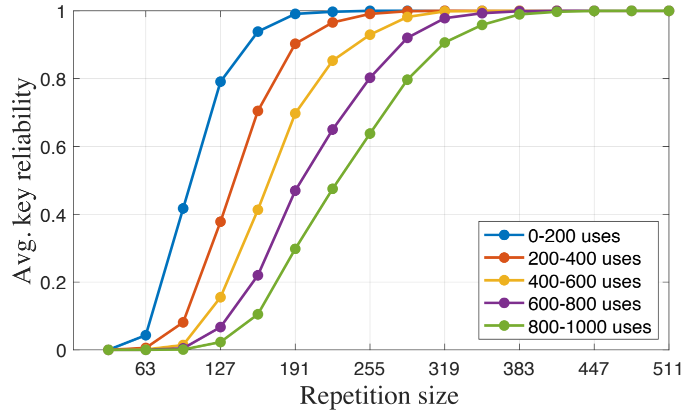


Figure 6.9. Key reliability as a function of cell repetition size for `BCH[255,215,5]`

recovery to occur in the cells.The SRAM cells that store redundant copies of each bit are from consecutive addresses, and therefore, likely adjacent in physical layout. Based on our observations from Section. 6.3.1 and Fig. 6.5, there are two groups of cells in the SRAM chips in terms of their inherent before-aging power-up Hamming weights. We refer to these as high and low Hamming weight groups. Considering only the cells that are aged to store a 1 for each group, parameters λ_1 and λ_2 are chosen by fitting Eq. 6.4 to the empirical CDF of cell 1-probabilities from experiments using Levenberg-Marquardt algorithm. The error probability is equal to the probability that the cell is powered-up to "0" (P_0), as we only consider the cells that are aged towards "1". Fig. 6.10 shows the cumulative distribution function (CDF) of bit error probability for each of the high and low Hamming weight groups, as well as their relative fitted model.

$$\mathbf{cdf}_{P1}(x) = \phi(\lambda_1 \phi^{-1}(x) + \lambda_2) \tag{6.4}$$

Assuming the expected Hamming weight of the SRAM cell in repetition block i is HW_i and that the cells are aged to produce value "1", the block produces an erroneous result (value "0") when the total post-aging Hamming weight of the power-up values

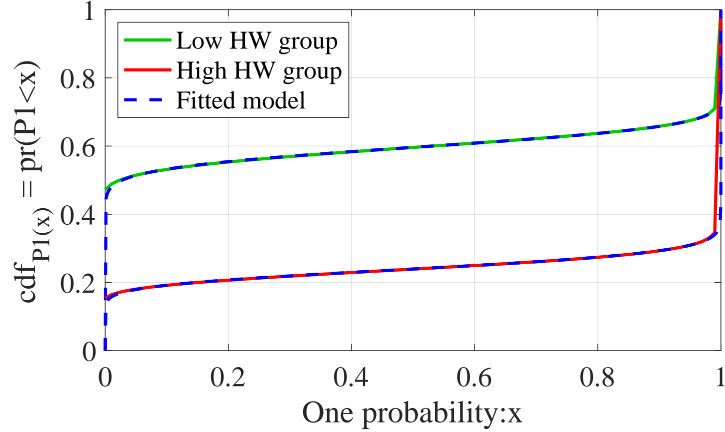


Figure 6.10. Cumulative distribution function of error probabilities from the experimental data and their relative fitted curves for the two SRAM cell groups with high and low Hamming weights.

in the block is less than HW_i (Eq. 6.3). If $P_{1,i}^M = (P_{1,i,1}, P_{1,i,2}, ..., P_{1,i,M})$ denote the list of 1-probabilities from M SRAM cells in repetition block i, the Poisson-binomial distribution $F_{PB}(HW_i; P_{1,i}^M)$ calculates the probability of having fewer than HW_i bits powering up to value "1", or the probability that the power-up value from repetition code is "0" according to Eq. 6.3. The value $F_{PB}(HW_i; P_{1,i}^M)$ can be calculated based on Eq. 5.3. The error rate $P_{e,i}$ of M-bit repetition block i can therefore be calculated according to Eq. 6.5.

$$P_{e,i} = P_{0,i} = F_{PB}(HW_i; P_{1,i}^M) \tag{6.5}$$

We calculated the error probabilities after repetition decoding by randomly choosing from the two groups (high and low Hamming weight cells) and using inverse transform sampling to pick P_1 of individual cells for each distribution. The P_1 values were then used in Eq. 6.5 to find the error rates after repetition decoding. Having the error rate of bits generated from the repetition block, the BCH block failure rate and the key

failure rate are calculated based on Eq. 5.4 and 5.2, respectively. We have repeated this procedure 100 times to measure the reliability of 100 different keys.

Fig. 6.11 shows the average key failure rate with respect to the repetition size M, for different BCH decoders. As expected, increasing the repetition size usually results in a decrease in the average key failure rate. Additionally, using a stronger BCH code that can correct a large number of errors also increases the reliability. For example, `BCH[255,131,18]`, which is the strongest error correction code shown in this figure with the capability to correct 18 error bits in a block of 255 bits, has the smallest failure rates for the same repetition size compared to the other BCH decoders.

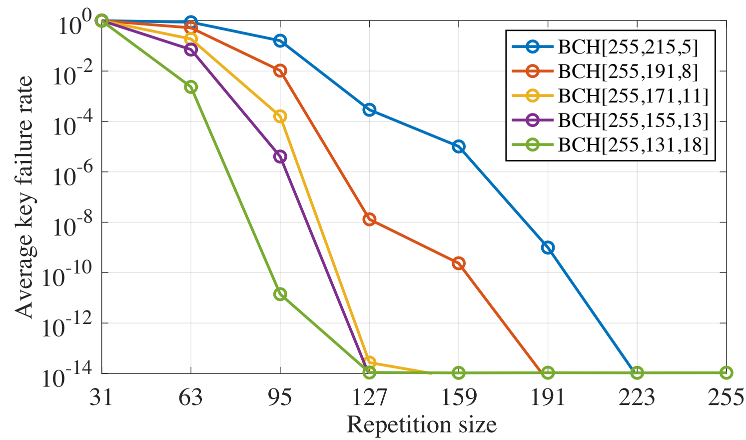


Figure 6.11. Average failure rate of key generation based on model from experimental data, with different amount of repetition. Each line represents one particular BCH code. Increasing the repetition size and using a stronger BCH code can both reduce the key failure rate.

6.4.4 Hardware Costs

As explained in previous sections, the key reliability can be adjusted by changing the repetition size and/or the strength of the BCH error correction block. Different combinations of these two can be used to meet a particular reliability criterion, and the various overheads should be considered while making these choices.

The reliability criterion that we consider is that 99% of chips should have failure rates of less than 10^{-6} after 900 trials of active recovery with $t_{on} = 30s$ and $t_{off} = 10s$. We anticipate that a much higher number of uses is possible, but chose 900 power-ups since that is what our model is fit to. To find the combinations of repetition and BCH decoder that meet the desired reliability criterion, we measure the key failure rate for different combinations of repetition sizes and BCH strengths based on the procedure explained in Section. 6.4.3. For each amount of repetition, we choose the weakest (and therefore lowest overhead) BCH code that will satisfy the reliability criterion mentioned earlier.

The BCH and repetition decoders are synthesized with Design Compiler using Nangate 45nm open cell library. To evaluate the area cost of SRAM, we use a value of $0.346\mu m^2$ as reported in [83] for the area of a single SRAM cell in 45nm technology. Table. 6.1 reports different pairs of repetition and BCH decoder parameters that meet our reliability criterion, as well as their area cost. It can be seen that amongst all choices of BCH and repetition, using a combination of 127-bits repetition and `BCH[255,187,9]` provides the smallest area overhead (as highlighted in green). Note that these numbers may vary for different technologies, and hence the optimal combination might differ, but the same methodology can be used to identify it.

Table 6.1. Combinations of repetition and BCH error correction that meet the reliability criterion of 99% chips having less than 10^{-6} error probabilities. The highlighted row corresponds to the lowest area cost. The reported area is in the units of μm^2.

Rep.	BCH code	#cells	Cells area	Rep. area	BCH area	Total area
31	[255,9,63]	118,575	41,027	27,964	48,574	187,566
63	[255,79,27]	32,130	11,117	5,205	27,714	44,036
95	[255,131,18]	24,225	8,382	3,918	21,658	33,958
127	[255,187,9]	32,385	11,205	4,460	17,120	32,785
159	[255,207,6]	40,545	14,028	4,950	15,485	34,463
191	[255,215,5]	48,705	16,852	5,441	15,067	37,360
255	[255,223,4]	65,025	22,499	6,531	14,359	43,389

6.4.5 Amount of Induced Bias in Cells

Since the details of the technology used in off-the-shelf ICs are usually unknown, there is no evidence on the value of threshold voltage of transistors in our experiments, nor on how much of threshold voltage offset the aging induces in cells. In this subsection, we try to correlate our experimental results to simulation data to estimate the amount of threshold offset that our aging experiments induce in cells.

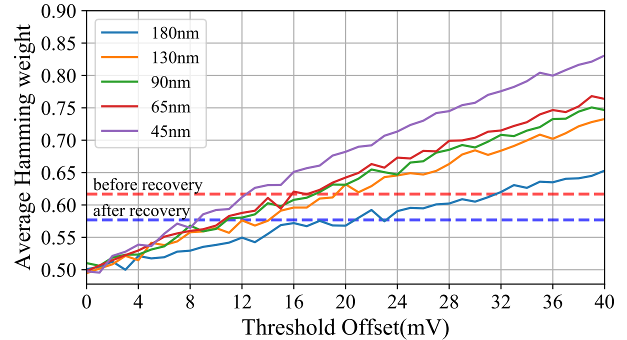
We consider two cases for the experimental data: Right after the accelerated aging and before any recovery happens on the chip and after 1000 power-ups of cells with the active recovery of $t_{on} = 30$s. For each of these cases, we calculate the average Hamming weight of all cells in the SRAM chip: 0.6168 before recovery and 0.5769 after the recovery.

We simulated a 6-T SRAM cell for a number of threshold voltage offsets (that cause it to favor the "1" state) in the presence of process variation. Since we do not know the technology that is used in our SRAM chips, we simulate different technology nodes: 45nm, 65nm, 90nm, and 180nm. The simulations are done in HSPICE using Predictive Technology Model (PTM) [11]. Threshold voltage offsets are varied in steps of 1mV, each considering 10,000 Monte-Carlo instances using the values reported in [87] for process variation in each technology. For each case, we evaluate the average Hamming weight and then compare these results to the average Hamming weight of the experimental data from before and after recovery to see how much threshold voltage change would cause the observed amount of bias from the experiments. The average Hamming weight for different values of threshold offsets for each technology is shown in the plot of Tab. 6.2. The Hamming weight from the SRAM aging experiments is shown as dashed horizontal lines for "before recovery" and "after recovery" cases. The intersection of these lines with the plot of each technology node specifies the threshold voltage offset that produces the same Hamming weight as the experimental data. These values are reported in Table. 6.2. Each row of the table estimates how much

the threshold voltage shift would have been if the SRAM chips from our experiments were built with the corresponding technology.

Table 6.2. Threshold offsets of different technology nodes from SPICE simulation that can explain the bias observed in power-up state of the experimental data. Data in table is derived from the plot at right, which shows the average Hamming weight corresponding to different threshold voltages at each technology nodes.

Technology	Before recovery(mV)	After recovery(mV)
180nm	31.7	23.1
130nm	19.3	14.1
90nm	17.7	10.9
65nm	17.1	10.6
45nm	12.4	8.6



6.5 Hardware Prototype Implementation

The key reliability evaluations in the previous sections were all performed using post-processing of SRAM data read out by an Arduino. In this section, we design and implement a hardware prototype of the full key generation system, using a combination of FPGA and SRAM chip. Our IP module prototype can be used to implement the aging-based keys on an ASIC. The encoded keys are programmed and written on this module by user or IP integrator through a USB interface from a personal computer, and the key is generated upon request as an output of this module. The SRAM holds the value of secret data in its biased transistors from the proposed aging-based approach discussed in the previous subsections. The FPGA interacts with the SRAM to perform read and write procedures, as well as performing error correction on the SRAM data to generate the key. Notably, the FPGA also controls power gating to the SRAM chip so that the chip is only powered on for the short duration of generating

the key. Employing power-gating is important because it minimizes the recovery. We will explore the hardware design in further detail in the following subsections.

6.5.1 Hardware Design

Fig. 6.12 shows the sketch of key generation hardware. For writing the key, the user sends the encoded key from a personal computer to the FPGA to handle writing it on the SRAM. The encoded key is replicated to the amount of repetition size and is written on SRAM by the read/write module through an SPI protocol that will be explained in Section. 6.5.1.1. Note that writing the key is done from a personal computer to avoid allocating any storage of secret data in hardware that can be vulnerable to reverse engineering. The module is then put into the thermal chamber to receive directed accelerated aging. After enough time for the SRAM chip to age, the hardware is removed from the thermal chamber, and the key values are ready to be generated from the biased power-up sate of SRAM cells, without any external interference from a user. The read/write module powers up the SRAM and reads out its secret value that is induced via aging. The read data then passes through the repetition decoding hardware that will be discussed in Section 6.4.1. To avoid allocating extra storage and save area in FPGA, each repetition block i is decoded serially after enough bits (M bits in an M-bit repetition block) are read from SRAM, before moving on to reading the next block. The output of the repetition decoder hardware then goes to the BCH decoder, which generates the final key value on its output. The generated secret key can then be used for arbitrary purposes but should never be revealed in clear.

All modules that are shown in Fig. 6.12 as part of the AXI4 peripheral IP block within the FPGA are written in Verilog from scratch, except for the BCH decoder that is from [33]. The IP was then used in a Vivado block design, in addition to Microblaze processor, which is part of a CMOD A7 FPGA. The Microblaze processor

controls and monitors reading and writing to some of our AXI4 IP signals through Xilinx Software Development Kit (SDK) environment. For example, the encoded key that is provided by the user as shown in Fig. 6.12 is done in the SDK environment. Additionally, we read out the keys generated on the output of the module from the SDK environment to verify the key reliability.

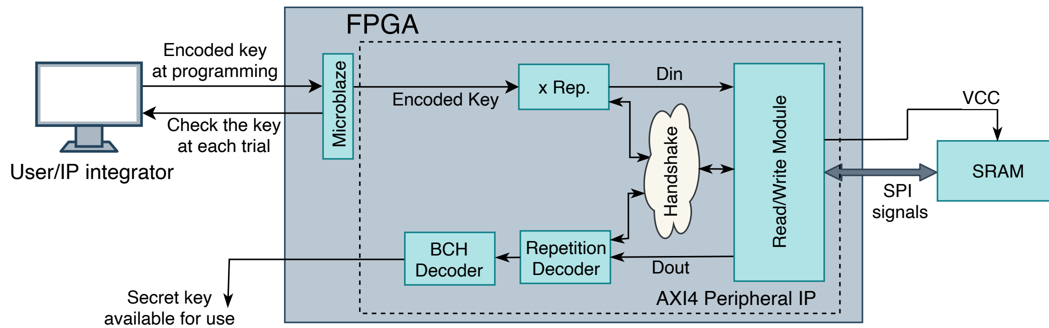


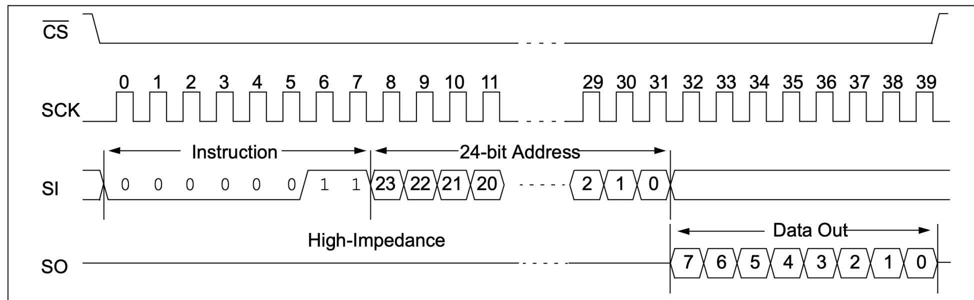
Figure 6.12. Sketch of the key storage hardware module

6.5.1.1 Read/Write Module

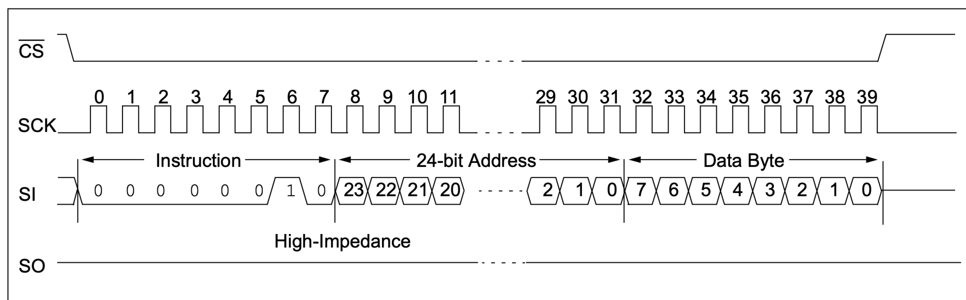
The memory chip is controlled by a read/write module that we designed in Verilog, which is based on the Serial Peripheral Interface (SPI) protocol. The SPI protocol used in the 23LC1024 SRAM chips contains four essential bus signals: a clock input `SCK`, a serial input data `SI`, a serial output data `SO`, and an active-low chip select $\overline{\texttt{CS}}$. Fig. 6.13a and 6.13b depict the read and write sequences of SPI protocol, respectively. The SRAM chip is selected for read/write operations by pulling $\overline{\texttt{CS}}$ low and providing a clock on `SCK`.

The 8-bit instruction is shifted through `SI` input, which is `00000011` for the read and `00000010` for the write operation. The 24-bit byte-address is shifted in after the instruction, with the seven MSB bits of the address being ”don’t care” bits. After the

address, the 8-bit data value is applied on `SI` for a write operation, or collected from the shifted out data from `SO` for a read operation.



(a) Read sequence



(b) Write sequence

Figure 6.13. Read and write operations in SPI protocol [3]

In addition to the signals that implement the SPI protocol, the FPGA also provides the supply voltage `VCC` to the SRAM such that the power is applied when reading and cut-off when reading from SRAM is complete. This power-gating approach alleviates the recovery effect by reducing the amount of time that the SRAM chip is powered-up when it is not used. In our experiments on the hardware prototype, the SRAM chips are only needed to be powered on for 200ms for a read operation, which is much shorter compared to t_{on} duration of the 30s from the previous sections. Considering the previously observed phenomenon that the recovery is dependent on the power-on time of the SRAM, we expect that our power-gating approach in the hardware prototype can improve the key reliability.

Aside from the ports that are used to communicate with the SRAM chip, the read/write protocol has other ports for communicating with the modules at the FPGA

side. These ports include 8-bit Din and Dout ports, a read/write input port that specifies the type of operation, and start and done ports that are used for handshaking with the other modules inside FPGA. Fig. 6.14 depicts the interfacing of read/write module to the SRAM and the other modules inside FPGA.

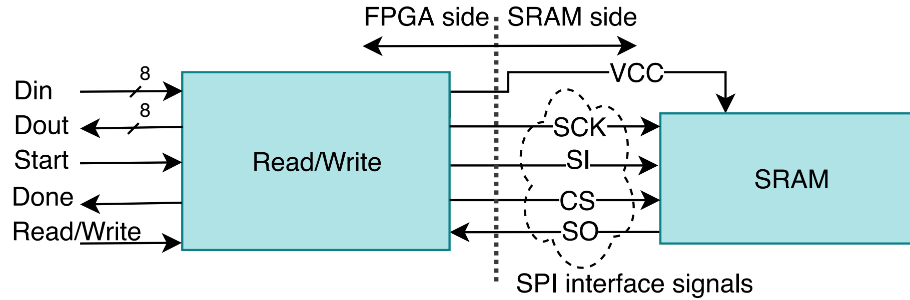


Figure 6.14. Interfacing of the read/write module to the FPGA side and SRAM side

6.5.1.2 Repetition Decoder Hardware

Fig. 6.15 shows the RTL schematic of the repetition decoder. The SRAM block provides the input to the repetition decoder, and contains the programmed cells that are aged towards holding the pre-defined values. Decoding is done serially, where fetching block i from the SRAM and calculating its corresponding output value is done before moving on to the block $i+1$. Each block i has an expected pre-aging Hamming weight HW_i that are constants derived from Eq. 6.2 and feed the input of a multiplexer with i as the select bit. `Shift reg1` and `Counter2` calculate the post-aging Hamming weight of block i by summing up bits of the block i read from SRAM, which is then compared to the output of Multiplexer (HW_i) to deduce the output value b_i (Eq. 6.3). The output value is saved in `Shift reg2`, and then `Incremental Addr.` issues the address corresponding to block $i+1$ to be fetched from SRAM.

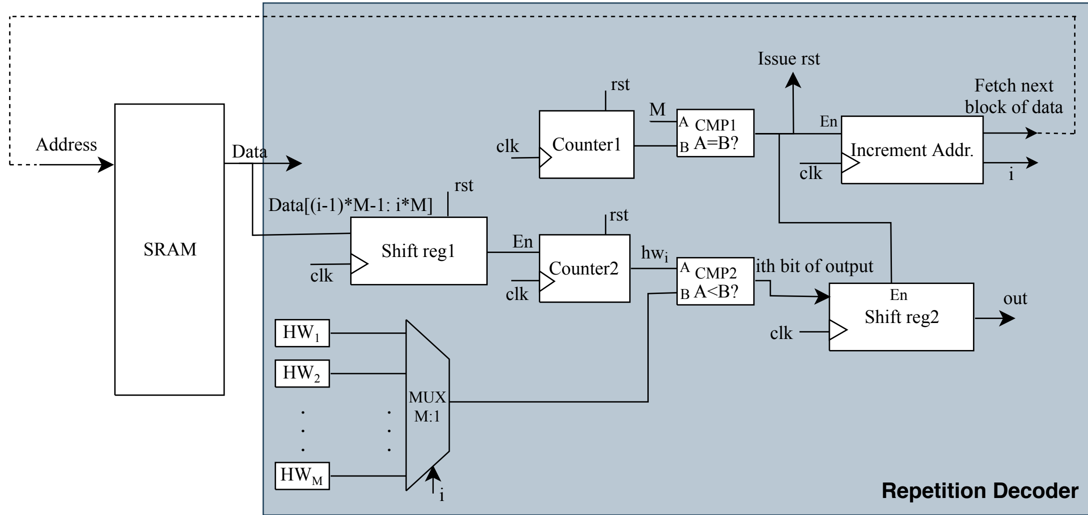


Figure 6.15. RTL schematic of the repetition decoder block from Fig 6.12

6.5.2 Hardware Implementation

We first verified the correct functionality of the design by implementing a working prototype on a breadboard. We used Cmod A7-35T FPGA from Digilent and 23LC1024 SRAM from Microchip. CMOD A7 has 28 I/O ports, from which we used five for the interfacing signals with the SRAM. The maximum clock frequency of this SRAM chip is 20MHz [3], and the clock frequency we provided was 10MHz. The FPGA provides the supply voltage to the SRAM directly from a 3.3V I/O of Cmod A7. The chip's operating voltage during accelerated aging was the same 3.3V. In our previous experiments, the chips received a supply voltage of 5V for one hour at 85°C. In this section, we allow a longer time to compensate for the lower supply voltage. During aging, the hardware remained in the thermal chamber for 3 hours at 85°C. The chip is then set to cool off too room temperature for about 20 minutes. The repetition value M is 255 bits, meaning that each encoded key bit value is stored on 255 individual SRAM cells. The BCH code was `BCH[255,131,18]`, meaning that it could cover up to 18 bits of errors in a block of 255 bits. Note that the error correction is different than the one reported in table 6.1 because the aging conditions have changed, which in

turn changes the amount of induced bias in SRAM cells. The design was synthesized and implemented in Vivado, as shown in Fig. 6.16. A total of 4699 slices are used on the FPGA, which constitutes about 23% of its total available resources.

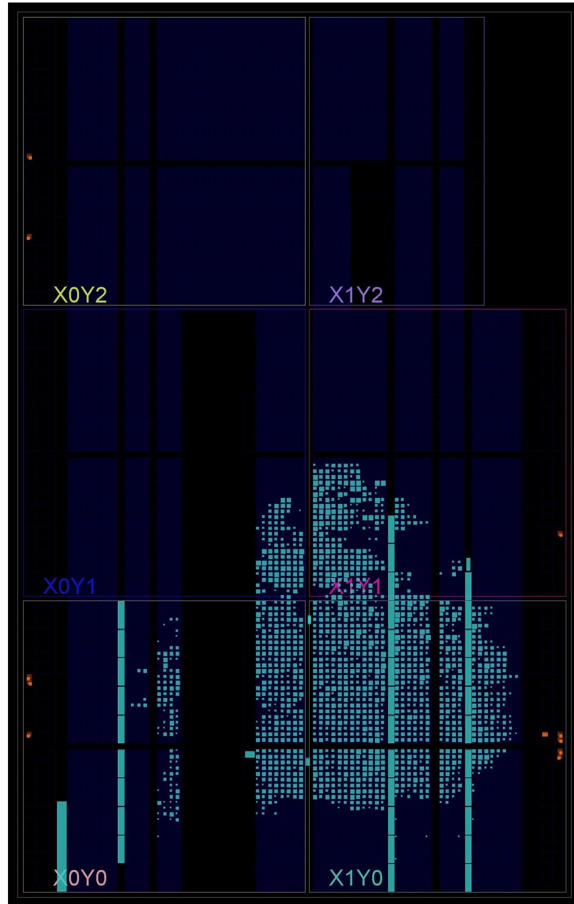


Figure 6.16. Device view from the implementation of the hardware design on the FPGA in Vivado

The design prototype is implemented and tested first on a breadboard and then on a PCB. Fig. 6.17 shows the prototype hardware implemented on a breadboard and Fig. 6.18 shows a picture of the fabricated prototype on PCB. The prototypes were tested for both read and write operations by writing the encoded key values in the SRAM and inducing accelerated aging in the thermal chamber, and reading SRAM values and generating keys multiple times after removing the chip from the thermal chamber. The hardware module was scheduled to generate the key in intervals of 10

seconds for one week, totaling to 60,000 key generations. We observed that the five out of five tested modules generated the correct key in 100% of the trials.

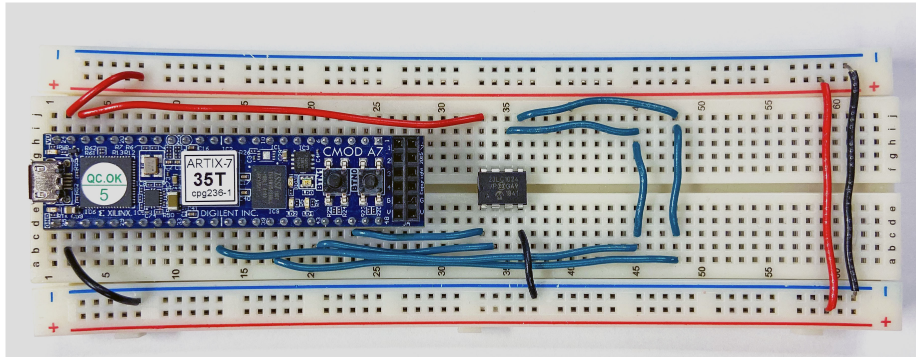


Figure 6.17. Key storage hardware prototype on breadboard

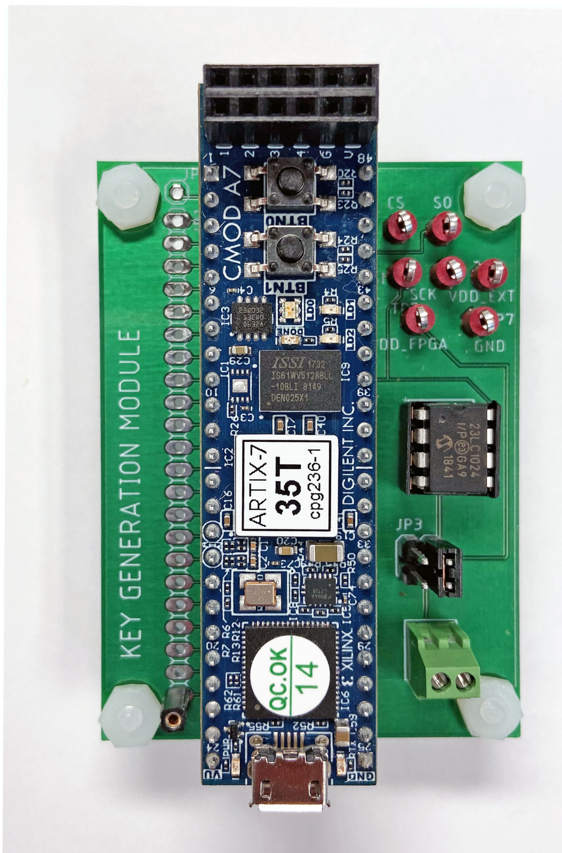


Figure 6.18. Key storage hardware prototype on printed circuit board

6.6 Resistance against Invasive Readout

In this section, we will evaluate our approach in terms of its resistance against reverse engineering. We use the same model as in Chapter. 5 here as well. Recall that the heterogeneous error rate model assumes two sources of variation in a cell. The first source of variation is the process variable (M) that accounts for the persistent impact of bias and process variations, and is modeled with a normal distribution of $N(\mu_M, \sigma_M)$. The second source of variation is the noise variable (N_i) that accounts for the cumulative effect of all noise sources during evaluation, and is modeled with a normal distribution of $N(0, \sigma_{N_i})$. The three unknowns in these distributions are reduced to two parameters λ_1 and λ_2, wthere $\lambda_1 = \sigma_{N_i}/\sigma_M$ and $\lambda_2 = (t - \mu_M)/\sigma_M$. The model parameters already account for the effects of manufacturing variations and noise. We model the measurement error as an increase of the noise parameter (N_i) for the attacker. The value of λ_1 corresponds to the ratio of noise variation to the process variation. Hence, an increase in λ_1 would indicate an increase in σ_{N_i} or the amount of noise. The P_1 of SRAM cells from the attacker's measurements can then be estimated by increasing λ_1 after fitting the model while keeping λ_2 constant. The attacker's ability to read out the key can then be calculated using the same model that calculates the key failure rate, and can be found using Eq. 6.5, 5.4 and 5.2.

For our results on the attacker's key readout, we have assumed that $\lambda_1 = \sigma_{N_i}/\sigma_M = 1$. In other words, the total cumulative noise from both environmental noises and attacker's measurement inaccuracies are considered to be equal to the amount of process variation ($\sigma_{N_i} = \sigma_M$). This ratio can be increased or decreased to reflect different attacker read accuracies. The parameters are fitted to the experimental data as in Section. 6.4.3, and with separate fitting for high and low Hamming weight regions. The value of λ_1 that we considered for modeling the attacker's key readout is about eight times as the original value of λ_1 for both high and low Hamming weight groups. Fig. 6.19 shows the average bit error rate with different amounts of repetition

for three cases: (1) The average bit error rates that are directly measured from the experimental data, (2) The average bit error rates that are calculated from the model, where σ_{N_i} is set to 0 after fitting the model to the experimental data, (3) The average bit error rates that are calculated from the model, where σ_N is set to be equal to σ_M after fitting the data to the model, to reflect the attacker's measurement inaccuracies. Note that the average bit error rate from the experimental data and the model with zero environmental noise more closely correlate, as compared to the case where the average bit error rate is increased due to the increased effect of noise that models the attacker's read inaccuracies.

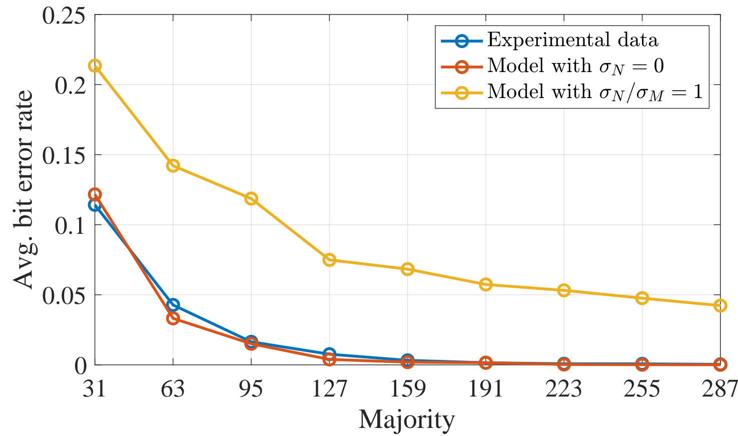


Figure 6.19. Bit error rate with only repetition and no BCH

As a comparison for the attacker's success versus the key reliability, we have depicted the key reliability versus the attacker's read failure rate, as shown in Fig. 6.20, for different combinations of repetition size and strengths of BCH error correction. The key failure rate decreases with going down at the y-axis, and the attacker's success rate drops with going left at the x-axis. Hence, the scheme provides more reliability if its corresponding point in the plot is more down and is more secure if it is more to the left compared to the other points. Based on the design requirements, the designer

can choose between these different trade-offs to provide acceptable key reliability and security against reverse engineering.

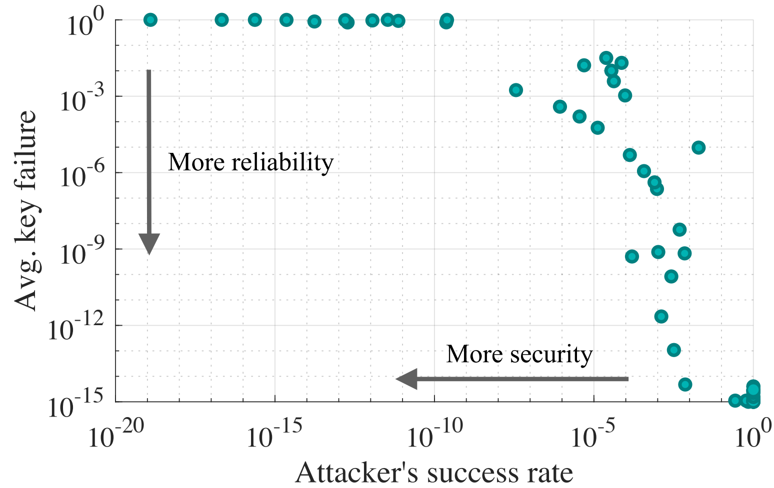


Figure 6.20. Attacker's read success rate versus key reliability for a combination of repetition and BCH codes

6.7 Conclusions

We have proposed and demonstrated a hardware key generation design that resists invasive readout, and is secure against an untrusted foundry. Our scheme incorporates NBTI-based accelerated aging in conventional 6-T SRAM cells to store secret keys. While the foundry is only responsible for fabricating the necessary infrastructure for storing the secret key, programming the keys is done after chip fabrication by a user or an IP integrator. We have proposed using an optimal combination of repetition and BCH decoder for error correction to ensure the key reliability of our approach while minimizing cost. Additionally, our approach uses a new repetition decoder that accommodates the non-uniform pre-aging patterns that we observed in SRAM cells. The amount of threshold voltage shift from the accelerated aging method is estimated by correlating our experimental results to the simulation data from different technology

nodes. We evaluated the key reliability of our approach, using both experimental evaluations and model-based analysis. Moreover, we have designed and fabricated a working prototype of the key generation hardware module on PCBs and verified the correct functionality and reliability of its key generation over time. Finally, a model-based security analysis method against reverse engineering is proposed to quantify the likelihood of successful readout for an attacker.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Summary and Conclusions

From their use in smartphones and smartwatches to their applications in healthcare and aviation, semiconductors have become a fundamental part of our daily lives. The growing application of semiconductors raises the need for addressing their security and privacy challenges as well. If hardware security is not well taken care of, it can provide an easy backdoor for a reverse engineer that tries to steal secret information stored in the hardware or steal and counterfeit intellectual property. This thesis has tried to address some of the security concerns of today's hardware systems.

First, we have investigated the privacy implications of voltage-overscaled or frequency-overscaled approximate computations. By performing a simulation-based analysis of different adder styles, we have shown that the identity of approximate computing devices can be revealed by providing certain inputs to them and observing their corresponding outputs.

Next, we have addressed the problem of protecting the chip against reverse engineering by improving the state-of-art gate obfuscation methods. We assumed the scenario of a more knowledgeable attacker that has partial knowledge about the functions that are viable in an obfuscated hardware design, and we have proposed an automation technique for designing circuits that can plausibly implement a number of chosen functions. The procedure comprises iterated synthesis and uses genetic programming to find an optimized pin assignment that maximizes shared logic between the functions.

Additionally, we have introduced a new SAT-based invasive reverse engineering technique that requires no knowledge about the gate functions or the connections between them. The new SAT formulation incorporates additional data from voltage probing and fault injections, and includes new levelization constraints to enforce acyclic topology and avoid encountering loops while solving the SAT formulation. Unlike existing SAT attacks, our method can recover the exact gate-by-gate netlist of the obfuscated circuit.

Finally, we have investigated the problem of secret key storage in hardware by first introducing a key generation approach that provides quantifiable security against invasive reverse engineering. The key generation scheme is based on offsetting the threshold voltage of transistors in a conventional 6-T SRAM cell and requires foundry support. We performed a model-based approach to evaluate the key reliability of our proposed scheme and its resistance to invasive readout. We then demonstrated a key generation approach that uses NBTI-based directed accelerated aging to program the key without the support of a trusted foundry. In this new scheme, the foundry fabricates the circuitry of the key storage module without knowing the key that the user/IP integrator will program afterward. We have built and validated a hardware prototype of this key storage approach on PCB using FPGA logic and standalone SRAM chips.

The following peer-reviewed articles are published as part of the research done towards this degree [32, 61–65, 108, 115].

7.2 Suggestions of Future Research

This dissertation has addressed some of the issues in protecting hardware designs against reverse engineering. However, there are still open challenges on the way to reaching a secure hardware design. Possible extensions from this thesis research are summarized as follows.

The first challenge is the security evaluation of cyclic obfuscation. Cyclic obfuscation is an approach in which combinational cycles are put intentionally in the design under the intent that a potential cycle in the circuit can put the SAT solver in an infinite loop when trying to deobfuscate the circuit [97]. In this dissertation, the SAT-based decamouflgaging method for backside reverse engineering restrict solution space to acyclic circuits using levelization constraints. Future work can investigate the possibility of breaking cyclic obfuscation approaches with these levelization formulation.

In the area of secret key generation, we have proposed using accelerated NBTI-based aging to induce bias in the threshold voltage of transistors to store the secret data. One of the advantages of NBTI-based accelerated aging is that it does not require an additional circuit to induce the aging effect. However, as proposed in [18], the baking time of the NBTI can be relatively high while the amount of induced threshold shift in transistors is low. While the HCI-based accelerated aging requires additional aging circuitry, the stress time is in the order of seconds and can create voltage shifts of greater than 100mV [18]. Future work can investigate the use of HCI-based accelerated aging for the proposed key storage: analyzing the amount of recovery from HCI-based accelerated aging over time, measuring the key reliability, and quantifying the attacker's success rate with this approach.

Finally, there is a problem of protecting a design against an untrusted foundry that tries to insert malicious hardware, which can enable and facilitate certain kinds of attacks. For example, it has been shown that setting the LSB of the 14th round of DES implementation to zero helps with recovering the secret key in only two messages [22]. An untrusted foundry can therefore insert hard-to-detect malicious hardware to control these data after fabrication and steal the secret information. One of the state-of-art approaches for addressing the issue of an untrusted foundry is split-manufacturing, which hides the function from the foundry by withholding upper

metal connections. Split manufacturing, however, comes with the additional overhead of around 1.6x the power consumption, 1.8x delay, and about 3x the area [52]. As a possible research direction, a new approach can be investigated to provide security against untrusted foundry in terms of state-of-art security metrics [52], while enabling the IP integrator/users to program the design and allow the correct functionality after the fabrication. Therefore, the proposed scheme should encompass hardware that enables design configurability with minimum hardware overhead, while still providing sufficient security, i.e. appearing ambiguous enough to the untrusted foundry.

BIBLIOGRAPHY

[1] https://www.semiconductors.org/annual-semiconductor-sales-increase-21.6-percent-top-400-billion-for-first-time/.

[2] http://www.crypto-it.net/eng/theory/kerckhoffs.html.

[3] 23LC1024 Data Sheet. http://ww1.microchip.com/downloads/en/DeviceDoc/20005142C.pdf.

[4] ABC: A System for Sequential Synthesis and Verification. https://people.eecs.berkeley.edu/~alanmi/abc/.

[5] Chipworks: circuit extraction software. http://www.chipworks.com/ko/node/298.

[6] Degate. http://www.degate.org/documentation/.

[7] Distributed Evolutionary Algorithms in Python. http://deap.readthedocs.io/en/master/.

[8] N01S830HA Data Sheet. https://www.onsemi.com/pub/Collateral/N01S830HA-D.PDF.

[9] TestEquity Model 115A Temperature Chamber . http://chamber.testequity.com/115.html.

[10] Yosys Open Synthesis Suite. http://www.clifford.at/yosys/.

[11] Predictive Technology Model 45nm CMOS, 2006. http://ptm.asu.edu/.

[12] NanGate 45nm Open Cell Library, 2010. http://www.nangate.com.

[13] Agarwal, Mridul, Paul, Bipul C, Zhang, Ming, and Mitra, Subhasish. Circuit failure prediction and its application to transistor aging. In *VLSI Test Symposium, 2007. 25th IEEE* (2007), IEEE, pp. 277–286.

[14] Anderson, Jason H. A PUF design for secure FPGA-based embedded systems. In *Proceedings of the 2010 Asia and South Pacific Design Automation Conference* (2010), pp. 1–6.

[15] Becker, Georg T. Robust fuzzy extractors and helper data manipulation attacks revisited: theory vs practice. *IEEE Transactions on Dependable and Secure Computing* (2017).

[16] Becker, Georg T, Regazzoni, Francesco, Paar, Christof, and Burleson, Wayne P. Stealthy dopant-level hardware trojans. In *International Workshop on Cryptographic Hardware and Embedded Systems* (2013), Springer, pp. 197–214.

[17] Bhargava, Mudit, Cakir, Cagla, and Mai, Ken. Reliability enhancement of bi-stable pufs in 65nm bulk CMOS. In *2012 IEEE International Symposium on Hardware-Oriented Security and Trust* (2012), IEEE, pp. 25–30.

[18] Bhargava, Mudit, and Mai, Ken. A high reliability PUF using hot carrier injection based response reinforcement. In *International Workshop on Cryptographic Hardware and Embedded Systems* (2013), Springer, pp. 90–106.

[19] Bogdanov, Andrey, Knudsen, Lars R, Leander, Gregor, Paar, Christof, Poschmann, Axel, Robshaw, Matthew JB, Seurin, Yannick, and Vikkelsoe, Charlotte. Present: An ultra-lightweight block cipher. In *CHES* (2007), vol. 4727, Springer, pp. 450–466.

[20] Boit, C, Schlangen, R, Glowacki, A, Kindereit, U, Kiyan, T, Kerst, U, Lundquist, T, Kasapi, S, and Suzuki, H. Physical IC debug–backside approach and nanoscale challenge. *Advances in Radio Science 6*, D. 5 (2008), 265–272.

[21] Bojinov, Hristo, Michalevsky, Yan, Nakibly, Gabi, and Boneh, Dan. Mobile device identification via sensor fingerprinting. *arXiv preprint arXiv:1408.1416* (2014).

[22] Boneh, Dan, DeMillo, Richard A, and Lipton, Richard J. On the importance of checking cryptographic protocols for faults. In *International conference on the theory and applications of cryptographic techniques* (1997), Springer, pp. 37–51.

[23] Borkar, Shekhar, Karnik, Tanay, Narendra, Siva, Tschanz, Jim, Keshavarzi, Ali, and De, Vivek. Parameter variations and impact on circuits and microarchitecture. In *Proceedings of the 40th Annual Design Automation Conference* (2003), DAC '03, pp. 338–342.

[24] Bösch, Christoph, Guajardo, Jorge, Sadeghi, Ahmad-Reza, Shokrollahi, Jamshid, and Tuyls, Pim. Efficient helper data key extractor on fpgas. In *International Workshop on Cryptographic Hardware and Embedded Systems* (2008), Springer, pp. 181–197.

[25] Chen, Shuai, Chen, Junlin, Forte, Domenic, Di, Jia, Tehranipoor, Mark, and Wang, Lei. Chip-level anti-reverse engineering using transformable interconnects. In *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), 2015 IEEE International Symposium on* (2015), IEEE, pp. 109–114.

[26] Chow, Lap-Wai, Baukus, James P, and Clark Jr, William M. Integrated circuits protected against reverse engineering and method for fabricating the same using an apparent metal contact line terminating on field oxide, Nov. 13 2007. US Patent 7,294,935.

[27] Cocchi, Ronald P, Baukus, James P, Chow, Lap Wai, and Wang, Bryan J. Circuit camouflage integration for hardware IP protection. In *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE* (2014), IEEE, pp. 1–5.

[28] Collantes, Maria I Mera, El Massad, Mohamed, and Garg, Siddharth. Threshold-dependent camouflaged cells to secure circuits against reverse engineering attacks. In *VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on* (2016), IEEE, pp. 443–448.

[29] De Wolf, Peter, Geva, M, Hantschel, Thomas, Vandervorst, Wilfried, and Bylsma, RB. Two-dimensional carrier profiling of InP structures using scanning spreading resistance microscopy. *Applied physics letters 73*, 15 (1998), 2155–2157.

[30] Delvaux, Jeroen, Gu, Dawu, Schellekens, Dries, and Verbauwhede, Ingrid. Helper data algorithms for PUF-based key generation: Overview and analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 34*, 6 (2014), 889–902.

[31] Delvaux, Jeroen, and Verbauwhede, Ingrid. Key-recovery attacks on various RO PUF constructions via helper data manipulation. In *Proceedings of the conference on Design, Automation & Test in Europe* (2014), European Design and Automation Association, p. 72.

[32] Dhanuskodi, Siva Nishok, Keshavarz, Shahrzad, and Holcomb, Daniel. LLPA: logic state based leakage power analysis. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* (2016), IEEE, pp. 218–223.

[33] Dill, Russ. Verilog BCH encoder/decoder. `https://github.com/russdill/bch_verilog/commits/master`, 2015.

[34] Ender, Maik, Ghandali, Samaneh, Moradi, Amir, and Paar, Christof. The first thorough side-channel hardware trojan. In *International Conference on the Theory and Application of Cryptology and Information Security* (2017), Springer, pp. 755–780.

[35] Erbagci, B., Erbagci, C., Akkaya, N. E. C., and Mai, K. A secure camouflaged threshold voltage defined logic family. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (May 2016), pp. 229–235.

[36] Esmaeilzadeh, Hadi, Sampson, Adrian, Ceze, Luis, and Burger, Doug. Architecture support for disciplined approximate programming. In *ASPLOS'12: Architectural Support for Programming Languages and Operating Systems* (Apr. 2012).

[37] Fawcett, Tom. An introduction to ROC analysis. *Pattern Recognition Letters 27*, 8 (2006), 861 – 874. ROC Analysis in Pattern Recognition.

[38] Flamm, Kenneth. Measuring moore's law: Evidence from price, cost, and quality indexes. Tech. rep., National Bureau of Economic Research, 2018.

[39] Gajda, Zbysek, and Sekanina, Lukas. Gate-level optimization of polymorphic circuits using cartesian genetic programming. In *2009 IEEE Congress on Evolutionary Computation* (2009), IEEE, pp. 1599–1604.

[40] Gassend, B, Clarke, D, and Van Dijk, M. Silicon physical random functions. In *Proceedings of the IEEE Computer and Communications Society* (2002).

[41] George, J., Marr, B., Akgul, B. E. S., and Palem, K. V. Probabilistic arithmetic and energy efficient embedded signal processing. In *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems* (New York, NY, USA, 2006), CASES '06, ACM, pp. 158–168.

[42] Ghandali, Samaneh, Becker, Georg T, Holcomb, Daniel, and Paar, Christof. A design methodology for stealthy parametric trojans and its application to bug attacks. In *International Conference on Cryptographic Hardware and Embedded Systems* (2016), Springer, pp. 625–647.

[43] Ghosh, Swaroop, and Roy, Kaushik. Parameter variation tolerance and error resiliency: New design paradigm for the nanoscale era. *Proceedings of the IEEE 98* (11 2010), 1718 – 1751.

[44] Guajardo, J, Kumar, S, Schrijen, GJ, and Tuyls, P. FPGA intrinsic PUFs and their use for IP protection. *Cryptographic Hardware and Embedded Systems* (2007).

[45] Guo, X., Burleson, W., and Stan, M. Modeling and experimental demonstration of accelerated self-healing techniques. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)* (June 2014), pp. 1–6.

[46] Gupta, V., Mohapatra, D., Raghunathan, A., and Roy, K. Low-power digital signal processing using approximate adders. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 32*, 1 (Jan 2013), 124–137.

[47] Gupta, Vaibhav, Mohapatra, Debabrata, Park, Sang Phill, Raghunathan, Anand, and Roy, Kaushik. Impact: imprecise adders for low-power approximate computing. In *Proceedings of the 17th international symposium on Low-power electronics and design* (2011), pp. 409–414.

[48] Han, T., and Carlson, D. A. Fast area-efficient vlsi adders. In *Computer Arithmetic (ARITH), 1987 IEEE 8th Symposium on* (May 1987), pp. 49–56.

[49] Hegde, R, and Shanbhag, N R. Soft Digital Signal Processing. *IEEE Transaction on Very Large Scale Integration (VLSI) Systems* (2001).

[50] Herder, Charles, Yu, Meng-Day, Koushanfar, Farinaz, and Devadas, Srinivas. Physical unclonable functions and applications: A tutorial. *Proceedings of the IEEE 102*, 8 (2014), 1126–1141.

[51] Holcomb, D. E., Burleson, W. P., and Fu, K. Power-up SRAM state as an identifying fingerprint and source of true random numbers. *IEEE Transactions on Computers 58*, 9 (Sept 2009), 1198–1210.

[52] Imeson, Frank, Emtenan, Ariq, Garg, Siddharth, and Tripunitara, Mahesh. Securing computer hardware using 3d integrated circuit (IC) technology and split manufacturing for obfuscation. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security)* (2013), pp. 495–510.

[53] ISO. Information technology – security techniques – lightweight cryptography – part 2: Block ciphers. *ISO/IEC 29192-2:2012* (2012).

[54] Iyengar, Anirudh, and Ghosh, Swaroop. Threshold voltage-defined switches for programmable gates.

[55] Jain, Anil K, Prabhakar, Salil, and Chen, Shaoyun. Combining multiple matchers for a high security fingerprint verification system. *Pattern Recognition Letters 20*, 11-13 (1999), 1371 – 1379.

[56] Jha, Susmit, Gulwani, Sumit, Seshia, Sanjit A, and Tiwari, Ashish. Oracle-guided component-based program synthesis. In *Software Engineering, 2010 ACM/IEEE 32nd International Conference on* (2010), vol. 1, IEEE, pp. 215–224.

[57] Kahng, Andrew B, and Kang, Seokhyeong. Accuracy-configurable adder for approximate arithmetic designs. In *Proceedings of the 49th Annual Design Automation Conference* (2012), ACM, pp. 820–825.

[58] Karimi, Elmira, Fei, Yunsi, and Kaeli, David. Hardware/software obfuscation against timing side-channel attack on a GPU. In *Hardware Oriented Security and Trust (HOST), IEEE International Symposium on* (2020), IEEE.

[59] Karimi, Elmira, Jiang, Zhen Hang, Fei, Yunsi, and Kaeli, David. A timing side-channel attack on a mobile gpu. In *2018 IEEE 36th International Conference on Computer Design (ICCD)* (2018), IEEE, pp. 67–74.

[60] Kedem, Zvi M, Mooney, Vincent J, Muntimadugu, Kirthi Krishna, and Palem, Krishna V. An approach to energy-error tradeoffs in approximate ripple carry adders. In *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design* (2011), pp. 211–216.

[61] Keshavarz, Shahrzad, and Holcomb, Daniel. Privacy leakages in approximate adders. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)* (2017), IEEE, pp. 1–4.

[62] Keshavarz, Shahrzad, and Holcomb, Daniel. Threshold-based obfuscated keys with quantifiable security against invasive readout. In *Proceedings of the 36th International Conference on Computer-Aided Design* (2017), IEEE Press, pp. 57–64.

[63] Keshavarz, Shahrzad, Paar, Christof, and Holcomb, Daniel. Design automation for obfuscated circuits with multiple viable functions. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2017), IEEE, pp. 886–889.

[64] Keshavarz, Shahrzad, Schellenberg, Falk, Richter, Bastian, Paar, Christof, and Holcomb, Daniel. SAT-based reverse engineering of gate-level schematics using fault injection and probing. In *Hardware Oriented Security and Trust (HOST), 2018 IEEE International Symposium on* (2018), IEEE.

[65] Keshavarz, Shahrzad, Yu, Cunxi, Ghandali, Samaneh, Xu, Xiaolin, and Holcomb, Daniel. Survey on applications of formal methods in reverse engineering and intellectual property protection. *Journal of Hardware and Systems Security 2*, 3 (2018), 214–224.

[66] Keutzer, K. DAGON: technology binding and local optimization by DAG matching. In *Proceedings of the 24th ACM/IEEE Design Automation Conference* (1987), ACM, pp. 341–347.

[67] Kindereit, Ulrike. Fundamentals and future applications of laser voltage probing. In *Reliability Physics Symposium, 2014 IEEE International* (2014), IEEE, pp. 3F–1.

[68] Kopanski, JJ, Marchiando, JF, and Lowney, JR. Scanning capacitance microscopy measurements and modeling: Progress towards dopant profiling of silicon. *Journal of Vacuum Science & Technology B: Microelectronics and Nanometer Structures Processing, Measurement, and Phenomena 14*, 1 (1996), 242–247.

[69] Koren, E, Rosenwaks, Y, Allen, JE, Hemesath, ER, and Lauhon, LJ. Nonuniform doping distribution along silicon nanowires measured by kelvin probe force microscopy and scanning photocurrent microscopy. *Applied Physics Letters 95*, 9 (2009), 092105.

[70] Kumar, Sandeep S, Guajardo, Jorge, Maes, Roel, Schrijen, Geert-Jan, and Tuyls, Pim. The butterfly PUF protecting IP on every FPGA. In *2008 IEEE International Workshop on Hardware-Oriented Security and Trust* (2008), IEEE, pp. 67–70.

[71] Leander, G., and Poschmann, A. On the classification of 4 bit s-boxes. In *Proceedings of the 1st International Workshop on Arithmetic of Finite Fields* (Berlin, Heidelberg, 2007), WAIFI '07, Springer-Verlag, pp. 159–176.

[72] Liu, Song, Pattabiraman, Karthik, Moscibroda, Thomas, and Zorn, Benjamin G. Flikker: saving DRAM refresh-power through critical data partitioning. In *Architectural support for programming languages and operating systems* (June 2011).

[73] Lofstrom, K, and Daasch, WR. IC identification circuit using device mismatch. *International Solid State Circuits Conference* (2000).

[74] Lohrke, Heiko, Tajik, Shahin, Boit, Christian, and Seifert, Jean-Pierre. *No Place to Hide: Contactless Probing of Secret Data on FPGAs*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016, pp. 147–167.

[75] Lorenz, Dominik, Georgakos, Georg, and Schlichtmann, Ulf. Aging analysis of circuit timing considering nbti and hci. In *2009 15th IEEE International On-Line Testing Symposium* (2009), IEEE, pp. 3–8.

[76] Maes, Roel. An accurate probabilistic reliability model for silicon pufs. In *Proceedings of the 15th International Conference on Cryptographic Hardware and Embedded Systems* (Berlin, Heidelberg, 2013), CHES'13, Springer-Verlag, pp. 73–89.

[77] Maiti, Abhranil, Gunreddy, Vikash, and Schaumont, Patrick. A systematic method to evaluate and compare the performance of physical unclonable functions. In *Embedded systems design with FPGAs*. Springer, 2013, pp. 245–267.

[78] Malik, Shweta, Becker, Georg T, Paar, Christof, and Burleson, Wayne P. Development of a layout-level hardware obfuscation tool. In *2015 IEEE computer society annual symposium on VLSI* (2015), IEEE, pp. 204–209.

[79] Massad, Mohamed El, Garg, Siddharth, and Tripunitara, Mahesh V. Integrated circuit (IC) decamouflaging: Reverse engineering camouflaged ics within minutes. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2014* (2015).

[80] Mathew, Sanu, Satpathy, Sudhir, Suresh, Vikram, Anders, Mark, Kaul, Himanshu, Agarwal, Amit, Hsu, Steven, Chen, Greg, Krishnamurthy, Ram, and De, Vivek. A 4fJ/bit delay-hardened physically unclonable function circuit with selective bit destabilization in 14nm tri-gate CMOS. In *2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits)* (2016), IEEE, pp. 1–2.

[81] Mazur, RG, and Dickey, DH. A spreading resistance technique for resistivity measurements on silicon. *Journal of the electrochemical society 113*, 3 (1966), 255–259.

[82] Merli, Dominik, Stumpf, Frederic, and Sigl, Georg. Protecting PUF error correction by codeword masking. *IACR Cryptology ePrint Archive 2013* (2013), 334.

[83] Mistry, K., Allen, C., Auth, C., Beattie, B., Bergstrom, D., Bost, M., Brazier, M., Buehler, M., Cappellani, A., Chau, R., Choi, C. H., Ding, G., Fischer, K., Ghani, T., Grover, R., Han, W., Hanken, D., Hattendorf, M., He, J., Hicks, J., Huessner, R., Ingerly, D., Jain, P., James, R., Jong, L., Joshi, S., Kenyon, C., Kuhn, K., Lee, K., Liu, H., Maiz, J., McIntyre, B., Moon, P., Neirynck, J., Pae, S., Parker, C., Parsons, D., Prasad, C., Pipes, L., Prince, M., Ranade, P., Reynolds, T., Sandford, J., Shifren, L., Sebastian, J., Seiple, J., Simon, D., Sivakumar, S., Smith, P., Thomas, C., Troeger, T., Vandervoorn, P., Williams, S., and Zawadzki, K. A 45nm logic technology with high-k+metal gate transistors, strained silicon, 9 Cu interconnect layers, 193nm dry patterning, and 100% Pb-free packaging. In *2007 IEEE International Electron Devices Meeting* (Dec 2007), pp. 247–250.

[84] Mohapatra, D., Chippa, V.K., Raghunathan, A., and Roy, K. Design of voltage-scalable meta-functions for approximate computing. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011* (March 2011), pp. 1–6.

[85] Nirmala, I. R., Vontela, D., Ghosh, S., and Iyengar, A. A novel threshold voltage defined switch for circuit camouflaging. In *2016 21th IEEE European Test Symposium (ETS)* (May 2016), pp. 1–2.

[86] Nohl, Karsten, Evans, David, Starbug, and Plötz, Henryk. Reverse-engineering a cryptographic RFID tag. In *USENIX security* (2008), vol. 28.

[87] Onabajo, Marvin, and Silva-Martinez, Jose. Process variation challenges and solutions approaches. In *Analog circuit design for process variation-resilient systems-on-a-chip*. Springer, 2012, pp. 9–30.

[88] Polak, A C, Dolatshahi, S, and Goeckel, D L. Identifying Wireless Users via Transmitter Imperfections. *Selected Areas in Communications, IEEE Journal on 29*, 7 (2011), 1469–1479.

[89] Rahman, M. Tanjidur, Rahman, M. Sazadur, Wang, Huanyu, Tajik, Shahin, Khalil, Waleed, Farahmandi, Farimah, Forte, Domenic, Asadizanjani, Navid, and Tehranipoor, Mark. Defense-in-depth: A recipe for logic locking to prevail. *CoRR abs/1907.08863* (2019).

[90] Rahmati, Amir, Hicks, Matthew, Holcomb, Daniel E., and Fu, Kevin. Probable cause: The deanonymizing effects of approximate DRAM. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture* (New York, NY, USA, 2015), ISCA '15, ACM, pp. 604–615.

[91] Rajendran, Jeyavijayan, Pino, Youngok, Sinanoglu, Ozgur, and Karri, Ramesh. Logic encryption: A fault analysis perspective. In *Proceedings of the Conference on Design, Automation and Test in Europe* (2012), DATE '12, pp. 953–958.

[92] Rajendran, Jeyavijayan, Sam, Michael, Sinanoglu, Ozgur, and Karri, Ramesh. Security analysis of integrated circuit camouflaging. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), ACM, pp. 709–720.

[93] Sampson, A, Nelson, J, Strauss, K, and Ceze, L. Approximate Storage in Solid-State Memories. *IEEE Micro* (2013).

[94] Schellenberg, Falk, Finkeldey, Markus, Gerhardt, Nils, Hofmann, Martin, Moradi, Amir, and Paar, Christof. Large laser spots and fault sensitivity analysis. *HOST* (2016).

[95] Sekanina, Lukas, and Vasicek, Zdenek. A SAT-based fitness function for evolutionary optimization of polymorphic circuits. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2012), IEEE, pp. 715–720.

[96] Selmke, Bodo, Brummer, Stefan, Heyszl, Johann, and Sigl, Georg. Precise Laser Fault injections into 90 nm and 45 nm SRAM-cells. In *Smart Card Research and Advanced Applications (CARDIS) 2015*, Lecture Notes in Computer Science. Springer International Publishing, 2015.

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

[98] Shiozaki, Mitsuru, Hori, Ryohei, and Fujino, Takeshi. Diffusion programmable device : The device to prevent reverse engineering. *IACR Cryptology ePrint Archive 2014* (2014), 109.

[99] Su, Y, Holleman, J, and Otis, Brian. A 1.6 pj/bit 96% stable chip-ID generating circuit using process variations. *International Solid State Circuits Conference* (2007).

[100] Subramanyan, P., Tsiskaridze, N., Li, W., Gascón, A., Tan, W. Y., Tiwari, A., Shankar, N., Seshia, S. A., and Malik, S. Reverse engineering digital circuits using structural and functional analyses. *IEEE Transactions on Emerging Topics in Computing 2*, 1 (March 2014), 63–80.

[101] Subramanyan, Pramod, Ray, Sayak, and Malik, Sharad. Evaluating the security of logic encryption algorithms. In *Hardware-Oriented Security and Trust (HOST)* (2015).

[102] Sugawara, Takeshi, Suzuki, Daisuke, Fujii, Ryoichi, Tawa, Shigeaki, Hori, Ryohei, Shiozaki, Mitsuru, and Fujino, Takeshi. Reversing stealthy dopant-level circuits. *Journal of Cryptographic Engineering 5*, 2 (2015), 85–94.

[103] Suh, G Edward, and Devadas, Srinivas. Physical unclonable functions for device authentication and secret key generation. In *2007 44th ACM/IEEE Design Automation Conference* (2007), IEEE, pp. 9–14.

[104] SypherMedia. Syphermedia library – circuit camouflage technology. `http://www.smi.tv/SMI_SypherMedia_Library_Intro.pdf`, 2012. Online; accessed 21-April-2015.

[105] Tavana, M. K., Kulkarni, A., Rahimi, A., Mohsenin, T., and Homayoun, H. Energy-efficient mapping of biomedical applications on domain-specific accelerator under process variation. In *Low Power Electronics and Design (ISLPED), 2014 International Symposium on* (Aug 2014), pp. 275–278.

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

[107] Torrance, Randy, and James, Dick. The state-of-the-art in semiconductor reverse engineering. In *Proceedings of the 48th Design Automation Conference* (New York, NY, USA, 2011), DAC '11, ACM, pp. 333–338.

[108] Usmani, Mohammad A, Keshavarz, Shahrzad, Matthews, Eric, Shannon, Lesley, Tessier, Russel, and Holcomb, Daniel E. Efficient puf-based key generation in fpgas using per-device configuration. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 27*, 2 (2018), 364–375.

[109] Valamehr, Jonathan, Chase, Melissa, Kamara, Seny, Putnam, Andrew, Shumow, Dan, Vaikuntanathan, Vinod, and Sherwood, Timothy. Inspection resistant memory: architectural support for security from physical examination. In *ACM SIGARCH Computer Architecture News* (2012), vol. 40, IEEE Computer Society, pp. 130–141.

[110] Van Woudenberg, Jasper GJ, Witteman, Marc F, and Menarini, Federico. Practical optical fault injection on secure microcontrollers. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2011 Workshop on* (2011), IEEE, pp. 91–99.

[111] Venkataramani, Swagath, Chippa, Vinay K., Chakradhar, Srimat T., Roy, Kaushik, and Raghunathan, Anand. Quality programmable vector processors for approximate computing. In *Proceedings of the 46th International Symposium on Microarchitecture* (2013), MICRO-46, pp. 1–12.

[112] Venkatesan, R, Agarwal, A, Roy, K, and Raghunathan, A. MACACO: Modeling and analysis of circuits for approximate computing. In *International Conference on Computer-Aided Design (ICCAD)* (2011), pp. 667–673.

[113] Wang, Huanyu, Forte, Domenic, Tehranipoor, Mark M, and Shi, Qihang. Probing attacks on integrated circuits: Challenges and research opportunities. *IEEE Design & Test 34*, 5 (2017), 63–71.

[114] Xie, Yang, and Srivastava, Ankur. Mitigating sat attack on logic locking. In *Cryptographic Hardware and Embedded Systems-CHES 2016*. Springer, 2016.

[115] Xu, Xiaolin, Keshavarz, Shahrzad, Forte, Domenic J, Tehranipoor, Mark M, and Holcomb, Daniel E. Bimodal oscillation as a mechanism for autonomous majority voting in PUFs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 26*, 11 (2018), 2431–2442.

[116] Yasin, M., and Sinanoglu, O. Transforming between logic locking and IC camouflaging. In *2015 10th International Design Test Symposium (IDT)* (Dec 2015), pp. 1–4.

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

[118] Yasin, Muhammad, Rajendran, Jeyavijayan JV, Sinanoglu, Ozgur, and Karri, Ramesh. On improving the security of logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 35*, 9 (2015), 1411–1424.

[119] Yu, C., Zhang, X., Liu, D., Ciesielski, M., and Holcomb, D. Incremental SAT-based reverse engineering of camouflaged logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 36*, 10 (Oct 2017), 1647–1659.